KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Juhani Pirinen

MOBIILIMAKSUJEN MAKSUVÄLITYSJÄRJESTELMÄN KEHITYS DRUPAL 8 -SISÄLLÖNHALLINTAJÄRJESTELMÄLLÄ

Opinnäytetyö
Maaliskuu 2017

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Maaliskuu 2017** **Tietojenkäsittelyn koulutusohjelma** Tikkarinne 9 80220 JOENSUU

Tekijä(t)
Juhani Pirinen

Nimeke
Mobiilimaksujen maksuvälitysjärjestelmän kehitys Drupal 8 -sisällönhallintajärjestelmällä

Toimeksiantaja
Mediayhtiö B105 Oy

Tiivistelmä

Opinnäytetyöraportissa kuvataan Drupal-sisällönhallintajärjestelmällä toteutettu web-sovelluskehitysprojekti back-end-kehittäjän näkökulmasta. Raportin kohderyhmänä ovat web-kehittäjät, jotka tahtovat tutustua Drupal 8 back-end-sovelluskehitykseen ja joilla on jo vähän kokemusta PHP-olio-ohjelmoinnista ja Drupalin käytöstä web-sivustojen rakentamiseen. Opinnäytetyöraportissa ei kuitenkaan julkisteta projektissa syntynyttä ohjelmakoodia, vaan työn painopiste on ohjelmistokehitysprosessin kuvaamisessa.

Raportin teoreettisessa osuudessa perehdytään Drupal 8:n toimintaan ja rakenteeseen. Raportin toiminnallisessa osuudessa kuvataan vaihe vaiheelta toimeksiantona saatu maksuvälitysjärjestelmän prototyypin suunnittelu ja toteutus. Maksuvälitysjärjestelmä integroi suomalaisten mobiilioperaattoreiden yhteisen Mobiilimaksu-maksutavan ja tarjoaa REST-rajapinnan maksutavan käyttämiseksi mobiili- ja websovelluksissa.

Opinnäytetyön tuloksena ymmärrys Drupal 8 -tietojärjestelmästä ja sen käyttömahdollisuuksista asiakas-palvelin-järjestelmänä osana monipuolista API-ekosysteemiä syventyi merkittävästi. Projektissa syntynyt prototyyppi oli toiminnallisuuksiltaan melko vaatimaton, mutta dokumentaatioineen se tarjoaa hyvän lähtökohdan jatkaa projektia aina tuotantokäyttöön soveltuvaksi maksuvälitysjärjestelmäksi asti.

Kieli	Sivuja
suomi	Liitteet

Asiasanat

maksupalvelut, systeemityö, verkkomaksaminen, verkko-ohjelmointi

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
March 2017
Degree Programme in
Business Information Technology

Tikkarinne 9
80220 JOENSUU
FINLAND

Author (s)
Juhani Pirinen

Title
Development of Payment Gateway for Mobile Payments with Drupal 8 Content Management System

Commissioned by
Mediayhtiö B105 Oy

Abstract

The aim of this thesis is to describe a web application development project with Drupal content management system of back-end developer's point of view. The target group of the report is web developers, who want to know Drupal 8 back-end application development and who already have some experience with PHP object oriented programming and using Drupal for building websites. The program code generated in the project is not published, but focus of the work is in describing the software development process.

Theoretical part of the report is focusing on operation and structure of Drupal 8. Functional part of the report is describing phase by phase the design and implementation of the commissioned payment gateway prototype. The payment gateway integrates Finnish mobile operators' joint Mobile Payment method and provides a REST interface to make use of the payment method in mobile and web applications.

As a result of the thesis understanding deepened significantly about Drupal 8 information system and possibilities of its use as a client-server system as a part of versatile API ecosystem. The prototype that was formed in the project had rather modest functionalities, but with its documentations it offers a good groundwork to continue the project all the way to a payment gateway suitable to production use.

Language	Pages
Finnish	Appendices

Keywords

online payment, payment services, systems development, web programming

Sisältö

Käsitteet

AM	Engl. Agile Modelling eli Ketterä mallinnus on ohjelmiston mallinnusta painottava ketterä projektinhallintamenetelmä (Ambler 2016a).
API	Engl. Application Programming Interface, sovellusohjelmointirajapinta (API University 2016).
Autentikointi	Autentikointi on prosessi, jossa tarkistetaan käyttäjän pääsytiedot (esimerkiksi käyttäjätunnus ja salasana), joilla käyttäjä todistaa identiteettinsä (Siriwardena 2014, 18).
Back-end	Ohjelmiston ”takapää” eli palvelinpuolen osuus ohjelmistosta; vastaparina front-endille (Octal Info Solution 2016).
CMF	Engl. Content Management Framework, sisällönhallintakehys (Symfony CMF 2016).
CMS	Engl. Content Management System, sisällönhallintajärjestelmä (Beal 2016).
Contrib	Lyhenne sanasta engl. ”contributed”. Pääasiassa Drupal.org-sivustolla julkaistut teema- ja moduuliprojektit, jotka eivät ole osa Drupal-ydintä. (Drupal.org 2017a.)
Core	Kaikki tiedostot ja moduulit, jotka ovat osa Drupal-ydintä (Drupal.org 2017a).
Custom	Lyhenne sanasta engl. ”customized”. Drupal-moduuli, -teema tai -ominaisuus, joka on tehty vain tiettyä käyttöä tai organisaatiota varten ja jota ei ole julkaistu contrib-moduulina tai -teemana.
Debuggaus	Ohjelmakoodin virheiden etsiminen ja korjaaminen (engl. debug) (Webopedia 2017a).
EPIC	Epiikka eli hyvin suuri käyttäjätarina (Cobb 2015, 67).
FR	Engl. Functional Requirement eli toiminnallinen vaatimus.
Front-end	Ohjelmiston ”etupää” eli käyttöliittymäpuolen osuus ohjelmistosta; vastaparina back-endille (Octal Info Solution 2016).
HTTP	Engl. Hypertext Transfer Protocol, sovellustason tiedonsiirtoprotokolla (IETF 1999).
HTTPS	Engl. Hypertext Transfer Protocol Secure, sovellustason suojattu tiedonsiirtoprotokolla (IETF 1999).
IDE	Engl. Integrated Development Environment. Ohjelmistokehitysympäristö, joka on suunniteltu sisältämään kaikki ohjelmistokehityksessä tarvittavat ominaisuudet, kuten editori, kääntäjä ja debuggeri (Veracode 2017).
IP-osoite	Engl. Internet Protocol. Yksilöllinen internet-osoite, joka identifioi laitteen internetissä tai lähiverkossa (TechTerms 2017a).
JSON	Engl. JavaScript Object Notation on ECMA-404 standardin mukainen kevyt tekstimuotoinen kieliriippumaton formaatti tiedonvälitykseen (ECMA 2013).
Git	Maksuton avoimen lähdekoodin hajautettu versionhallintajärjestelmä (Software Freedom Conservancy 2017).
Kauppias	Tässä opinnäytetyössä ”kauppiaaksi” kutsutaan verkkokauppaa, web-sovellusta tai natiivia mobiilisovellusta ylläpitävää tahoa, joka on integroinut palveluunsa Mobiilimaksu-maksutavan maksuvälitysjärjestelmää käyttäen.
Kehittäjä	Kehittäjä (engl. developer) on henkilö, joka suunnittelee, ohjelmoi ja muokkaa sovelluksia (Career Centre 2016).
Kehys	Engl. framework. Alusta sovellusten kehittämiselle (TechTerms 2017b).

Ketterät menetelmät	Engl. agile methodology. Asiakaslähtöinen, iteratiivinen, yhteistyöhön ja vuorovaikutukseen pyrkivä projektinhallintamenetelmä (Wrike 2017).
Käyttäjä	Tässä opinnäytetyössä ”käyttäjäksi” kutsutaan ketä tahansa maksuvälitysjärjestelmää tavalla tai toisella käyttävää tahoa.
LOC	Engl. Lines Of Code, ohjelmakoodirivien määrä.
Loppukäyttäjä	Tässä opinnäytetyössä ”loppukäyttäjäksi” kutsutaan mobiilioperaattorin liittymäasiakasta, joka maksaa Mobiilimaksu-maksutavalla.
Olio-ohjelmointiparadigma	Engl. object oriented programming paradigm. Ohjelmointitapa, jossa ohjelmiston suunnittelu perustuu modulaaristen ja uudelleenkäytettävien olioiden väliselle vuorovaikutukselle (Tutorialspoint 2017a).
Operaattori	Tässä opinnäytetyössä ”operaattoriksi” kutsutaan mobiilioperaattoria, joka tarjoaa Mobiilimaksu-maksutavan.
Perintökoodi	Engl. legacy code on sovelluksen lähdekoodia, jota ei enää tueta tai kehitetä, vaan korkeintaan paikataan (Techopedia 2017a).
REST	Engl. Representational State Transfer. HTTP-protokollaa käyttävä arkkitehtuurityyli asiakas-palvelin-järjestelmien väliseen viestintään internetissä yhtenäistä rajapintaa käyttäen. REST-tyyli identifioi käytettävät resurssit URI-polun perusteella. (Fielding 2000.)
Maksusilta	Engl. payment gateway eli maksuvälitysjärjestelmä on kauppiaille tarjottava palvelu maksujen prosessointia varten.
MSISDN	Engl. Mobile Station International Subscriber Directory Number eli matkapuhelinnumero (Beal 2017).
MVP	Engl. Minimum Viable Product on tuote, jossa on vain tärkeimmät ydinomaisuudet, jotka riittävät siihen, että se täyttää tarkoituksensa (Scavarda 2011, 212).
NFR	Engl. Non-Functional-Requirement eli ei-toiminnallinen vaatimus.
Päätepiste	Engl. Endpoint eli palvelimen URL, josta asiakkaalla on pääsy REST-resurssiin.
Scrum	Projektinhallintaprosessimalli ohjelmistokehityksessä (Scrum.org 2017).
SIM	Engl. Subscriber Identification Module eli SIM-kortti on matkapuhelimissa käytetty mikropiiri (ComputerHope 2017).
SOAP	Engl. Simple Object Access Protocol on XML-pohjainen viestintäprotokolla informaation vaihtamiseen järjestelmien välillä, johon se voi käyttää HTTP-protokollaa(Tutorialspoint 2017b).
Suunnittelumalli	Engl. design pattern. Sunnittelumallit edustavat yleisiä ohjelmointiongelmia ratkaisevia parhaita käytäntöjä, jotka ovat syntyneet pitkän ajan kuluessa yrityksen ja erehdyksen kautta useiden ohjelmoijien kokemuksen pohjalta (Tutorialspoint 2017c).
UML	Engl. Unified Modeling Language on standardoitu ohjelmistojen rakenteen ja toiminnan määrittely- ja kuvailukieli (Object Management Group 2005).
URI	Engl. Uniform Resource Identifier määrittelee resurssin polun web-palvelimella. Polkuun ei siis kuulu protokolla kuten HTTP tai HTTPS, eikä isäntänimi. (Techterms 2017c.)
URL	Engl. Uniform Resource Locator määrittelee resurssin www:ssä osoitteen, joka koostuu protokollasta, isäntänimestä ja polusta (Indiana University 2014).
Valmisohjelmakoodi	Engl. boilerplate code on käyttövalmista ohjelmakoodia, jota voi käyttää uudelleen useita kertoja ilman, että alkuperäiseen koodiin välttämättä tarvittaisiin muutoksia (Techopedia 2017b).
Web services	Standardoitu tapa integroida web-sovelluspalveluita internetin välityksellä (Webopedia 2017b).

XP	Engl. Extreme Programming, ketterä projektinhallintamenetelmä (Wells 2013).
Ylläpitäjä	Tässä opinnäytetyössä ”ylläpitäjäksi” kutsutaan tahoa, joka ylläpitää maksuvälitysjärjestelmää.

1 Johdanto

Drupal on sisällönhallintajärjestelmä, joka soveltuu websivujen ja -sovellusten toteuttamiseen. Se tarjoaa web-sisältöjen ja -rakenteiden hallintaan monipuoliset työkalut. Järjestelmää pidetään suorituskyvyltään luotettavana ja tietoturvatasoltaan hyvänä. Drupal on hyvin joustava ja modulaarinen. Sen ytimen toiminnallisuuksia voi laajentaa moduuleilla ja teemoilla. Se on mahdollista integroida toisiin palveluihin ja sovelluksiin, osaksi olemassa olevaa infrastruktuuria. Drupal on avointa lähdekoodia ja kirjoitettu PHP-kielellä. (Drupal.org 2016a.)

Uusin Drupal-pääversio, Drupal 8, on rakennettu Symfony-web-sovelluskehyksen päälle (Byron & McGuire 2016, 23). Drupalia voidaan pitää sekä sisällönhallintajärjestelmänä että sisällönhallintakehyksenä. Sisällönhallintakehys (engl. Content Management Framework, CMF) on sisällönhallintajärjestelmän ja sovelluskehyksen välimuoto, jonka päälle on tarvittaessa mahdollista rakentaa oma räätälöity web-sovellus hyödyntäen kehyksen ohjelmointirajapintoja (engl. Application Programming Interface, API). (Drupal.org 2016b.)

Drupal 8:n ytimessä on vahva tuki REST-pohjaisille web-rajapinnoille (Drupal.org 2016c). Sisältö on Drupalissa palvelu, joka on tarvittaessa minkä tahansa toisen sovelluksen käytössä (Drupal.com 2017). Drupal-projektin johtaja Dries Buytaert kirjoittaa blogissaan (Buytaert 2016a) Drupalin voivan toimia API-first back-end-alustana niin selainpohjaisille sovelluksille kuin natiiveille sovelluksille, esimerkiksi mobiilisovelluksille, kauppojen asiointipäätteille tai jopa lentokoneiden viihdejärjestelmille. Web services -integrointitapa mahdollistaa sisällönhallintajärjestelmässä mm. sen, että järjestelmän back-end ja front-end voidaan kytkeä irti (engl. decouple) toisistaan, jolloin front-end voidaan toteuttaa eri teknologialla ja eri järjestelmässä, esimerkiksi JavaScript-pohjaisella kehyksellä. Drupal voi toimia myös itse front-end-järjestelmänä jollekin toiselle back-end-järjestelmälle. (Pantheon 2017). Drupalia voidaan siis käyttää osana laajempaakin web-API-ekosysteemiä, jossa Drupal voi integroitua samanaikaisesti useiden front-end- ja back-end-järjestelmien kanssa.

1.1 Mobiilimaksujen maksuvälitysjärjestelmä

Mobiilimaksu on suurimpien suomalaisten mobiilioperaattorien yhteinen uusi maksutapa, jossa suomalaisen mobiili-internetyhteyden käyttäjät tunnistetaan suoraan SIM-kortin perusteella ja tehdyt ostokset veloitetaan jälkikäteen puhelinlaskulla tai vähennetään prepaid-liittymän saldosta. Mobiilimaksun käyttäjälle maksamistapahtuma on aina samanlainen operaattorista riippumatta. Mobiilimaksu on tarkoitettu tuotteiden ja palveluiden maksamiseen nopeasti ja helposti, ensisijaisesti mobiililaitteita käyttäen. (Teleforum 2016a; Vaittinen 2016.)

Drupal 8:n monipuolisista käyttömahdollisuuksista opinnäytetyössä kuvataan web-sovelluskehitysprojekti, jossa suunnitellaan ja toteutetaan toimeksiantona saatu mobiilimaksujen maksuvälitysjärjestelmän prototyyppi. Maksuvälitysjärjestelmä eli maksusilta (engl. payment gateway) on palvelinpuolen web-sovellus. Se tarjoaa mobiilisovellusten, verkkokauppojen ja muiden internetiin kytkeytyneiden natiivien tai selainpohjaisten sovellusten käyttöön Mobiilimaksu-maksutavan yhden ja yhtenäisen REST API:n kautta, joka toimii yhdyskäytävänä suomalaisten mobiilioperaattoreiden erilaisiin ja toisistaan erillisiin mobiilimaksu-API:hin.

1.2 Opinnäytetyön tavoite, painopiste ja kohderyhmä

Opinnäytetyön tavoitteena on esitellä Drupal 8:n rakennetta ja toimintaa sekä kuvata ja selittää Drupalia toteutusalustana käyttävän web-sovelluskehitysprojektin suunnittelu- ja työskentelyprosessia. Drupalia ja vastaavia sisällönhallintajärjestelmiä käytetään yleensä web-sivujen rakentamiseen, jolloin projekti koostuu tyypillisesti sivuston rakenteiden teosta, ulkoasun valmistamisesta, moduuleiden valinnasta ja kustomoinnista sekä sisällöntuotannosta. Opinnäytteessä painopiste on web-suunnittelun sijaan web-sovelluskehityksessä. Drupal-sovelluskehityksessä hyödynnetään back-end-ohjelmointityön lisäksi ohjelmistotuotannon, sovellussuunnittelun ja projektinhallinnan käytänteitä.

Opinnäytetyön kohderyhmänä ovat web-kehittäjät, jotka tahtovat tutustua Drupal 8 back-end-sovelluskehitykseen teoriassa ja käytännössä ja joilla on jo jon-

kin verran kokemusta PHP-olio-ohjelmoinnista ja Drupalin käytöstä web-sivustojen rakentamiseen. Monet kehittäjät kokevat Drupalin oppimisen rankaksi ja pitkäksi prosessiksi. Kysymys on osittain uskomuksesta, mutta asiassa on myös perää. Viking Code School -koulutusyrityksen blogissa (Trautman 2015) esitetään kehittäjäksi oppimisen jakautuvan neljään vaiheeseen. *Ensimmäisessä* vaiheessa kehittäjä tutustuu oppimateriaalien ja harjoitustehtävien avulla tehtäväkenttään ja kehittäjän itseluottamus omaa osaamista kohtaan alkaa nousta nopeasti. *Toisessa* vaiheessa kehittäjä alkaa soveltaa opittua omassa työssään, mikä tuntuu haastavalta, sillä ratkaisuja ongelmiin on opittava löytämään itse ja niitä ei löydy helposti. Tämä saa kehittäjän itseluottamuksen nopeasti laskemaan, kun työ ei edisty. *Kolmannessa*, edellisiä paljon pidemmässä vaiheessa kehittäjä on täynnä jokseenkin epämääräisiä kysymyksiä ja vaeltaa ”epätoivon autiomaassa” etsien ratkaisuja – työskentely on hidasta ja kehittäjän itseluottamus on vähäinen. *Neljännessä* vaiheessa kehittäjän tietomäärä ja ymmärrys ovat jo kasvaneet merkittävästi. Kysymyksenasettelu tulee relevantiksi, ongelmanratkaisukyky paranee ja vastaukset löytyvät nopeammin. Työskentely alkaa sujua ja kehittäjän itseluottamus nousee kokemuksen myötä merkittävästi. Toivonkin, että tämä opinnäytetyö saattaisi hyvässä tapauksessa pystyttää Drupal-kehittäjäksi aikoville muutamia tienviittoja, joita voisi käyttää oman kartan ja kompassin kanssa apuna ”autiomaassa” päämäärään pääsemiseksi.

1.3 Opinnäytetyön rakenne

Opinnäytetyöraportti jakautuu karkeasti ottaen kahteen osaan: Drupalin ja Drupal-projektin kuvaamiseen. Teoreettisessa Drupal-osassa keskitytään Drupal 8:n coren eli ytimen kuvaamiseen, josta esitellään perusteet, tärkeimpiä pohja- ja alijärjestelmiä ja moduuleita sekä lyhyesti muuta ytimeen liittyvää.

Toiminnallinen projekti-osa on jakautunut esitutkimus-, suunnittelu-, toteutus- ja katselmointivaiheisiin. Vaiheesta toiseen on edetty pääsääntöisesti lineaarisesti, mutta opinnäytetyöraportin kirjoittamisen, tehtävänä olevan projektin kokeellisen luonteen, ketterien työmenetelmien soveltamisen ja työskentelyn aikana saatujen lisätietojen ja oivallusten vuoksi lähes jokaisen vaiheen tietosisältöä ja

kuvausta on täydennetty ja syvennetty useaan otteeseen. Lopuksi esitetään yhteenveto ja pohditaan opinnäytetyötä kokonaisuutena.

1.4 Projektin rajaaminen

Toimeksiantajan toiveesta projektissa oli lähtökohtana MVP- eli ”pienin mahdollinen tuote”-lähestymistapa (engl. Minimum Viable Product): ohjelmiston suunnittelu ja toteutus oli siis rajattava pienimpään mahdolliseen ominaisuusjoukkoon, joka riittäisi ohjelmiston ottamiseksi tuotantokäyttöön (Cobb 2015, 29). Valitettavasti kaikkea MVP-julkaisuun tarvittavaa ei kuitenkaan pystytty sisällyttämään opinnäytetyöhön. Projektin työläys vaati voimakkaampaa aiheen rajaamista, koska opinnäytetyön tekemiseen oli käytettävissä vain hyvin rajalliset resurssit. Karsintaa tehtiin voimakkaimmin mahdollisesti prototyypin laatuvaatimuksista (mm. ohjelmistotestaus ja tietoturva), mutta myös toiminnallisia vaatimuksia rajattiin opinnäytteen ulkopuolelle. Työssä keskityttiin vain maksuvälitysjärjestelmän ydinominaisuuksiin. Rajaamista tehtiin sekä opinnäytetyöprosessin alussa että sen aikana.

Projektia rajattiin opinnäytetyössä seuraavasti:

- Työssä käsiteltiin vain teknistä suunnittelua ja toteutusta sekä projektinhallintaa. Työssä ei käsitelty liiketoimintapuoleen liittyviä asioita, paitsi jos ne koskettivat myös prototyypin teknistä toteutusta.
- Työssä toteutettiin vain maksuvälitysjärjestelmän back-end vain esi-alfatason prototyypin laajuudessa. Mobiilimaksu-napin sisältävää testisivua lukuun ottamatta ei suunniteltu mitään ohjelmistokirjastoja tai käyttöohjeita, joiden avulla palveluntarjoajat voisivat integroida maksuvälitysjärjestelmän omaan mobiilisovellukseen tai verkkokauppaan.
- Integroitavaksi otettiin vain yksi mobiilioperaattori, jonka API tuntui olevan helpoiten otettavissa käyttöön ja jonka kanssa yhteistyö sujui parhaiten.
- Yhden mobiilioperaattorin mobiilimaksu-API:sta integroitiin vain yksinkertainen ”yhden askeleen veloitus”-maksutoiminnallisuus. Työssä ei toteu-

tettu mm. monimutkaisempaa katevarauksen tekevää maksutoiminnallisuutta, joka mahdollistaisi maksun veloittamisen tai perumisen myöhemmin.

- Työssä ei suunniteltu maksuvälitysjärjestelmän ylläpitoliittymää. Drupalytimen tarjoamaa hallintaliittymää käyttäen ylläpitäjä voi kuitenkin määritellä mm. maksuvälitysjärjestelmää käyttävien sovellusten asetuksia ja tarkastella järjestelmää ajettaessa syntyviä entiteettejä.
- Työssä ei toteutettu maksupalveluliiketoiminnassa tarvittuja ominaisuuksia, kuten API:n käytön maksatus ja laskutus, virhemaksujen selvittely- ja hyvittämistoiminnot, toistuvat maksut ja niiden peruminen.
- Työssä ei toteutettu maksuvälitysjärjestelmän tietoturvaominaisuuksia. Järjestelmä vaati erittäin korkeaa tietoturvatasoa, joka olisi edellyttänyt syvällistä paneutumista tietoturvakysymyksiin. Esimerkiksi ei toteutettu kaavailtua OAuth 2.0 -autentikaatiota maksuvälitysjärjestelmän API:ssa ja tingittiin myös syötteiden tarkistuksissa.
- Työssä ei tehty ohjelmistotestausta. Tämä on ehdottoman välttämätön laatutekijä ohjelmistotuotannossa ja Drupal-sovelluskehityksessä, mutta se jouduttiin rajaamaan pois opinnäytetyön laajuuden vuoksi.
- Työssä ei käsitelty mitään maksuvälitysjärjestelmän operointiin liittyvää kuten tuotantoympäristössä ajettavan järjestelmän ylläpitoa tai tuotantoympäristön toteuttamista.
- Työssä käytetään vain Drupal-versiota 8.2.x ja PHP-versiota 7.0.x, ellei raportissa ole toisin mainittu.

2 Drupal

2.1 Tietojärjestelmät, kehykset ja sisällönhallintajärjestelmät

Tietojärjestelmää voi luonnehtia ihmisistä, tietojenkäsittelylaitteista, tiedonsiirtolaitteista ja ohjelmistoista koostuvaksi järjestelmäksi, jonka tarkoitus on helpottaa ja tehostaa tai tehdä mahdolliseksi jokin toiminta. Eri tarkoituksia varten on olemassa useita erityyppisiä tietojärjestelmiä, esimerkiksi toiminnanohjausjärjestelmät (engl. Enterprise Resource Planning, ERP), asiakkuudenhallintajärjestelmät (engl. Customer Relations Management, CRM) ja sisällönhallintajärjestelmät (engl. Content Management System, CMS). (Mantere 2014, 1–2, 24, 33.)

Tietojärjestelmät rakennetaan usein sellaista sovelluskehystä käyttäen, joka tarjoaa perusrakenteen sovelluskehitykselle jossain tietyssä ympäristössä (Techopedia 2016). Esimerkiksi web-sovelluskehys mahdollistaa web-sovellusten, kuten sisällönhallintajärjestelmän, kehittämisen web-ympäristöön. Sovelluskehyksen käyttö helpottaa ja nopeuttaa tietojärjestelmän kehitystyötä, sillä se tarjoaa kehittäjälle käyttövalmiit, dokumentoidut, uudelleenkäytettävät, ylläpidetyt, päivitettävät komponentit, jotka ovat testattuja, vakaita ja korkealaatuisia ja joita voi laajentaa omalla ohjelmakoodilla tai muilla ohjelmistokirjastoilla (Symfony.com 2016a).

Sisällönhallintajärjestelmät ovat tietojärjestelmiä, jotka tarjoavat työkalut sisällön luomiseen ja julkaisemiseen websivustolla. Sisältö voi tarkoittaa mitä tahansa, kuten uutisartikkelia, blogikirjoitusta, kuvaa tai videota, jonka käyttäjä voi lukea, nähdä tai kuulla. Sisällönhallintajärjestelmien ominaisuuksiin kuuluu tyypillisesti mm. hallintaliittymä sisällön, toimintojen ja käyttäjien hallintaan, mahdollisuus sisällön luokitteluun ja merkintään, työkalut valikoiden ja navigaation luomiseen, toiminto ulkoasun valintaan ja web-lomakkeet. (Tomlinson 2015, 1.)

2.2 Yleistä Drupalista

2.2.1 Historia

Drupal sai alkunsa vuonna 2000 opiskelijoiden vapaa-ajan projektista, joka oli oman porukan käyttöön tehty keskustelupalstasovellus. Kokeilujen, keskustelujen ja jatkuvan ideoinnin myötä sovelluksesta alkoi vähitellen muotoutua sisällönhallintajärjestelmä. (Drupal.org 2016d.)

Drupalista on tähän mennessä julkaistu kahdeksan pääversiota. Uusin pääversio, Drupal 8, julkaistiin marraskuussa 2015 (Drupal.org 2016e). Verrattuna edelliseen pääversioon 7, Drupalin kokoluokka on kasvanut uusimpaan pääversioon mennessä noin nelinkertaiseksi, lähes 700 000 koodiriviin (LOC, lines of code) (Black Duck Open Hub 2016).

Myös Drupalin käyttö on kasvanut. Esimerkiksi lokakuusta 2012 toukokuuhun 2016 käytössä olevien Drupal-ytimien määrä oli noussut noin 50 %, yli miljoonaan aktiiviseen ytimeen. (Drupal.org 2016f.)

2.2.2 Missio ja perusperiaatteet

Drupal-yhteisön mukaan Drupal-sisällönhallintajärjestelmän missiona on tuottaa avoimeen lähdekoodiin perustuva sisällönhallintaohjelmistokehys, joka edustaa uusimpia ideoita ja parhaita käytäntöjä.

Kehitystyössä noudatetaan perusperiaatteita, joiden mukaan järjestelmä:

- perustuu standardeihin
- on modulaarinen ja helposti laajennettavissa
- on avointa lähdekoodia, joka on korkealaatuista, eleganttia ja hyvin dokumentoitua
- on helppokäyttöinen

- vaatii vain vähän resursseja. (Drupal.org 2016g.)

Drupal-kehitystä pyritään tekemään yhteistyössä, jossa tiedonkulku on avointa (Drupal.org 2016g). Drupal 8:n kehityksessä punaisena lankana on ollut ”Proudly Invented Elsewhere”-periaate, jonka mukaan pyörää ei tarvitse keksiä uudestaan, vaan projekti on avattu parhaiden mahdollisten jo olemassaolevien avoimen lähdekoodin teknologioiden hyödyntämiselle. Samalla on pyritty eroon ”drupalismista”, jolla tarkoitetaan Drupal-kehittäjien keksimiä omaperäisiä ratkaisuja ja erikoisia käytäntöjä, joita ei löydy muista ohjelmistoista ja jotka eivät noudata mitään yleisesti tunnustettua standardia. (Byron & McGuire 2016, 23.)

Drupal-ytimen ja Drupal-ydintä laajentavien ns. contrib-moduuleiden kehityksessä pyritään noudattamaan ohjelmointistandardeja. Keskeisin ohjelmointistandardi lähinnä ottaa kantaa PHP-koodin muotoiluun Drupal-projektissa. Kehittäjä voi käyttää apuna standardin mukaisessa muotoilussa mm. Coder-moduulia tai useisiin koodieditoreihin integroitavaa Code Sniffer -työkalua. Ohjelmointistandardit ottavat myös kantaa mm. olio-ohjelmointitapaan, nimiavaruuksien käyttöön, palveluiden nimeämiseen ja PHP-poikkeuksiin. Lisäksi ohjelmointistandardeissa opastetaan mm. JavaScriptin ja CSS:n käyttöön sekä API-dokumentaatioiden laadintaan. (Drupal.org 2017b.)

2.2.3 Teknologiaperusta

Toimiakseen Drupal tarvitsee käyttöönsä teknologiapinon, johon kuuluvat sen itsensä lisäksi tietoverkkoon kytketty palvelinlaitteisto, palvelinkäyttöjärjestelmä (esim. Linux tai Windows), web-palvelin (esim. Apache tai Microsoft IIS), tietokanta (esim. MySQL tai PostgreSQL) ja PHP-ohjelmointikieli (Drupal.org 2016h). Tarkemmat järjestelmävaatimukset vaihtelevat versioittain, esimerkiksi PHP-ohjelmointikielen versiolle ja kokoonpanolle on paljon yksityiskohtaisia vaatimuksia. Myös Drupal-ytimen sisältämät teknologiat vaihtelevat versioittain. (Drupal.org 2016i.)

Uusin Drupal 8 hyödyntää itsensä lisäksi mm. seuraavia back-end-puolen teknologioita:

- **Symfony.** Drupal on rakennettu Symfony PHP-komponenttien päälle (Byron & McGuire 2016, 23).
- **PHPUnit.** Kirjasto yksikkötestauksen tekemiseen.
- **Composer.** Ulkoisten ohjelmistokirjastojen riippuvuuksien käyttöönottoon. PSR4-standardin mukaiseen luokkien automaattiseen lataamiseen.
- **Guzzle.** PHP HTTP -kirjasto www-sovelluspalveluiden kanssa keskusteluun.
- **YAML.** Käyttäjäystävällinen datan sarjallistamisen standardi (Yaml.org 2016), jota Drupal käyttää mm. erilaisissa konfiguraatiotiedostoissa.

Front-end-puolen teknologioista Drupal 8:n käytössä ovat mm. seuraavat:

- **Twig.** Sivupohjamoottori, joka erottelee esitys- ja liiketoimintalogiikan.
- **JavaScript**-kirjastot ja -apuohjelmat JQuery, JQuery UI, Modernizr, Underscore.js ja Backbone.js. Lisäksi käytössä on CKEditor WYSIWYG-editori sisällön muokkaamiseen.
- **HTML5- ja CSS3-merkintäkielet.** CSS3-tyylit ja semanttinen HTML5 ovat oletusarvoisesti käytössä. Tämän vuoksi Drupal 8 ei enää tue vanhempia selaimia, kuten Internet Explorer 6, 7 ja 8.

(Byron & McGuire 2016, 4–24.)

2.3 Drupal-kehyksen rakenne

Drupal-kehys on rakennettu Drupal- ja Symfony-komponenttien varaan. Drupal Kernelin ja komponenttien päälle on rakennettu Drupalin pohjajärjestelmä, joka jakautuu edelleen alijärjestelmiksi. Drupal-moduulit rakentuvat pohja- ja alijärjestelmien päälle. (Drupal.org 2016j; Symfony.com 2016c.)

Järjestelmän rakennetta voi hahmottaa mm. sen elementtien keskinäisten riippuvuussuhteiden perusteella. Komponenteilla saa olla riippuvuus toisiin Drupal-komponentteihin tai ulkoisiin kirjastoihin ja paketteihin, mutta niillä ei saa olla

riippuvuutta mihinkään muuhun Drupal-koodiin. Alijärjestelmillä saa olla riippuvuus toisiin Drupal-komponentteihin ja toisiin alijärjestelmiin, mutta niillä ei saa olla riippuvuutta Drupal-moduuleihin. (Drupal.org 2016j). Moduulit saavat olla riippuvaisia pohja- ja alijärjestelmien ja toisten moduulien tarjoamista luokista, palveluista ja ohjelmointirajapinnoista sekä mistä tahansa kirjastoista ja paketeista.

Drupalin lähdekoodista on nähtävissä, kuinka Drupal mukauttaa ja optimoi useita Symfony-komponentteja omiin tarpeisiinsa. Useista Symfony-komponenteista on olemassa osatoteutuksia myös saman nimisinä Drupal-komponentteina. Lisäksi jonkin ohjelmistoelementin osatoteutus voi esiintyä samalla nimellä kahdella tai kolmella järjestelmän rakennetasolla (komponentit, alijärjestelmät ja moduulit) (Drupal.org 2016j).

2.4 Drupal-pohjajärjestelmä: Drupal Kernel ja Drupal-komponentit

Drupal Kernel on Drupal-kehyksen sisin ydin, joka rakentaa palvelusäiliön ja huolehtii ympäristön käyttöönotosta mm. lataamalla moduulit ja palvelut. Drupal Kernel delegoi HTTP-pyyntöjen ja vastausten käsittelyn Http Kernel -komponentille. Drupal Kernel toteuttaa kehyksessä yhteistyössä Http Kernelin saman ytimen roolin kanssa kuin Symfony CMF -sisällönhallintakehyksen Kernel, mutta muilta osin kernelien toiminta poikkeaa toisistaan. (Drupal.org 2016k; 2016l; Cipix 2013.)

Drupalin käytössä ovat seuraavat Symfony-komponentit: Class Loader, Console, Css Selector, Dependency Injection, Event Dispatcher, Http Foundation, Http Kernel, Process, Routing, Serializer, Translation, Validator ja Yaml (Symfony.com 2016c). Näiden lisäksi Drupal käyttää omia komponenttejaan, joita Drupal-versiossa 8.2.x ovat Annotation, Assertion, Bridge, Class Finder, Datetime, Dependency Injection, Diff, Discovery, Event Dispatcher, File Cache, File System, Gettext, Graph, Http Foundation, Php Storage, Plugin, Proxy Builder, Render, Serialization, Transliteration, Utility ja Uuid (Drupal.org 2016j; Cgit-.drupalcode.org 2016a).

Seuraavissa alaluvuissa esitellään lyhyesti muutamia keskeisimpiä Drupal-komponentteja, jotka Drupal-kehittäjän on hyvä tuntea järjestelmän toiminnan ymmärtämisen kannalta. Näitä ovat Dependency Injection, Event Dispatcher, Http Foundation ja Http Kernel.

2.4.1 Dependency Injection, palvelusäiliö ja palvelut

BuildAModule (2016) luonnehtii *Depencency Injection -komponenttia* Drupalin selkärangaksi, sillä Drupal-kehyksen toiminta rakentuu sen varaan. Komponentti toteuttaa riippuvuuksien injektointi -suunnittelumallin, joka mahdollistaa järjestelmän eri osien riippuvuuksien hallinnan. Tämä yhdenmukaistaa ja keskittää tavan, jolla oliot on järjestelmässä rakennettu. Käytännössä komponentti toteuttaa tämän riippuvuuksien injektointi -säiliön avulla (engl. Dependency Injection Container), jota Drupalissa ja Symfonyssa kutsutaan *palvelusäiliöksi* (engl. Service Container). (Symfony.com 2016b.)

Palvelusuuntautuneessa arkkitehtuurissa (engl. service-oriented architecture) ohjelmakoodin tarjoamat toiminnallisuudet pyritään eriyttämään sarjaksi *palveluita* (engl. service). Kun PHP-olio on globaali eli järjestelmän laajuinen, sitä kutsutaan palveluksi. Palvelusäiliö mahdollistaa palveluiden toiminnallisuuksien luonnin irtikytketyllä (engl. decoupled) tai löyhästi kytketyllä (engl. loosely coupled) tavalla. Irtikytkeminen helpottaa järjestelmän hallintaa, laajentamista ja ylläpitoa, esimerkiksi yksittäinen palvelu on helposti vaihdettavissa toiseen tai sen toiminta on mukautettavissa ja testattavissa. Kytkettyä (engl. coupled) olio-ohjelmointitapaa pyritään välttämään, koska se luo vahvoja riippuvuuksia järjestelmän eri osien välille; tällöin esim. muutos järjestelmän yhteen luokkaan voi vaatia muutoksen myös niihin luokkiin, jotka ovat siitä riippuvaisia. (Cohn 2014; Drupal.org 2016l; 2016m; PHP-DI 2016; Symfony.com 2016d.)

Palveluita voivat tarjota käytettäväksi niin Drupal-ydin kuin contrib- ja custom-moduulitkin. Palveluilla voi olla riippuvuuksia muihin palveluihin. Drupal-ytimen palvelut on määritelty `CoreServiceProvider.php`- ja `core.services.yml`-tiedostoissa. (Drupal.org 2016l.) Säiliön sisältämät palvelut saa listattua myös

Drupal Consolen komentorivikomennolla `drupal container:debug` (Drupal Console 2016). Palvelut ovat merkittävissä palvelutägeillä (engl. Service Tags) esim. `event_subscriber`, joilla voi ryhmitellä tägejä ja määritellä niiden toimintaa. Palveluluokan on toteutettava tägiä vastaava rajapinta. (Drupal.org 2017c.)

Olio-ohjelmoinnissa riippuvuudet tulee ottaa käyttöön konstruktori- tai setter metodi-injektioina. Property-injektion käyttöä ei suositella. (Symfony.com 2016b.) Konstruktori-injektio (engl. constructor injection) toteutetaan Drupalissa yleensä palveluiden välityksellä, jolloin riippuvuudet määritellään argumentteina `*.services.yml`-tiedostoihin. Toinen suosittu malli konstruktori-injektion tekemiseen Drupalissa on tehdasinjektio (engl. factory injection), jossa riippuvuudet määritellään luokan konstruktoriin (engl. constructor) ja otetaan käyttöön create-tehdasmetodilla. (Bosch 2016.) Molempien konventioiden hyvä hallinta on välttämätöntä Drupal 8 -kehittäjälle.

Drupal-ydin kuitenkin tarjoaa palvelusäiliön käytölle vaihtoehdoksi myös globaalin Drupal-luokan, josta palveluita on kutsuttavissa staattisella `\Drupal::service()`-metodilla. Sen käyttöä ei kuitenkaan suositella, sillä Drupal-luokka on olemassa vain, jotta perinteisemmätkin Drupal-kehittäjät ehtisivät paremmin refaktoroida moduuliensa proseduraalisen perintökoodin (engl. legacy code) Drupalin uuteen oliosuuntautuneeseen koodipohjaan. (Drupal.org 2016o.)

2.4.2 Event Dispatcher, Http Foundation ja Http Kernel

Event Dispatcher -komponentti toteuttaa komponenttien, alijärjestelmien ja moduuleiden keskinäisen vuorovaikutuksen mahdollistamalla reaktiot järjestelmän eri tapahtumiin tapahtumakuuntelijoiden (engl. Event Listener) ja -tilaajien (engl. Event Subscriber) avulla. Näistä jälkimmäinen on hiukan edistyneempi tapa kuuntelijoiden käyttöön. (Symfony.com 2016e.) Tapahtumatilaajat ovat olioiden rekisteröimiä palveluita, jotka on merkitty `event_subscriber`-tägillä ja toteuttavat `EventSubscriberInterface`-rajapinnan. Järjestelmän oliot voivat ti-

lata jonkin tietyn tapahtuman, joka on ensin määritelty jossain toisessa järjestelmän osassa. Tapahtuman toteutuessa siitä lähetetään tieto tilaajille, jotka näin saavat mahdollisuuden reagoida siihen. (Drupal.org 2016p.)

Drupal 8:ssa Event Dispatcher toteuttaa olio-ohjelmointiparadigman mukaisesti saman ominaisuuden kuin proseduraalista ohjelmointitapaa edustavat *koukut* (engl. hook), joten järjestelmässä on rinnakkain kaksi tapaa toteuttaa sama asia. Koukut ovat todennäköisesti poistumassa Drupalista seuraavassa pääversiossa 9. (Drupal.org 2016n.) Drupal version 8.2.x ytimen lähdekoodia tutkittuani en kuitenkaan havainnut varsinaista päällekkäisyyttä eli tiettyyn tapahtumaan reagointiin tarjottaneen mahdollisuus vain joko koukkua tai tapahtumatilaajaa käyttäen, vaikka se periaatteessa onkin mahdollista kumpaakin tapaa käyttäen. Tästä hyvänä esimerkkinä on Hook Event Dispatcher -contrib-moduuli, joka tarjoaa tapahtumatilaajajärjestelmän käyttömahdollisuuden vaihtoehtona useille Drupal-ytimen koukuille (Drupal.org 2016q).

Http Foundation -komponentti tarjoaa oliopohjaisen lähestymistavan HTTP:n monipuoliseen käyttöön turvallisella ja systemaattisella tavalla, mikä pienentää virhemahdollisuuksia ja helpottaa testattavuutta. Komponentin metodeita käyttäen voidaan saada, asettaa, lisätä ja poistaa kaikkia HTTP-pyynnön ja -vastauksen muuttujia, parametrejä ja otsikoita (engl. headers). Lisäksi komponentilta onnistuu pyyntöihin vastaaminen, pyyntöjen simulointi, edelleenohjaukset, evästeiden asettaminen ja käsittely sekä välimuistin hallinta. (Symfony.com 2016f.)

Http Kernel -komponentti käsittelee rakenteisesti HTTP-pyyntöjä ja -vastauksia ja tarjoaa tapahtumapohjaisen työnkulun käsitellä pyyntödataa ja generoida vastausdataa niiden välissä. Se on pidemmälle abstraktiotasolle edennyt komponentti, joka on suunniteltu muodostamaan kehyksen ydin. Http Kernel tarjoaa myös alipyyntöominaisuuden (engl. Sub Request), joka koko sivun sijasta kohdistuu vain pieneen osaan sivusta ja voi näin tuottaa alivastauksena palasen sivusta. Http Kernel perustuu Http Foundation ja Event Dispatcher -komponentteihin ja yhdistää niiden voiman. (Symfony.com 2016g.)

2.5 Drupal-alijärjestelmät

Drupal-versio 8.2.x sisältää 70 alijärjestelmää ytimen luokkakirjastoina, jotka löytyvät Drupal-asennuksen juuresta katsoen polusta `core/lib/Drupal/Core` (Cgit.drupalcode.org 2016b). Osa alijärjestelmien toteutuksista on myös proseduraalisessa ei-modulaarisessa muodossa ytimen liitännäisinä (engl. includes). Useiden liitännäistiedostojen lähdekoodissa mainitaan tiedoston poistosta ennen seuraavaa Drupal-pääversiota 9. (Cgit.drupalcode.org 2016c; Letharion 2015.)

Seuraavissa alaluvuissa esitellään lyhyesti muutamia keskeisimpiä alijärjestelmiä, jotka Drupal-moduulikehittäjän on hyvä tuntea järjestelmän toiminnan ymmärtämisen kannalta. Alijärjestelmät tarjoavat API:eja eli ohjelmointirajapintoja, joita voi hyödyntää omassa contrib- ja custom-moduulikehityksessä.

2.5.1 Config

Konfiguraatio on informaatiota, joka ei ole sisältöä ja vaihtuu harvoin, esimerkiksi web-sivuston nimi tai sisältötyypit (Drupal.org 2016r). *Config-alijärjestelmä* jakautuu Drupalissa kahdeksi API:ksi: *Simple Configuration API* ja *Configuration Entity API* (Drupal.org 2016s). Ensimmäinen on tarkoitettu yksittäisten konfiguraatioarvojen tallennusta varten, kun jälkimmäinen mahdollistaa monimutkaisempien konfiguraatiokokonaisuuksien tallentamisen entiteeteiksi (Drupal.org 2016r). Yksinkertaiset konfiguraatiot eivät ole entiteettejä (Drupal.org 2016t).

Konfiguraatio tallennetaan oletusarvoisesti tietokantaan, mutta tallentaminen on mahdollista myös YAML-konfiguraatiotiedostoiksi (Drupal.org 2016u). Jälkimmäinen mahdollistaa konfiguraation viemisen ohjelmakoodin mukana versionhallintaan ja työnkulun, jossa esimerkiksi kehitys-, staging- ja tuotantoversiot järjestelmästä konfiguraatioineen eriytetään, kaikki muutokset pystytään versioimaan ja jakamaan koko kehitystiimin kesken (Anderson 2015).

2.5.2 Entity ja Field

Entiteetit ovat datan pysyvää säilytystä varten tarkoitettuja olioita, joita on olemassa kahta eri varianttia: sisältö- ja konfiguraatioentiteettejä (Drupal.org 2016v; 2016w). Sisältöentiteetit tukevat mm. kenttiä (engl. field) ja versiointia (Drupal.org 2016w). Konfiguraatioentiteetit eivät näitä tue, mutta toisin kuin sisältöentiteetit, ne integroituvat *Configuration Entity API:iin* ja ne helpottavat monikielisen käyttöliittymän toteuttamista skeema-tiedostoilla (Drupal.org 2016w; 2016x; 2016y). Molemmat entiteettivariantit tukevat kielenkäännöksiä (Drupal.org 2016w).

Entity API mahdollistaa entiteeteille kaikki CRUD-operaatiot (Luo, Lue, Päivitä ja Poista) ja pääsynhallinnan (Drupal.org 2016v; 2016z). Entiteetit toteuttavat *Typed Data API:n*, joka mahdollistaa datatyyppien määrittämisen PHP-kieltä tarkemmin, korjaten näin erään PHP-kielen puutteista (Drupal.org 2016å). *Entity Validation API:n* avulla tarkistetaan, että entiteetteihin syötettävä data vastaa datatyyppimäärittelyjä (Drupal.org 2016ä).

Drupal-moduuleilla voidaan luoda *Plugin API:a* käyttäen jompaa kumpaa entiteettivarianttia edustavia entiteettityyppejä, esimerkiksi sisältöentiteettejä, kuten kuva, käyttäjä tai node, tai konfiguraatioentiteettejä, kuten näkymä (engl. view), rooli ja valikko (engl. menu) (Drupal.org 2016v; 2016ö; 2016aa). Entiteettityypit voivat jakautua edelleen nipuiksi (engl. bundle), esimerkiksi Node-sisältöentiteetti jakautuu sisältötyypeiksi (engl. content type), joita voivat olla vaikkapa artikkeli, perussivu tai uutinen (Drupal.org 2016v; 2016ö). Käytännössä kaikki Drupal-järjestelmän tietosisältö ja kaikki monimutkaiset konfiguraatiot ovat entiteettejä.

Sisältöentiteettien tietorakenteet määritellään kenttiä käyttäen. Drupal-ytimessä on 29 erilaista kenttätyyppiä. (Drupal.org 2017d.) Niput ovat kenttäkokoelmia. Nipuista ja kenttien tietorakenteista vastaa *Field API.* FieldStorage-tietorakenne määrittelee entiteetteihin liitettävät kenttä-datatyypit. Field-tietorakenne määrittelee nippuun liitetyt yksittäiset kentät. *Field Attach API* vastaa kenttien liittämisestä nippuun. (Drupal.org 2017e.)

2.5.3 Plugin

Plugin on yksi Drupalin käytetyin ja joustavin alijärjestelmä (Bosch 2016). Shindelarin (2014) mukaan plugin-järjestelmää voi luonnehtia kokoelmaksi suunnittelumalleja (engl. design pattern), jotka ratkaisevat jonkin tietyn ohjelmointiongelman tietyssä kontekstissa. Drupal-kehittäjän ei tarvitse keksiä pyörää uudestaan, vaan hän voi käyttää toisen Drupal-moduulin tai -alijärjestelmän tarjoamaa hyväksi havaittua ratkaisua.

Pluginit tekevät mahdolliseksi moduuleille ja alijärjestelmille tarjota oliopohjaisella tavalla toisilleen uudelleen käytettäviä toiminnallisuuksia, jotka ovat kapseloituja ja vaihdettavissa olevia. *Plugin API* mahdollistaa jonkin tietyn pienen käyttötapauksen toteuttavien plugin-tyyppien (engl. plugin type) määrittämisen, eri keinoja – kuten koukut, annotaatiot, YAML ja staattinen – pluginien löytämiseksi (engl. plugin discovery) koodipohjasta ja plugin-tehtaan (engl. plugin factory) jonkin tietyn plugin-tyypin vaatimusten mukaisten plugin-ilmentymien (engl. plugin instance) luomiseksi. Jokaisella plugin-tyypillä on palveluksi rekisteröity manageri, jonka kautta plugineja voi löytää ja luoda. Samaa plugin-tyyppiä edustavilla plugin-ilmentymillä on kaikilla sama ID. (Wolanin 2016; Drupal.org 2016ab; 2016ac.) Jotta omien pluginien yksikkötestaus olisi mahdollista, palvelut tulee injektoida pluginiin staattisen `\Drupal::service`-metodin käytön sijaan (Bosch 2016).

2.5.4 Render

Renderöinnillä tarkoitetaan datan generoimista esitettävään muotoon, esim. web-selaimen ymmärtämään HTML-formaattiin (Drupal.org 2016ad). *Render-alijärjestelmä* toteuttaa Drupalin teemajärjestelmän, jonka tarkoitus on antaa teemoille ulkoasun täysi hallinta ja palvelunlaajuinen yhtenäinen käyttöliittymä. Tähän päästään, kun moduulikehittäjät käyttävät Render API:a ja pidättäytyvät HTML:n kirjoittamisesta. (Drupal.org 2016ae.) API koostuu kahdesta osasta: render-putkistosta (engl. render pipeline), jolla Drupal rakentaa sivut ja render-

taulukoista (engl. render arrays), joilla tehdään sivujen rakenne-elementtejä (Drupal.org 2016ad; 2016af; 2016ag).

Render-putkiston tehtävä on koostaa Drupalin saamaan HTTP-pyyntöön HTTP-vastaus selaimelle, mihin se käyttää Http Kernel -komponenttia. Drupalin ydin tarjoaa neljä pääsisältörenderöintipalvelua (engl. main content renderer service): HTML-vastausten renderöijän ja kolme AJAX-vastausten renderöijää (Ajax, Dialog ja Modal). Kuitenkin putkisto voi periaatteessa käyttää muitakin vastausformaatteja kuten XML tai JSON. (Drupal.org 2016ag.)

Render-taulukot ovat sisäkkäisiä puumaisia assosiatiivisia avain/arvo-paritaulukoita (engl. associative array), jotka koostuvat render-elementeistä (Drupal.org 2016ad). Drupal 8:n ytimessä on yli 50 erilaista render-elementtityyppiä, joita laajennusmoduulit voivat määritellä lisää (Drupal.org 2016ah). Taulukoihin voi kytkeä erilaisia renderöintiin vaikuttavia ominaisuuksia, kuten teemakoukkuja (engl. theme hook), -funktioita ja -callbackejä. Lisäksi taulukoihin voi määritellä *Cache API:n* toteuttaman välimuistin käyttötavan. (Drupal.org 2016ad.)

2.5.5 Routing

Routing-alijärjestelmä perustuu Symfonyn Routing-komponenttiin, mutta laajentaa sen toiminnallisuuksia Drupalin tarpeisiin omaksi Routing API:ksi. Reititysjärjestelmä yhdistää eri protokollilla saapuvat HTTP-pyynnöt konfiguraatiomuuttujien joukkoihin, esimerkiksi URL-polut kontrolleri-luokkiin (Symfony.com 2016h). Drupalin alijärjestelmät ja moduulit voivat rekisteröidä reittejä, joihin ne voivat vastata staattisesti tai dynaamisesti eli reitti voi pysyä aina samana tai se voi muuttua tapauskohtaisesti. Esimerkiksi kun käyttäjä luo sisältöä Node-moduulin toimintoja käyttäen, jokaista sisältösivua varten luodaan reitti, jossa tietty URL-polku kytketään näyttämään kyseinen sisältösivu. (Drupal.org 2016ai.)

Reitit tallennetaan YAML-asetustiedostoihin `*.routing.yml`. Kukin reitti koostuu reitin nimestä, oletusasetuksesta ja vaatimuksista. Oletusasetukset voivat sisältää tiedot mm. sivun nimestä, kontrollerista, lomakkeista, näkymistä ja listoista. Vaatimuksissa voidaan eri tavoin rajata reitin ominaisuuksia ja oikeuksia,

esimerkiksi reitti voi toimia vain tietyllä HTTP-metodilla tai internet-mediatyypillä ja pääsyoikeus voidaan rajata Drupalin pääsyoikeusjärjestelmän, käyttäjän roolin, sisältötyypin, moduuliriippuvuuden tai entiteetin pääsytason perusteella. Reitille voi asettaa myös lisäehtoja, kuten välimuistin kytkeminen pois päältä. (Drupal.org 2016aj.)

2.5.6 Muita alijärjestelmiä

Access-alijärjestelmä tarjoaa luokat ja rajapinnat reitti- eli URL-polkupohjaiseen pääsynhallintaan. Näin voidaan kontrolloida käyttäjien pääsyä järjestelmän eri resursseihin. Access ei tarjoa omaa API:a, vaan on käytettävissä Entity API:n ja Node-moduulin tarjoaman API:n kautta. Toteutus vaatii myös Routing API:n dokumentaatioon tutustumista. (Drupal.org 2016j.)

Ajax API mahdollistaa Drupalin front-end- ja back-end puolten välisen keskustelun. Ajax on teknologia, joka mahdollistaa muutokset web-sivuun ilman, että web-sivua on tarpeen ladata uudestaan. Käyttöliittymään voidaan tehdä liipaisimia (engl. trigger), esimerkiksi napin painaminen tai hiiren liikuttaminen, jotka laukaisevat pyynnön palvelinpuolelle ja voivat muuttaa web-sivua palvelimelta saatavan vastauksen perusteella. (Drupal.org 2016ak; Segue Technologies 2013.)

Form API on Drupalin lomakerajapinta. Se tarjoaa kolme erilaista lomakepohjaa: geneerisen pohjan `FormBase` kaikenlaisille lomakkeille tietojen tallentamiseen, konfiguraatiopohjan `ConfigFormBase` järjestelmäkonfiguraatioiden luomiseen ja vahvistuspohjan `ConfirmFormBase` toimintojen vahvistamiseen. Lomakeluokkien luonti-, lähetys- ja validointimetodeilla hoidetaan kaikki lomakkeisiin liittyvä logiikka. (Drupal.org 2016al.) Kooditasolla lomakkeet esitetään sisäkkäisinä render-taulukkorakenteina eli Form API -taulukkoina, jotka tulostetaan käyttöliittymään Render API:a käyttäen (Drupal.org 2016am).

Muita alijärjestelmiä ovat mm. *Annotation* PHP-metadatan määrittelyyn (Drupal.org 2017f), *Cache API* datan hallittuun säilyttämiseen välimuistissa suoritus-

kyvyn parantamiseksi (Drupal.org 2017g), *Logging API* lokitietojen tallentamiseen ja generointiin (Drupal.org 2017h), *State API* järjestelmäympäristön tilan tallentamiseen yksittäisinä avain-arvo-pareina (Drupal.org 2017i) ja *Translation API* järjestelmän monikielisyyden toteuttamiseen (Drupal.org 2017j).

2.6 Drupal-moduulit

Drupal-moduulit voidaan jakaa kolmeen kategoriaan: core-, contrib- ja custom-moduuleiksi. Core-moduulit sisältyvät Drupalin ytimeen ja niitä ylläpitävät samat kehittäjät kuin ydintäkin. Contrib-moduulit (engl. contributed modules) ovat kaikkien kehittäjien julkaisemia, vapaasti saatavana olevia moduuleja, joista osa on hyvin suosittuja, osa vain harvojen käytössä. Custom-moduulit ovat mukautettuja moduuleita, jotka ovat projektikohtaisia ja syntyvät projektia tehdessä projektin tarpeisiin. (Lorétan 2011, 501.)

Tarkkaan ottaen contrib-moduuleiden voidaan katsoa jakautuvan edelleen maksuttomiin ja maksullisiin moduuleihin. Maksullisia moduuleita myy osa Drupal-palveluita tarjoavista yrityksistä (Drupal.org 2016an). Maksullisten moduulien tarjonta on kuitenkin vähäistä, mistä Drupal-kehittäjien piirissä on käyty paljon keskustelua (Tift 2016).

Drupal 8.2.x Core sisältää yhteensä 70 moduulia (Cgit.drupalcode.org 2017). Drupal.org:n contrib-moduuleita listaavassa hakemistossa moduuleita oli helmikuussa 2017 saatavana yli 36 000, joista vajaat 3 000 oli Drupal 8:lle (Drupal.org 2017k). Lukuihin eivät sisälly ns. sandbox-tilassa olevat contrib-moduulit. Custom-moduulien määrää on vaikea arvioida, sillä niistä julkaistaan vain hyvin pieni osa contrib-moduuleina.

2.6.1 Node ja Field

Node-moduuli tuo sisältöentiteettien hyödyntämisen käyttäjien saataville helpolla ja monipuolisella tavalla, joka riittää yleisimpiin sisällöntuotantotarpeisiin. Ta-

vallisesti lähes jokainen Drupal-sivustolla oleva sisältö on ”node”. Node-sisältötyypit ovat nippuja, jotka ovat variaatioita node-sisältöentiteetistä. Node-sisältötyyppejä voivat olla esimerkiksi sivu, artikkeli, blogimerkintä tai foorum-kirjoitus. (Drupal.org 2017l.)

Olemassaolevia Node-sisältötyyppejä on helppo muokata ja uusia on nopea luoda Drupalin käyttöliittymää käyttäen. Sisältötyyppien rakenne luodaan kentillä, joiden perusta on *Field-moduulissa*. Kentät ovat uudelleenkäytettäviä eli ne voivat kuulua useaan eri nippuun kuten eri Node-sisältötyyppeihin. Node-sisältötyypeillä on oltava vähintään title-kenttä, mutta muut kentät saa määritellä vapaasti. Kunkin kentän rakenne, sisältövaatimukset, näkyvyys ja widgetti on mahdollista määritellä. (Tomlinson 2015, 47-72.) Drupal 8 -ytimen mukana tulevia kenttätyyppejä (engl. field types), joita käyttöliittymän tasolla on liitettävissä node-sisältötyyppeihin, ovat mm. Checkbox, Datatime, Entity Reference, File, Image, Link, Number, Radio Button, Selection, Telephone, Term Reference ja Text (Drupal.org 2017m). Muita kenttätyyppejä on saatavana contrib-moduuleina.

2.6.2 Basic Auth, REST ja Serialization

HTTP Basic Authentication -moduuli tarjoaa Basic Access -autentikaatioskeeman RFC 2069 (IETF 1997) käyttömahdollisuuden Drupal-käyttäjien todentamiseen. Basic Auth -autentikointitapa voidaan määrittää toimintaan valituille reiteille. Autentikointitavan voi ottaa käyttöön esimerkiksi websovelluspalvelun REST-päätepisteelle. (Drupal.org 2017n.)

RESTful Web Services -moduuli tekee mahdolliseksi sisältöjen tarjoamisen sisällönhallinnan REST-rajapinnan kautta. Sisältöentiteettien luominen, lukeminen, päivittäminen ja poistaminen on mahdollista REST-päätepistettä käyttäen. *RESTful Web Services API* tarjoaa ohjelmointirajapinnan, jolla järjestelmän moduulit voivat toteuttaa Web Services -palveluita. (Drupal.org 2017o.)

Serialization-moduuli mahdollistaa sisällön muuttamisen formaatista toiseen, esimerkiksi PHP-taulukko voidaan serialisoida JSON- tai XML-formaattiin tai

JSON- tai XML-data voidaan deserialisoida PHP-taulukoksi. Moduuli perustuu Symfonyn *Serializer-komponenttiin* ja tarjoaa *Serialization API:n* moduuleiden käytettäväksi. (Drupal.org 2017p.) Esimerkiksi RESTful Web Services -moduuli käytää Serialization API:a JSON- ja XML-dataformattien käsittelyyn. (Drupal.org 2017o.)

2.6.3 Muita moduuleita

Views-moduuli generoi näkymiä sisältöentiteeteistä kuten lohko, node, user, taxonomy tai custom-sisältöentiteetti. Views-moduulin näkymät voivat olla esimerkiksi sivuja, lohkoja, syötteitä ja REST-vientejä. Näkymät voidaan esittää eri näyttötavoilla, kuten listauksina, taulukkoina ja ruudukkoina. Näkymissä näkyvät kentät voi valita ja niiden näyttötavan voi mukauttaa. Näkymiä voi suodattaa monin eri tavoin ja suodatuksen voi tarvittaessa viedä käyttöliittymään kaikkien käyttäjien säädettäväksi. Näkymän voi koostaa yhdesta tai useasta entiteetistä koostuvista datasta ja entiteettien välille voi luoda suhteita. (Drupal.org 2017q; 2017o.) Eräässä mielessä Views-näkymä on eräänlainen tietokantakysely, joka tuottaa hakutuloksen annetuilla ehdoilla, mutta esittää hakutuloksen edistyneellä ja sisällönhallintajärjestelmän kannalta käyttökelpoisella tavalla.

User-moduuli toteuttaa Drupal-ytimen käyttäjähallinnan ja roolipohjaisen käyttöoikeuksien hallinnan. Se luo user-sisältöentiteetin käyttäjätietojen – kuten käyttäjätunnus, salasana ja email – tallentamista varten. Roolien ja niiden käyttöoikeuksien asettamiseksi se määrittää role-konfiguraatioentiteetin. Oletusarvoisesti Drupalissa on käytössä kolme roolia: anonyymi, autentikoitunut ja ylläpitäjä. (Drupal.org 2017r.)

Block-moduuli tuottaa Drupal-ytimen lohko-ominaisuuden. Drupalissa lohkolla tarkoitetaan vempainta (engl. widget), joka voidaan sijoittaa sivun jollekin alueelle, missä se esimerkiksi näyttää sisältöä tai tarjoaa käyttöliittymän johonkin toiminnallisuuteen (Tomlinson 2015, 91). Drupal 8:ssa lohkot ovat kombinaatio entiteettiä ja pluginia. Lohkotyypit ovat plugineja, kun taas lohkojen ilmentymät ovat entiteettejä. Lohkojen käsittelyyn on kaksi API:a *Block Plugin*

API ja *Block Entity API*, joista ensimmäinen pohjautuu *Plugin API:iin* ja jälkimmäinen *Entity API:iin*. (Drupal.org 2016ao; 2016ap.)

Features-moduulia ei käytetä maksuvälitysjärjestelmän prototyypissä, eikä se kuulu Drupal-ytimeen, vaan se on ns. contrib-moduuli, mutta se ansaitsee tulla mainituksi, koska monissa Drupal-projekteissa se on tärkeässä roolissa projektin työnkulun kannalta. Drupal 8:ssa Features-moduulia voi käyttää edistyneeseen konfiguraationhallintaan. Moduuli mahdollistaa sivuston yksittäisten ominaisuuksien automaattisen kartoittamisen, viemisen ominaisuuspaketeiksi ja pakettien tuomisen ominaisuuksiksi johonkin toiseen Drupal-järjestelmään. Esimerkiksi kuvagalleria tai web-lomake on konfiguraatiokokonaisuus, jonka huolellinen rakentaminen voi olla työlästä, mutta joka voidaan moduulin avulla helposti käyttää uudestaan jossakin toisessa projektissa. (Drupal.org 2017s; 2017t.) Maksuvälitysjärjestelmän kannalta Features-moduulin käyttötapa voisi nopeuttaa järjestelmän kehitystä tuotantovalmiiksi palveluksi tuomalla vaikkapa taloushallintoon liittyviä ominaisuuksia jostakin toisesta Drupal-projektista.

2.7 Drupal-teemat

Teemoja käytetään toteuttamaan Drupalin front-endin ulkoasu ja rakenne (Drupal.org 2017u). Sisällönhallintajärjestelmissä ulkoasu on erotettu koodista omaksi kokonaisuudekseen, joten järjestelmän ulkoasu on helposti vaihdettavissa toiseen (Waqa Studios 2011). Kuten moduulit, myös teemat ovat modulaarisia ohjelmatiedostopaketteja. Teema määrittelee mm. värit, tyylit, fontit, kuvien ja grafiikan sijoittumisen, sivun layoutin kuten valikoiden, lohkojen, rivien ja sarakkeiden sekä muiden elementtien ja alueiden sijainnit (Tomlinson 2015, 73).

Drupal 8 Coren mukana tulevat teemat Bartik, Classy, Seven, Stable ja Stark (Drupal.org 2017v). Teemoilla on eri käyttötarkoitukset, esimerkiksi Bartik ja Seven on tarkoitettu lähinnä järjestelmän ylläpito- ja kokeilukäyttöön, Classy on tarkoitettu pohjaksi omien teemojen kehittämiseen, Stark on pelkkä minimalistinen kehys ilman varsinaista ulkoasua ja Stable kokoaa yhteen Drupal-ytimen CSS-koodit ja muut ytimen teemat ovat sen aliteemoja (engl. sub-theme) yh-

teensopivuuden varmistamiseksi (Drupal.org 2017w; Tomlinson 2015, 76). Drupal.org-sivuston teemahakemisto listaa yli 2000 valmista avoimen lähdekoodin contrib-teemaa, joista helmikuussa 2017 vasta vajaat 200 oli Drupal 8:lle (Drupal.org 2017u). Lisäksi useat yritykset valmistavat ja myyvät Drupal-teemoja. Drupal-kehittäjien suosimia contrib-teemoja, joiden varaan web-sivustojen ja -sovellusten custom-teemoja rakennetaan ovat Classyn lisäksi mm. HTML5-teemat Omega ja Zen sekä front-end-kehykset Bootstrap ja ZURB Foundation.

Drupal 8:n teemajärjestelmä käyttää oletuksena Symfonysta tunnettua PHP-kielellä kirjoitettua Twig-teemakonetta (engl. template engine), jota kehutaan moderniksi, tehokkaaksi ja olio-ohjelmointiparadigmaan hyvin istuvaksi teknologiaksi. Twig-teemat eivät sisällä PHP-kieltä, vaan niissä on käytössä oma syntaksi, joka mahdollistaa mm. muuttujien, ehto- ja kontrollirakenteiden sekä suodattimien käytön. Twig kääntää ennen teeman esittämistä `*.html.twig`-tiedostot välimuistiin, mikä parantaa teemakoneen suorituskykyä. (Drupal.org 2017x.) Drupal-teemat määritellään teeman omassa `*.info.yml`-asetustiedostossa (Drupal.org 2017y). CSS- ja JavaScript-tiedostot ovat Drupal 8:ssa assetseja, jotka lisätään Drupal-teemoihin käyttämällä `*.libraries.yml`-tiedostoihin määriteltäviä kirjastoja. Teeman tyyli-, JavaScript- ja mediatiedostot sijoitetaan tavallisesti teeman alihakemistoihin, mutta ne voidaan sijoittaa muuallekin, kuten ulkoisille palvelimille tai CDN-pilveen (engl. Content Delivery Network). (Drupal.org 2017z.)

3 Mobiilimaksun toiminta ja maksuvälitysjärjestelmän toteutusympäristö

Drupal 8:n käytöstä web-sovelluskehityksessä kuvataan projekti, jossa suunnitellaan ja toteutetaan prototyyppi mobiilimaksujen maksuvälitysjärjestelmästä. Työskentely aloitettiin ns. esitutkimuksella, jossa kartoitettiin Mobiilimaksu-maksutavan toimintaperiaatteita ja arvioitiin Drupalin soveltuvuutta maksuvälitysjärjestelmäksi. Esitutkimusosuudessa löydettyjen tietojen syventämiseen ja täsmentämiseen palattiin useaan otteeseen opinnäytetyöprosessin aikana.

3.1 Mobiilimaksu

Mobiilioperaattorit DNA, Elisa ja TeliaSonera päättivät vuonna 2014 yhdistää voimansa ja valtuuttaa Teleforum ry:n luomaan IP-laskutukselle operaattoreiden yhteisen, muista maksutavoista erottuvan Mobiilimaksu-brändin (Teleforum 2014). Mobiilimaksu-maksutapahtuman on oltava aina samanlainen palvelun-tarjoajasta tai operaattorista riippumatta (Vaittinen 2016). Mobiilimaksua käytettäessä asiakas tunnistetaan mobiililaitteen SIM-kortin perusteella. Tällöin maksutapahtuma ei välttämättä edellytä asiakastietojen antamista tai kirjautumista palveluun, koska kauppias saa tunnistetietoja – kuten asiakkaan matkapuhelinnumeron – automaattisesti suoraan mobiilioperaattorilta. (MAPEL 2017; Teleforum 2016a.) Mobiilimaksu toimii vain operaattoriverkossa eli operaattorin tarjoamalla mobiili-internetyhteydellä. Mobiilimaksu on tarkoitettu käytettäväksi ensisijaisesti älypuhelimella tai tabletilla. (Vaittinen 2016.) Tammikuussa 2017 Mobiilimaksu-maksutapaa Elisan mobiili-internetyhteydellä kokeiltuani varmistuin, että maksutavalla pystyi maksamaan niin tietokoneilla kuin mobiililaitteilla, joissa ei ollut SIM-korttia, kunhan yhteys internetiin oli muodostettu mobiilidata-yhteydellä esim. nettitikun tai mobiilireitittimen kautta.

Mobiilimaksu soveltuu sekä palveluiden että tuotteiden pienmaksamiseen. Maksutapahtuman enimmäissumma on 60 € / kertaostos. Kukin operaattori on määrittänyt mobiilimaksulle lisäksi kuukausirajan; esimerkiksi DNA:n kuukausiraja oli tätä kirjoittaessa oletusarvoisesti 150 € / kk, mutta rajaa oli mahdollista

korottaa asiakaskohtaisesti. Tehdyt ostokset operaattori veloittaa suoraan asiakkaansa puhelinlaskulla tai prepaid-saldosta. (DNA 2017a; Teleforum 2016a.)

3.2 Mobiilimaksu-maksutavan tarjoaminen

Palveluntarjoajien tulee noudattaa Mapelin eli Teleforum ry:n ylläpitämän Maksullisten puhelinpalveluiden eettisen lautakunnan laatimia eettisiä sääntöjä eli Mapel-normistoa. Uudet Mobiilimaksun toteuttamista koskevat erityissäännöt (normiston 25. §) julkistettiin alkuvuodesta 2016 ja niillä pyritään turvaamaan kuluttajien oikeudet aiempaa paremmin. (Teleforum 2016b; 2016c.) Täsmennetyt säännöt ovat tulleet selvästi tarpeeseen, sillä aikaisemmin joidenkin sisältöpalveluntarjoajien toiminta ei ollut omiaan lisäämään luottamusta uutta maksutapaa kohtaan; esimerkiksi useille mobiililiittymien käyttäjille oli ostosten veloittaminen puhelinlaskulla saattanut tulla täysin yllätyksenä ja laskulla olleet veloitustiedot olivat saattaneet olla epämääräisiä (esimerkiksi DNA 2017b). Asiaan ottivat kantaa niin Viestintävirasto (Juutinen 2016) kuin Kilpailu- ja kuluttajavirastokin (Ojajärvi 2016). Käyttäjillä ei myöskään ollut tietoa uuden maksutavan ilmestymisestä mobiililiittymään automaattisesti, eikä tietoa siitä, miten se toimi tai miten sen sai pois päältä. Lisäksi operaattorit tulkitsivat viranomaismääräyksiä estoluokituksista toisistaan poikkeavasti. (Juutinen 2016.)

Uusien Mobiilimaksu-erityissääntöjen mukaan asiakkaalle on kerrottava selvästi, mitä hyödykettä kauppa koskee, ostoksen hinta ja keskeiset ehdot. Asiakkaalle on kerrottava, että mobiililiittymä tunnistetaan automaattisesti, matkapuhelinnumero siirtyy palveluntarjoajan tietoon ja maksu veloitetaan puhelinlaskulla. Maksutapa on oltava tunnistettavissa Mobiilimaksu-logosta (kuvio 1), jonka käyttötavat määritetään graafisessa ohjeistossa (Teleforum 2016b). Maksunappi on merkittävä yksiselitteisesti ja sen on erotuttava ympäristöstä niin, että asiakkaalle on selvää napin painamisesta aiheutuva maksuvelvollisuus. Maksunapissa tai sen välittömässä läheisyydessä on oltava tieto maksettavasta summasta ja valuutasta. Maksutapahtuman on oltava vähintään kaksivaiheinen siten, että maksun vahvistaminen edellyttää vähintään kahta napin painallusta

tai muuta siihen verrattavaa aktiivista toimenpidettä. Jos kyseessä on kestotilaus, maksun vahvistamisen jälkeen on kerrottava tilauksen peruutustapa. Asiakas ei saa pystyä vahvistamaan uutta maksua, ennen kuin hän voi todeta edellisen maksutapahtuman onnistuneen tai epäonnistuneen. Maksutapahtuman onnistumisesta tai epäonnistumisesta on kerrottava erikseen sanallisesti, jos asiakas ei saa hyödykettä heti kokonaan käyttöönsä. (MAPEL 2017.) Virhetilanteissa ensisijaisesti palveluntarjoajan asiakaspalvelun tulee pystyä selvittämään tilanne asiakkaan kanssa ja hyvittää mahdolliset virhemaksut (Vaittinen 2016).

Kuvio 1. Mobiilimaksu-tunnuksen muodostavan SIM-kortin muotoinen ikoni ja tekstin sisältävä nimilogo. Pelkkää SIM-ikonia voi käyttää silloin, kun Mobiilimaksu on osa muita maksutapoja. (Teleforum 2016b.)

Estoluokkien eli palveluryhmien käytöstä mobiilimaksamisessa säädetään vuoden 2017 alusta lähtien myös Viestintäviraston määräyksellä 35 R/2016 M. Määräyksen 6. § ulottaa määräyksen palveluryhmistä yksiselitteisesti myös datayhteydellä käytettäviin palveluihin, jotka ryhmitellään määräyksen 5. §:n 1. momentissa osoitettuihin palveluryhmiin, joita ovat 1. yleishyödylliset palvelut, 2. asiointipalvelut, 3. ajanvietepalvelut ja 4. aikuisviihdepalvelut. Mikäli palvelun sisältöön kuuluu eri palveluryhmien palveluja, sijoitetaan koko palvelu korkeimpaan palveluryhmään. (Viestintävirasto 2016a.) Esimerkiksi, jos palvelun sisältöön kuuluu sekä 2. asiointi- että 3. ajanvietepalveluita, sijoitetaan palvelun koko sisältö ajanvietepalveluiden palveluryhmään 3. Mobiilimaksua ei sallita, jos mobiililiittymään on kytketty esto maksun palveluluokkaan kuuluville ostoksille (Teleforum 2016a).

Jos maksuvälitysjärjestelmää tarjotaan palveluna toisille yrityksille eli maksuvälitysjärjestelmä välittää rahaa maksajan ja maksunsaajan välillä, kysymys on maksupalvelun tarjoamisesta (Finanssivalvonta 2014). Maksupalvelun tarjoa-

misesta säädetään maksupalvelulaissa (290/2010) ja maksulaitoslaissa (297/2010). Maksupalvelu- ja maksulaitoslakien noudattamista valvoo Finanssivalvonta, jonka määräykset eli oikeussäännökset velvoittavat maksupalvelun tarjoajia. Lisäksi maksupalvelun tarjoamisessa on huomioitava mm. laki rahanpesun ja terrorismin rahoittamisen estämisestä ja selvittämisestä (503/2008). (Finanssivalvonta 2017.) Maksulaitoslain 6. §:n mukaan maksupalvelua saa tarjota vain, jos Finanssivalvonta on myöntänyt toimiluvan, mikä tarkoittaa, että oikeushenkilö on rekisteröitynyt maksulaitokseksi. Maksulaitoslain 7. §:n mukaan maksupalvelutoimintaa saa kuitenkin harjoittaa ilmoituksenvaraisesti ilman toimilupaa, jos maksutapahtumien yhteenlaskettu enimmäismäärä on oikeushenkilöllä 3 miljoonaa euroa kuukaudessa ja luonnollisella henkilöllä 50 000 euroa kuukaudessa, mikä lasketaan viimeisten 12 kuukauden keskiarvosta. (Finlex 2017.) Maksulaitoslain mukaisen ilmoituksen liitteeksi tarvitaan mm. liiketoimintasuunnitelma, johon tulee sisällyttää esimerkiksi ”kuvaus palveluun tunnistautumisesta ja maksutapahtumien hyväksymisestä”, ”kuvaus maksujen välittämiseen käytettävistä tietojärjestelmistä ja palvelimien tietokantojen sijainnista”, ”selvitys järjestelmien ja tietokantojen tietoturvasta” ja ”kuvaus toiminnan jatkuvuudesta (it-järjestelmät ja liiketoiminnan jatkuvuus)”. Maksupalvelutoiminnan saa aloittaa vasta, kun Finanssivalvonta on tutkinut ilmoituksen ja antanut päätöksen siitä, että toiminnalle ilman toimilupaa on edellytykset. Myös ilman toimilupaa maksupalvelutoimintaa harjoittavat rekisteröidään Finanssivalvonnan maksulaitosrekisteriin. Maksupalvelutoiminnan harjoittamiseen liittyy myös joitakin ilmoitusvelvollisuuksia, esimerkiksi jos maksupalvelun toiminnassa ilmenee häiriöitä tai vikoja, palveluntarjoajan on ilmoitettava niistä oma-aloitteisesti Finanssivalvonnalle. (Finanssivalvonta 2017.)

3.3 Maksuvälitysjärjestelmän toteutusympäristö

Mobiililiittymän käyttäjille näkyvän yhteisen Mobiilimaksu-brändin takaa integraation toteuttamista harkitsevat palveluntarjoajat löytävät kuitenkin jokaisen operaattorin erilaiset mobiilimaksu-API-toteutukset, hinnoittelumallit ja sopimusehdot. Se tekee Mobiilimaksu-maksutavan käyttöönoton haastavaksi ja kalliiksi etenkin pienemmille toimijoille. Tilanne luo mahdollisuuden palveluintegraatiolii-

ketoiminnalle, jossa Mobiilimaksu-maksutavalle luodaan kohtuuhintainen ja helposti käyttöönotettava, kaikkien operaattoreiden palvelut tarjoava maksuvälitysjärjestelmä eli maksusiltapalvelu, joka toimii yhdistävänä väylänä operaattoreiden erilaisiin mobiilimaksu-API:hin.

Palveluntarjoajiksi – joita myös Mapel-normisto velvoittaa – katsotaan mobiilioperaattoreiden lisäksi sekä maksuintegraattori eli ”tukkukauppias” että maksuvälitysjärjestelmän omaan palveluunsa integroinut palveluntarjoaja eli ”kauppias”, joten vastuut ja niiden jakaminen eri toimijoiden kesken on huomioitava hyvin mm. maksuvälitysjärjestelmän sopimusehtoja laadittaessa. Koska jokaista mobiilioperaattorin liittymäasiakkaalle Mobiilimaksu-maksutavan tarjoamiseen osallistuvaa tahoa voidaan kutsua palveluntarjoajaksi, käytetään tässä opinnäytetyöraportissa seuraavia käsitteitä:

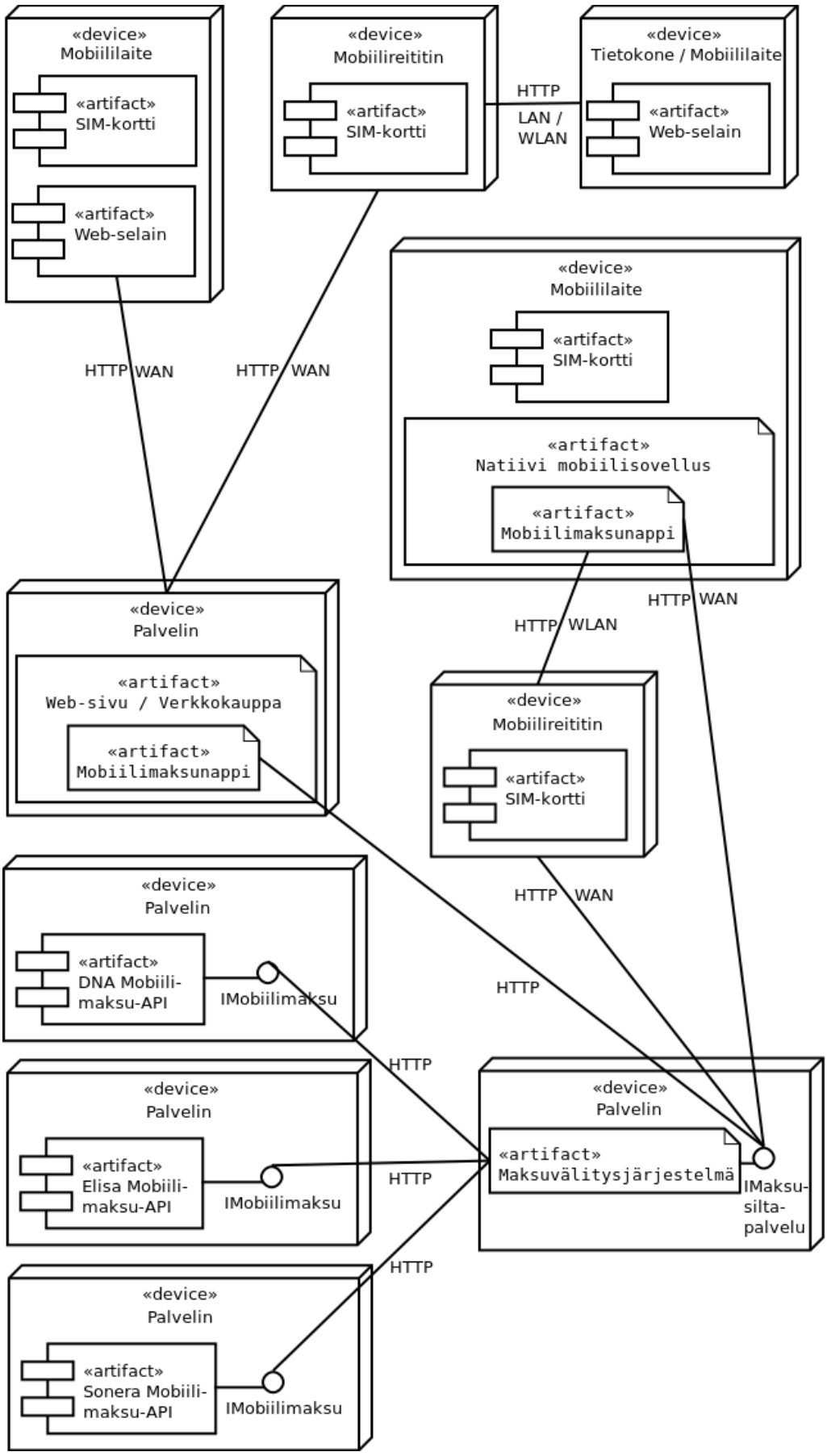
- mobiilioperaattori-palveluntarjoajaa kutsutaan *operaattoriksi* tai *mobiilioperaattoriksi* ja heidän muille palveluntarjoajille tarjoamaansa palvelua *mobiilimaksu-API:ksi* ja loppukäyttäjille tarjoamaansa palvelua *Mobiilimaksuksi*
- maksuvälitysjärjestelmä-palveluntarjoajaa eli palveluintegraattoria kutsutaan *maksuintegraattoriksi* ja heidän tarjoamaansa palvelua *maksuvälitysjärjestelmäksi*, *maksuvälitysjärjestelmän API:ksi, maksusillaksi* tai *maksusiltapalveluksi*
- loppukäyttäjille jotakin hyödykettä myyvää ja Mobiilimaksua asiakkailleen maksutapana tarjoavaa palveluntarjoajaa kutsutaan *kauppiaaksi* ja heidän tarjoamaansa palvelua, jossa hyödykettä myydään *kauppiaan sovellukseksi*
- mobiilioperaattorin liittymäasiakasta kutsutaan *loppukäyttäjäksi* tai *mobiililiittymän käyttäjäksi*.

Jotta maksuvälitysjärjestelmän sijoittuminen toimintaympäristöön tulisi helpommaksi hahmottaa, laadin kuvion 2 UML-sijoittelukaavion (engl. deployment diagram) muotoon. Maksuvälitysjärjestelmään tulisi siis integroida kolme erilaista Imobiilimaksu-rajapintaa ja tarjota edelleen yksi Imaksusiltapalvelu-rajapinta kauppiaiden sovelluksille, kuten erilaisille verkkokaupoille ja mobiilisovelluksille.

Mobiilimaksunappi voidaan sijoittaa niin web-sivuille kuin mobiilisovelluksiinkin. Älypuhelimien ja tablettien suoran mobiili-internetyhteyden lisäksi loppukäyttäjä voi tehdä Mobiilimaksun myös mobiilireitittimen, USB-nettitikun tai vastaavan laitteen kautta muodostuneen mobiili-internetyhteyden kautta, jolloin maksu toimii myös mm. tietokoneista käsin käytettynä. Esimerkiksi, jos mobiililaitteen liittymään on tehty mobiilimaksun käyttöesto tai laitteessa ei ole liittymää, niin mobiililaitteella voidaan kuitenkin tehdä Mobiilimaksu, jos sillä kytkeydytään langatonta lähiverkkoyhteyttä (WLAN) käyttäen mobiilireitittimeen, jonka liittymään käyttöestoa ei ole tehty; tällöin ostos veloitetaan mobiilireititintä käyttävän liittymän laskussa.

Kuvion 2:n kuvaus:

- Edellytys operaattorin Mobiilimaksun käyttöön on asiakkaan kytkeytyminen internetverkkoon operaattorin mobiili-internetyhteyttä käyttäen, sillä operaattorit tunnistavat asiakkaan SIM-kortin perusteella.
- Jos mobiili-internetyhteys on jaettu edelleen lähiverkossa esim. mobiilireitittimellä, voidaan Mobiilimaksu tehdä kaikista jaettua yhteyttä käyttävistä laitteista.
- Jokainen operaattori tarjoaa oman yksilöllisen rajapintansa IMobiilimaksu, joka integroidaan maksuvälitysjärjestelmään.
- Maksuvälitysjärjestelmä tarjoaa oman Imaksusiltapalvelu REST-rajapinnan, jonka kautta palveluun voivat integroitua niin natiivit sovellukset kuin web-selaimella käytettävät sovellukset, verkkokaupat ja web-sivut.
- Maksuvälitysjärjestelmään integroitavat palvelut pystyvät tarjoamaan Mobiilimaksun maksutapana kaikkien operaattoreiden asiakkaille.
- Yhteydet sekä eri operaattoreiden että maksuvälitysjärjestelmän rajapintoihin toteutuvat suojatulla HTTPS-protokollalla, lukuunottamatta maksutapahtumaa edeltävää loppukäyttäjän tunnistamista, johon käytetään suojaamatonta HTTP-protokollaa.



Kuvio 2. Maksuvälitysjärjestelmän toteutusympäristön kuvaus UML-sijoittelukaaviona.

3.4 Mobiilioperaattoreiden mobiilimaksu-API:t

Ennen maksuvälitysjärjestelmäprojektin suunnittelua oli välttämätöntä tutustua perusteellisesti mobiilioperaattoreiden mobiilimaksu-API-dokumentaatioihin. Vertailun tavoitteena oli pystyä hahmottamaan, mitä eri API:t vaativat järjestelmältä ja valitsemaan yksi helpoiten integroitavissa oleva mobiilimaksu-API, josta integraatioiden tekeminen voitaisiin aloittaa. Vertailun olennaisimmat tulokset on esitetty taulukossa 1.

Taulukko 1. Suomalaisten mobiilioperaattoreiden mobiilimaksu-API:t syksyllä 2016 (tiedot koottu seuraavista lähteistä: Elisa yritysasiakaspalvelu 2015; Järvinen 2016a; Järvinen 2016b; Koskelainen 2016; Sonera 2016a; Sonera 2016b; Tieto Connection / VAS Center 2014).

Mobiilimaksu-API:n nimi ja versionumero	**Elisa Mobile Interface Billing Platform 2.0**	**Sonera OMA Payment RESTful interface 5.1**	**Tieto Connection IP Connectivity Charging API 1.2.2**
API:a käyttävät mobiilioperaattorit	Elisa, Saunalahti	Sonera, Tele Finland	DNA, Moi
API:n toteutustapa	SOAP over HTTP	REST	HTTP GET/POST
Loppukäyttäjän tunnistaminen	MSISDN Resolution	MSISDN Enrichment	IP Billing Request
Kehittäjäympäristön saatavuus	Ei	Kyllä, Sonera Opaali -portaali	Ei
Maksuttoman testiympäristön saatavuus	Ei	Kyllä, poislukien MSISDN Enrichment	Kyllä
Dokumentaation julkisuus	Ei	Kyllä, poislukien MSISDN Enrichment	Ei

Vertailun perusteella integraatioiden tekeminen kannatti aloittaa Soneran API:sta. API oli muiden operaattoreiden ratkaisuihin verrattuna tuorein, ja lisäksi opinnäytteen tekijällä oli aiempaa kokemusta juuri RESTful-tyypin API:eista. API:n dokumentaatio oli julkinen, joten myös opinnäytetyön lukijoilla on mahdollisuus tutustua siihen mutkattomasti. Dokumentaation esitystapa oli hyvin selkeä, joten sen toiminnan hahmottaminen oli suhteellisen vaivatonta. Valinnan vahvistivat vielä Soneran tarjoama maksuton Opaali-kehittäjäportaali testiympä-

ristöineen ja yhteistyöhalukkuus suoraan opiskelijan kanssa. Loppukäyttäjän tunnistamisen toteuttavan Soneran MSISDN Enrichment -palvelun käyttö ei ollut mahdollista maksutta, mutta sen toteutustapa vaikutti niin yksinkertaiselta, että sen toimintaa simuloivan oman testiskriptin laatimisen opinnäytettä varten arveltiin olevan mahdollista.

3.5 Drupal 8:n soveltuvuus maksuvälitysjärjestelmän toteutusalustaksi

Maksuvälitysjärjestelmän tapauksessa Drupalia ei käytetä niinkään sisällönhallintajärjestelmänä vaan ns. väliohjelmistona (engl. middleware) useiden eri back-end-järjestelmien väliseen hallittuun viestintään. API:n käyttöoikeuksien hallintaan ja liikenteen seurantaan käytettävää ylläpitoliittymää lukuunottamatta front-end-puolta ei varsinaisesti ole ollenkaan. Maksuvälitysjärjestelmä ei suoraan toimi back-end-puolena asiakasohjelmistojen front-end-puolille vaan ne tarvitsevat oman back-end-puolensa API:n käyttöön. Kun Drupalista hyödynnetään vain back-end-puolta, siitä voidaan käyttää myös nimitystä ”päätön Drupal” (engl. Headless Drupal) (Burge 2014).

Opinnäytetyössä käytettävä alusta (Drupal 8) oli sovittu jo hyvissä ajoin ennen tietoa tehtävästä prototyypistä, joten raportissa ei ollut relevanttia tarkemmin tutkia ja vertailla eri alustojen soveltuvuutta maksuvälitysjärjestelmäksi. Näin ollen ratkaiseva peruste juuri Drupalin käyttöön toteutusalustana oli pyrkimys jatkaa opinnäytetyötä varten aloitettua Drupal 8 back-end-kehitystyön laajaa ja vaativaa oppimisprosessia sekä myös koettaa hiukan venyttää mielikuvituksen rajoja, mitä kaikkea alustalla on mahdollista menestyksellisesti rakentaa.

On kuitenkin hyvä hiukan arvioida Drupal 8:n soveltuvuutta käyttötarkoitukseen. Web-kehittäjä Tatu Tamminen (2016) on blogissaan esittänyt tarkistuslistan sen arviointiin, kuinka tietty työkalu tai työkalun versio karkeasti ottaen voisi soveltua jonkin ongelman ratkaisemiseen. Käytin tarkistuslistaa taulukossa 2, jossa arvioin Drupal 8:n soveltuvuutta tapaukseen yleisesti ottaen.

Taulukko 2. Arvio Drupal 8:n sopivuudesta maksuvälitysjärjestelmäksi tarkistuslistaa (Tamminen 2016) soveltaen.

Kysymys	**Arvio**
Onko työkalu sopiva ongelmaan?	Drupal on sisällönhallintajärjestelmä, mutta maksuvälitysjärjestelmässä ei varsinaisesti ole kysymys sisällönhallinnasta. Suurinta osaa Drupalin ominaisuuksista ei tarvita, jos Drupal on pelkkä palvelinten välisten viestien välittäjä ja hallinnoija. Tietokantaan tallentuvat tiedot maksutapahtumista, kauppiaiden sovellusten asetustiedot ja laskutustiedot olisivat tätä ”sisältöä”, mutta käyttötapaus on erilainen, kuin mihin sisällönhallintajärjestelmää yleensä käytetään. Drupal 8 on kuitenkin myös sisällönhallintakehys, joka perustuu Symfony HTTP-kehykseen ja käyttää myös Guzzle HTTP-kirjastoa, joten tässä mielessä Drupal alustana venyy hyvin tämänkin ongelman ratkaisuun. Drupalin varsinaisia sisällönhallintaominaisuuksia voitaisiin tarvita lähinnä, jos kauppiaille pyritään tarjoamaan hyvä ylläpitoliittymä palveluun, joka tarjoaa raportointiominaisuuksia, laskutuksen, työkalut virhemaksujen tutkintaan ja hyvittämiseen, ym. Drupal on myös hyvin tietoturvallinen ratkaisu, mikä maksusillassa on tärkeintä. Drupalin suorituskyky väliohjelmistona hiukan mietityttää, mutta ei kuulu opinnäytteessä käsiteltäviin kysymyksiin.
Kuinka olennainen työkalu on ongelman ratkaisun kannalta?	Drupal-alustaan voidaan tukeutua toteutuksessa täysin, mutta omien laajennusosien kehittäminen Drupaliin on välttämätöntä.
Mitä vaihtoehtoja on olemassa?	Drupalin sijaan voisi olla mahdollista käyttää suoraan Symfonya tai jotain kevyempää Symfonyyn perustuvaa alustaa kuten Silex-mikrokehystä. Myös kokonaan toinen teknologia kuten NodeJS tai Zend Framework 2:een perustuva Apigility voisivat olla vaihtoehtoja.
Kuinka helppoa on vaihtaa toiseen vaihtoehtoon?	Drupalista toiseen vaihtoehtoon vaihtaminen ei olisi kovin helppoa, sillä ohjelmakoodi tulisi kirjoittaa uudestaan lähes täysin. Toisaalta Drupaliin tarvittavan custom-ohjelmakoodin määrä ei ole massiivinen, joten vaihtaminen toiseen ratkaisuun ei olisi mahdotontakaan.
Minkälaista tukea on saatavana?	Tukea saa maksutta lähinnä Drupal-yhteisöltä ja toisilta Drupal-kehittäjiltä. Kaupallista tukea voi olla saatavana, mikäli toteutuksessa käytetään esim. Acquian Drupal-palveluita tai ostetaan tukea joltakin Drupal-asiantuntijayritykseltä.
Liittyykö työkaluun paljon avoimia ongelmia?	Drupal on käyttövalmis ohjelmisto, joka kehittyy jatkuvasti ja jonka tietoturvasta huolehditaan aktiivisesti (Lullabot 2016). Drupal 8:n käyttöön ei liittyne ongelmia, kunhan maksuvälitysjärjestelmän kehitys ja ylläpito on hoidettu mahdollisimman ammattitaitoisesti ja tietoturva pidetään aina ajan tasalla.
Kuinka suuri on käyttäjäkunta ja kuinka aktiivinen on yhteisö?	Drupal-yhteisöön kuuluu yli miljoona henkilöä, joten se on yksi suurimpia avoimen lähdekoodin käyttäjäkuntia (Drupal.org 2016a). Näin ollen aktiivisiakin käyttäjiä on merkittäviä määriä.
Kuinka helppoa työkalu on oppia (dokumentaatio, tutoriaalit, kirjat, videot, ym)?	Keväällä 2016 back-end-kehitykseen soveltuvaa Drupal 8:aa käsittelevää kirjallisuutta ei ollut saatavana lainkaan ja myöskin projektin dokumentaatio oli hyvin keskeneräistä. Lähinnä kaksi valmennuskurssia oli saatavana. Syksyllä 2016 back-end-kehityksen kannalta tarpeellisen dokumentaation määrä oli lisääntynyt merkittävästi, mutta kirjallisuuden saatavuus oli edelleenkin hyvin niukkaa.
Mitä tiedetään teknologian tulevaisuudesta?	Maksuvälitysjärjestelmän kannalta merkittävintä on Drupal API-first initiatiiven edistyminen. Drupalin ominaisuudet erilaisiin Web Services API-toteutuksiin olivat parantumassa vahvasti jo lähitulevaisuudessa 2017-

	2018. (Buytaert 2016b.)
Mitkä ovat lisensointivaatimukset?	Drupal on avointa lähdekoodia ja lisensoitu GNU General Public -lisenssillä, mikä mahdollistaa ohjelmiston vapaan käyttö- ja muokkausoikeuden. Omat lisäosat on lisensoitava samalla lisenssillä. Lisenssi ei kuitenkaan edellytä, että ohjelmistoa ei saisi myydä tai että koodi pitäisi julkaista vapaasti saataville tai luovuttaa jollekin taholle. Maksuvälitysjärjestelmän tapauksessa palvelua tarjottaisiin asiakkaille todennäköisesti sovellusvuokrauksena eli SaaS-palveluna (engl. Software as a Service), joten GNU-lisensoidun ohjelmiston käyttö alustana soveltuu hyvin.
Onko helppoa löytää ihmisiä, jotka ovat kiinnostuneet oppimaan työkalun?	Drupal-kehittäjistä on jatkuvasti kovasti kysyntää ja kysynnän arvellaan jatkuvasti kasvavan. 92% Drupal Associationin kyselyyn 2015 vastanneista web-kehittäjistä oli opiskelemassa tai aikeissa opiskella Drupal-kehitystä. (Drupal Association 2015.) Suomen tilanne lienee jokseenkin saman suuntainen, esim. Vierityspalkki-blogin työpaikat-listalla etsitään jatkuvasti Drupal-kehittäjiä (Vierityspalkki 2017). Drupal-kehityksen oppiminen vie suhteellisen kauan aikaa ja on hyvin vaativaa (Shindelar 2016), joten osaavien kehittäjien saatavuus ei mahdollisesti aina ole helppoa.

4 Projektin suunnittelu ja menetelmät

Esitutkimusosuuden jälkeen siirryttiin projektin suunnitteluosuuteen. Ensiksi prototyypin kokoa ja projektin luonnetta rajattiin paremmin opinnäytetyöhön sopivaksi. Rajausta tarkennettiin useaan otteeseen opinnäytetyöprosessin aikana. Drupal-projektien suunnittelua ja ketteriä menetelmiä tarkasteltiin eri näkökulmista. Projektille tehtiin ns. korkean tason suunnitelma. Projektin käyttäjätarinat kirjoitettiin ja syötettiin projektinhallintajärjestelmään.

Projekti suunniteltiin esi-alfa-version (engl. pre-alpha) tasolla olevan prototyypin laajuudessa. Esi-alfa-versiolla tarkoitetaan ohjelmiston ns. alfa-versiota eli alfatestausvaihetta (engl. alpha testing) edeltävää kehitysversiota, jossa ohjelmiston kaikkia julkaisua varten tarvittavia ominaisuuksia ei ole vielä kehitetty, joten se ei ole vielä valmis arvioitavaksi ja testattavaksi kokonaisuutena. Alfatestausvaiheessa järjestelmä on jo kokonaisuutena testattavissa testiympäristössä ja järjestelmän kaikki julkaisua varten tarvittavat ominaisuudet on toteutettu, mutta vielä viimeistelemättä. Alfaa seuraavassa betatestausvaiheessa (engl. beta testing) järjestelmä on jo koekäytössä tuotantoympäristössä ns. beta-versiona, mutta tuotetta ei ole vielä otettu julkisesti käyttöön, vaan sen toimintaa tarkkaillaan ja siitä kerätään palautetta ohjelmiston viimeistelemiseksi julkaisuvalmiiksi tuotteeksi. (Kasurinen 2013, 73; Wiktionary 2016.)

4.1 Drupal ja projektinhallintamenetelmät

Drupal-kehityksessä voidaan käyttää niin perinteistä vesiputousmallia kuin ketteriäkin menetelmiä (Scavarda 2011, 207–209). Vesiputousmallissa – jota voidaan kutsua myös suunnitteluvetoiseksi lähestymistavaksi (Cobb 2015, 4) tai versioksi klassisesta elinkaarimallista (Rouse 2007) – projekti etenee lineaarisesti vaiheesta toiseen, joita ovat 1. viestintä (projektin käynnistäminen, vaatimusten kerääminen), 2. suunnitelma (arviointi, aikataulutus, seuranta), 3. mallinnus (analyysi, suunnittelu), 4. rakentaminen (suunnittelu, testaus) ja 5. käyttöönotto (toimitus, tuki, palaute) (Pressman 2010, 39). Lähestymistapaa nimite-

tään vesiputousmalliksi, sillä projekti etenee vaiheittain kuin kalliolta laskeva vesiputous, eikä palaa enää takaisin aiempiin vaiheisiin (Rouse 2007).

Ketteriä menetelmiä käyttäen projekti etenee iteratiivisesti ja ohjelmisto kasvaa inkrementaalisesti lisäten itseensä edellisten iteraatioiden tulokset. Esimerkiksi XP:n prosessimallissa toistetaan jokaisessa iteraatiossa joustavasti vaiheita: 1. suunnitelma (käyttäjätarinat, hyväksymistestauksen kriteerit), 2. suunnittelu (yksinkertaisin menetelmin ja prototyypein), 3. ohjelmointi, 4. testaus, 5. julkaisu (Pressman 2010, 73–74, 80–81). Lähestymistavan ajattelumalli on määritelty ytimekkäästi Ketterän ohjelmistokehityksen julistuksessa, mikä auttaa myös ymmärtämään, kuinka se eroaa klassisesta elinkaarimallista. Julistuksen suomennoksen mukaan ”löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä”. ”Kokemuksemme perusteella arvostamme: yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja; toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota; asiakasyhteistyötä enemmän kuin sopimusneuvotteluja; vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa”. ”Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän”. (Beck, Beedle, Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mellor, Schwaber, Sutherland & Thomas 2001.)

Yksinkertaistaen menetelmien olennainen ero on siinä, että vesiputousmallia käytettäessä toteutus suunnitellaan perusteellisesti projektin alkuvaiheessa, projekti etenee vaiheesta toiseen lineaarisesti ja viestintä asiakkaan kanssa painottuu projektin alkuun ja loppuun, kun taas ketteriä menetelmiä käytettäessä projekti suunnitellaan ja toteutetaan syklisesti pienin kehitysaskelin, jolloin asiakkaalla on mahdollisuus vaatia muutoksia ja vaikuttaa lopputulokseen ennen uuden iteraation aloittamista, mikä on *ketterää* (Cobb 2015, 5–6). Perinteisiä ja ketteriä menetelmiä voidaan myös yhdistää eri tavoin, mitä kutsutaan adaptiiviseksi lähestymistavaksi (engl. adaptive approach) (Cobb 2015, XV). Drupal-projekteissa voidaan hyötyä molempien lähestymistapojen yhdistämisestä. Vesiputousmallin painottaminen sopii lähinnä asiakasprojekteihin, joissa on tarkka aikataulu tai joustamaton budjetti ja joissa voidaan tietää hyvin ennalta, mitä projektissa syntyy ja miten projekti tehdään. Ketterien menetelmien painottaminen sopii sekä yrityksen sisäisiin projekteihin että asiakasprojekteihin,

joissa syntyvä tuote on vasta hahmottumassa ja kehitystyöhön liittyy enemmän haasteita ja epävarmuustekijöitä, mikä vaatii myös joustavamman aikataulun ja budjetin. (Scavarda 2011, 207–209.)

Maksuvälitysjärjestelmän projektinhallinta päätettiin toteuttaa ketterillä menetelmillä. Valinta oli hyvin perusteltu, sillä kehitystyöhön liittyi merkittäviä epävarmuustekijöitä. Kysymyksessä ei ollut triviaali web-sivusto, jonka toteutuksen kulku ja sisältö oli mahdollista ennustaa aiemman kokemuksen perusteella, vaan web-sovellus, jolla oli hiukan kokeellinen luonne – ei voitu tietää tarkasti, mitä kaikkea sen kehitystyössä tulisi vastaan. Lisäksi alusta oli kehittäjälle eli opinnäytetyöntekijälle uusi, eikä back-end-kehityksen suhteen tekijäkään ollut kokenut. Lisäksi projekti oli toimeksiantajan sisäinen kehitysprojekti. Syntyvän tuotteen idea ja ominaisuudet olivat vasta hahmottumassa.

4.2 Ketterien projektien dokumentointi

Ketterän ohjelmistokehityksen julistuksessa (Beck ym. 2001) mainittiin, että ”toimivaa ohjelmistoa arvostetaan enemmän kuin kattavaa dokumentaatiota”. Cobbin (2015, 57) mukaan tavallinen väärinkäsitys tästä on, että ketterä projekti ei vaadi suunnittelua lainkaan. Projekti ketterillä menetelmillä toteutettuna vaatii yhtä paljon suunnittelua kuin perinteisiäkin menetelmiä käytettäessä, mutta se vain tehdään hyvin eri tavalla.

Perinteisiä projektinhallintamenetelmiä voi luonnehtia suunnittelukeskeisiksi ja niissä tehdään projektin aluksi mahdollisimman kattava dokumentaatio. Ennen toteutusta tehty suunnittelu on kuitenkin spekulatiivista ja osoittautuu usein vääräksi, jolloin tarvitaan aikaa vievää uudelleensuunnittelua. Ketterissä menetelmissä hyödynnetään sen sijaan ns. vyöryvän aallon suunnittelutapaa (engl. rolling-wave planning), jossa projektista tehdään aluksi vain korkean tason suunnitelma, jota tarkennetaan projektin edetessä. Suunnittelupäätökset pyritään tekemään niin myöhään kuin mahdollista, ilman että siitä on projektille haittaa. (Cobb 2015, 57–59.)

Ketteriä menetelmiä käytettäessä spekulatiivisen informaation määrä pyritään pitämään mahdollisimman pienenä. Dokumentaatiota tehdään enemmän vasta sitten, kun projektista syntyvä informaatio on vakiintunutta. Näin dokumentaatio on hyödyllistä ja vastaa sitä, mitä projektissa on syntynyt. Haasteena on saada dokumentaatio valmiiksi projektin loppuun mennessä ja dokumentoida riittävästi myös itse projektin aikana, jotta kaikki kriittinen tieto saadaan mukaan. (Ambler 2016b.) Jos projektin joka iteraation päätteeksi on tarkoitus syntyä julkaisukelpoinen tuote, tehdään dokumentaatiota tällöin jatkuvasti, jotta se saadaan valmiiksi aina iteraation loppuun mennessä (Ambler 2016c).

Toiminnallisessakin opinnäytetyössä dokumentaatio on keskeisellä sijalla, sillä opinnäytetyöstä kirjoitetaan raportti. Koska lähestymistavaksi raportissa oli otettu projektin toteutuksen kuvailu, päätettiin muistiinpanoja tehdä koko ajan työskentelyn ohessa ja kirjoittaa lopuksi kuvaus kustakin iteraatiosta, kun sen toteutus oli valmistunut. Ketterien menetelmien käytön kuvaaminen oli haastavaa opinnäytetyössä sikäli, että raportin oli edettävä lineaarisesti alusta loppuun ”kuin vesiputous”, eikä siinä voitu palata aiempiin vaiheisiin, joten kehitysprosessista oli vaikeaa antaa tarkkaa kuvaa. Käytännössä dokumentaatio kokisi muutoksia koko projektin ajan, joten iteraatiota 1 lukuun ottamatta, toisteisuuden välttämiseksi pyrittiin kuviot ja taulukot esittämään vain yhden kerran ja evoluutiotasolla, jossa ne olivat Esi-Alfa-1-prototyypin toteutuksen päätyttyä.

Dokumentaationa käytettiin myös ohjelmistoa mallintavia UML-kaavioita, jotka tehtiin UML-mallinnuskielen (engl. Unified Modeling Language) versiota 2.0 noudattaen. Mallinnusta tehtiin AMDD:tä (engl. Agile Model Driven Development) soveltaen vähän kerrassaan ja vain sen verran, että se oli juuri ja juuri riittävää projektin tarpeisiin. Kuhunkin vastaan tulleeseen suunnittelutarpeeseen pyrittiin valitsemaan tarkoituksenmukainen artefakti eli UML-kaaviotyyppi. (Ambler 2017.)

4.3 Korkean tason suunnitelma

Korkean tason suunnitelma sisältää projektin vision, soveltamisalan (engl. scope) ja tavoitteet. Suunnitelma tulisi aloittaa kirjaamalla visio siitä, mitä liiketoi-

minta-arvoa sovellus tuottaa. Seuraavaksi visio tulisi purkaa ohjelmiston toiminnallisuuksiksi. (Cobb 2015, 57–61.) Cobb (2015, 287–291) kiteyttää esimerkin korkean tason suunnitelmasta projektisuunnitelmapohjan muotoon, jonka runko on esitetty taulukossa 3. Suunnitelmaa tulisi mukauttaa ja lyhentää kunkin projektin tarpeisiin. Suunnitelman esittämiseen ja toteuttamiseen kannattaa käyttää työvälineohjelmia, kuten esim. kirjoittamiseen wiki-tyyppistä tietopankkiohjelmaa ja projektin hallintaan ja seurantaan agile-tehtävienhallintasovellusta (Cobb 2015, 142–143).

Taulukko 3. Esimerkki korkean tason suunnitelman rungosta (Cobb 2015, 287–291).

1. Projektin yleiskuva
1.1. Tausta
1.2. Ongelma
1.3. Visio
1.4. Onnistumiskriteerit
1.5. Lähestymistapa
2. Projektisuunnitelma
2.1. Soveltamisala
2.2. Käyttäjätarinat
2.3. Projektin ulkopuolella
3. Projektiin liittyvät toiset projektit ja järjestelmät
4. Projektin osanottajat
5. Reunaehdot, olettamukset ja riskit
5.1. Reunaehdot
5.2. Olettamukset
5.3. Riippuvuudet
5.4. Riskit
5.5. Aikatauluarvio

Projektille laadittiin korkean tason suunnitelma Cobbin rungon pohjalta. Projektin visiona (taulukko 5) oli kehittää yrityksen omaan ja asiakkaiden käyttöön verkkokauppoihin, sähköisiin asiointipalveluihin ja mobiilisovelluksiin sopiva maksuvälitysjärjestelmä, joka tarjoaa helpon, nopean ja edullisen Mobiilimaksumaksutavan ja jolla tehdyt ostokset maksetaan loppuasiakkaan matkapuhelin-

laskulla. Visio vastaa pitkälti epiikkaa US1, jota on edelleen elaboroitu pienemmiksi käyttäjätarinoiksi. Suunnitelmassa määriteltiin projektinhallintamenetelmä ja käyttäjätarinat, listattiin käytettävät työvälineet (taulukko 4) ja päätettiin konfiguraation- ja versionhallinnan menetelmistä sekä BitBucket-versionhallintajärjestelmän käyttöönotosta. Suunnitelmaan kirjattiin myös projektin tärkein reunaehto eli se oli toteutettava Drupal 8:lla. Suunnitelmaa päivitettiin projektin edetessä useaan otteeseen.

Taulukko 4. Projektin työvälineohjelmat.

Nimi	**Käyttötarkoitus**
Chromium	Avoimen lähdekoodin web-selain (Chromium Projects 2017).
Dia	Editori kaavioiden piirtämiseen (Dia 2017).
Drupal Console	Komentorivityökalu Drupal 8 -kehitys- ja ylläpitotyöhön mm. valmisohjelmakoodin generointiin (Drupal Console 2017).
Drush	Komentorivityökalu Drupal-kehitys- ja ylläpitotyöhön (Drush 2017).
Git	Komentorivityökalu Git-versionhallinnan käyttöön (Software Freedom Conservancy 2017).
Google Docs	Online-tekstinkäsittelytyökalu (Google 2017).
KADOS	Online-projektinhallintaohjelma ketteriä menetelmiä varten (KADOS 2017).
Lubuntu Linux	Kehitysympäristöksi valittiin Lubuntu Linux -käyttöjärjestelmä, jota ajettiin VirtualBox-virtuaalikoneessa Windows-isäntäjärjestelmässä.
Papyrus	Mallinnusympäristö kaavioiden piirtämiseen (Eclipse Foundation 2017).
PHPStorm	Monipuolinen IDE (engl. Integrated Development Environment), jonka koodieditoriin oli saatavana Drupal-tuki (Vink 2016).
Postman	Työkalu web-API:en kehittämiseen ja testaamiseen (Postdot Technologies 2017).

Taulukko 5. Projektin visio mukaillen korkean tason suunnitelman rungossa esitettyä mallia (Cobb 2015, 287–388).

Asiakas:	- Verkkokaupan omistaja - Mobiilisovelluksen omistaja - Omat sähköiset- ja mobiilipalvelut
joka tarvitsee:	- Helpon, nopean ja edullisen maksutavan mobiililaitteita käyttävälle asiakkailleen
Tuote:	- Mobiilimaksu
joka kuuluu tuoteryhmään:	- Maksupalvelut
ja tarjoaa asiakkaalle:	- Mahdollisuuden maksaa ostos matkapuhelinlaskulla helposti ja nopeasti

Projektinhallintamenetelmäksi ei valittu mitään tiettyä ketterää menetelmää. AM-, Scrum- ja XP-malleista otettiin löyhästi soveltaen käyttöön elementtejä kuten vyöryvän aallon suunnittelutapa, korkean tason suunnitelma, käyttäjätarinat, iteraatiot, backlog ja ohjelmiston mallintaminen. Prototyyppiä ei tehty tiimityönä, vaan opinnäytetyöntekijä työskenteli yksin ja etätyönä. Esimerkiksi Scrum-tiimiä ei perustettu, Scrum-rooleja ei käytetty eikä Scrum-päivätapaamisia pidetty. Iteraatioiden katselmoinnit toimitettiin screencast-videoina sekä toimeksiantajalle että lopputyön ohjaajalle.

4.4 Käyttäjätarinat

Ohjelmistoprojekteissa tehtävälle ohjelmistolle määriteltyjä vaatimuksia on tavallisesti kolmea eri tyyppiä: toiminnallisilla vaatimuksilla (engl. functional requirement) määritetään, mitä ohjelmistolla on pystyttävä tekemään; ei-toiminnallisilla vaatimuksilla (engl. non-functional requirement) määritetään, mitä laatuominaisuuksia ohjelmistolla on oltava; ja reunaehdoilla (engl. constraints) määritellään, millä ehdoin ohjelmiston toteutus sallitaan (Haikala & Mikkonen 2011, 61-62). Vesiputousmallia käytettäessä vaatimusmäärittely tehdään tavallisesti ohjelmiston vaatimusmäärittelydokumentin (engl. software requirements specification) ja/tai käyttötapausten (engl. use case) muotoon. Ketterissä menetel-

missä vaatimusten määrittelyyn käytetään yleensä käyttäjätarinoita (engl. user story). (Nazzaro & Suscheck 2010.)

Esitutkimuksen perusteella havaittiin, että projektissa oli runsaasti tekijöitä, joita ei voitu tietää riittävän hyvin ennalta. Lisäksi projekti oli toimeksiantajan sisäinen kehitysprojekti. Näillä perustein ketterän lähestymistavan valinta oli selvää. Vaatimusmäärittely tehtiin suoraan käyttäjätarinoiden muotoon (taulukko 6). Käyttäjätarinat kirjoitettiin toimeksiantajan kanssa käytyjen neuvottelujen pohjalta. Projektin kuluessa käyttäjätarinoita jonkin verran muokattiin, jaettiin pienemmiksi ja yhdisteltiin. Muutama uusikin tarina oli tarpeen lisätä.

Käyttäjätarinat pyrkivät noudattamaan yleistä formaattia: <roolissa> tahdon <pystyä tekemään jotakin> jotta <seuraa hyötyä>. Käyttäjätarinoille annettiin yksilöivä tunnus (merkintä: US + numero). Käyttäjätarinat saivat tunnuksen siinä järjestyksessä, kun ne kirjoitettiin. Käytössä oli kolme erityyppistä käyttäjätarinaa: toiminnallisen vaatimuksen esittävä käyttäjätarina eli järjestelmän toimintaa kuvaava vaatimus (merkintä: FR) ja ei-toiminnallisen vaatimuksen eli laatuvaatimuksen esittävä käyttäjätarina (merkintä: NFR) ja epiikka (engl. epic) eli hyvin suuri käyttäjätarina, joka ei sellaisenaan sovellu toteutettavaksi, vaan se on jaettava edelleen joukoksi pienempiä toteuttamiskelpoisia käyttäjätarinoita (merkintä: EPIC). (Cobb 2015, 65–67.) Lisäksi ohjelmistolle esitettiin reunaehtoja (engl. constraints), mutta ne eivät olleet käyttäjätarinoiden muodossa, vaan ne esitettiin taulukkona projektisuunnitelmassa. Vain osa käyttäjätarinoista valittiin toteutettavaksi opinnäytteessä prototyyppiin (merkintä: Esi-Alfa-1). Loput tarinat jäivät odottamaan myöhempiä kehitysvaiheita tuotteen backlogiin (engl. product backlog) (merkintä: Backlog).

Taulukko 6. Maksuvälitysjärjestelmän vaatimusmäärittely käyttäjätarinoiden muodossa. Käyttäjätarinoita päivitettiin projektin kuluessa, joten käyttäjätarinat esitetään taulukossa siinä tilassa kuin ne olivat opinnäytetyöprojektin päättyessä.

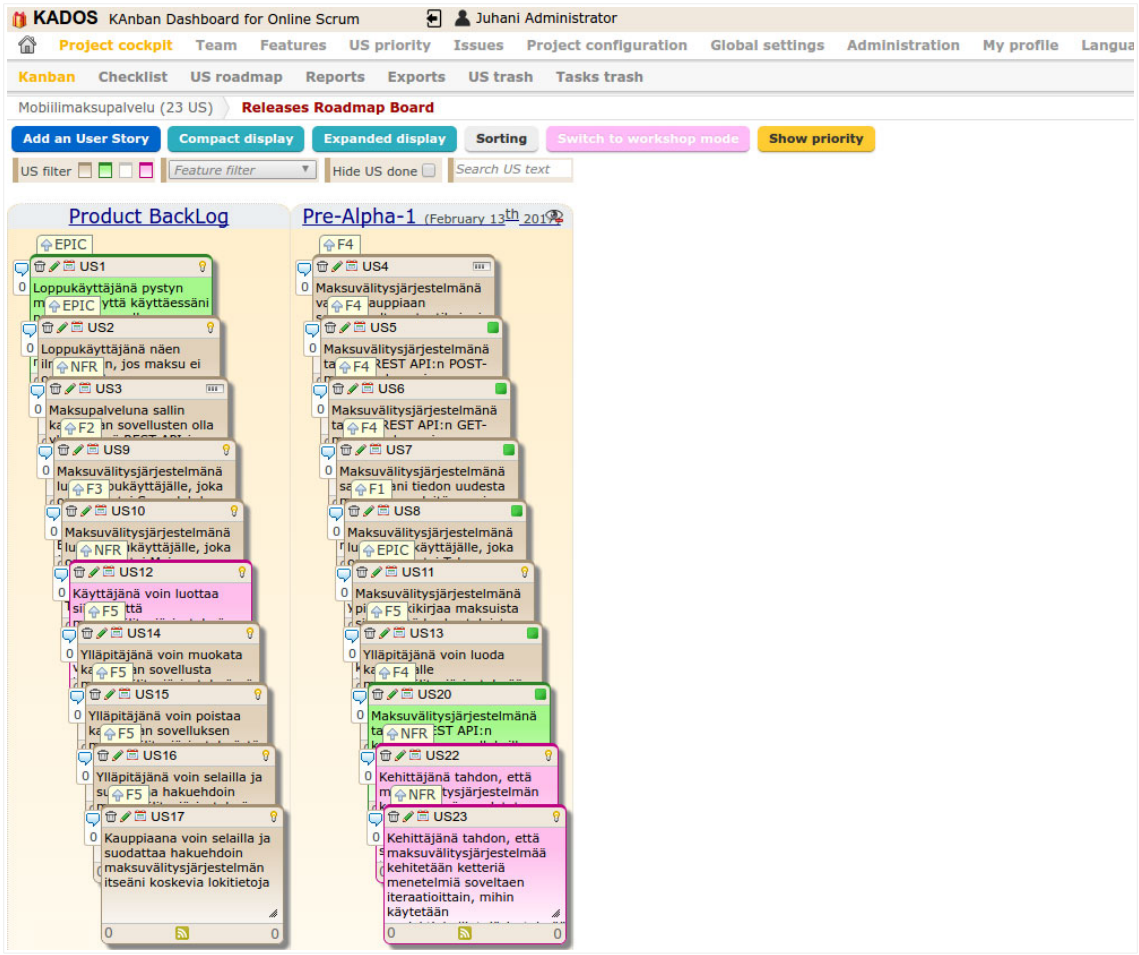
Id	Tarina	Tyyppi	Julkaisu
US1	Loppukäyttäjänä pystyn mobiili-internetyhteyttä käyttäessäni nappia painamalla maksamaan ostoksen, mistä operaattori veloittaa minua osana yhteyskuluja	EPIC	Backlog
US2	Loppukäyttäjänä näen ilmoituksen, jos maksu ei onnistunut	EPIC	Backlog
US3	Maksuvälitysjärjestelmänä sallin kauppiaan sovellusten olla yhteydessä REST API:iin vain SSL-suojattuna	FR	Backlog
US4	Maksuvälitysjärjestelmänä vaadin kauppiaan sovellukselta autentikoinnin REST API:n käyttöön	FR	Esi-Alfa-1
US5	Maksuvälitysjärjestelmänä tarjoan REST API:n POST-metodilla kauppiaan sovelluksen luoda uuden maksutapahtuman	FR	Esi-Alfa-1
US6	Maksuvälitysjärjestelmänä tarjoan REST API:n GET-metodilla kauppiaan sovellukselle tiedot maksutapahtuman tilasta	FR	Esi-Alfa-1
US7	Maksuvälitysjärjestelmänä saadessani tiedon uudesta maksusta, selvitän ensin, minkä operaattorin asiakas maksaja on eli operaattorin rajapintaa käytetään maksun veloittamiseen	FR	Esi-Alfa-1
US8	Maksuvälitysjärjestelmänä luon loppukäyttäjälle, joka on Soneran tai Tele Finlandin liittymäasiakas, uuden maksun ottamalla yhteyden Soneran mobiilimaksu-API:iin ja saan vastaukseksi tiedon, onnistuiko vai epäonnistuiko maksu ja maksutapahtuman tunnisteen	FR	Esi-Alfa-1
US9	Maksuvälitysjärjestelmänä luon loppukäyttäjälle, joka on Elisan tai Saunalahden liittymäasiakas, uuden maksun ottamalla yhteyden Elisan mobiilimaksu-API:iin ja saan vastaukseksi tiedon, onnistuiko vai epäonnistuiko maksu ja maksutapahtuman tunnisteen	FR	Backlog
US10	Maksuvälitysjärjestelmänä luon loppukäyttäjälle, joka on DNA:n tai Moin liittymäasiakas, uuden maksun ottamalla yhteyden Tieto Connectionin mobiilimaksu-API:iin ja saan vastaukseksi tiedon, onnistuiko vai epäonnistuiko maksu ja maksutapahtuman tunnisteen	FR	Backlog
US11	Maksuvälitysjärjestelmänä pidän lokikirjaa maksuista siten, että keskusteluista sekä kauppiaan sovelluksen että operaattorin kanssa on kerätty ja yhdistetty toisiinsa kummankin osapuolen maksutapahtuman tunnisteet, summat, statustiedot ja tilitiedot	FR	Backlog
US12	Käyttäjänä voin luottaa siihen, että maksuvälitysjärjestelmän tietoturva on aina kunnossa ja mitään tietoja ei koskaan voi joutua ulkopuolisten haltuun	NFR	Backlog
US13	Ylläpitäjänä voin luoda kauppiaalle maksuvälitysjärjestelmään sovelluksen	FR	Esi-Alfa-1
US14	Ylläpitäjänä voin muokata kauppiaan sovellusta maksuväli-	FR	Backlog

	tysjärjestelmässä		
US15	Ylläpitäjänä voin poistaa kauppiaan sovelluksen maksuvälitysjärjestelmästä	FR	Backlog
US16	Ylläpitäjänä voin selailla ja suodattaa hakuehdoin maksuvälitysjärjestelmän lokitietoja	FR	Backlog
US17	Kauppiaana voin selailla ja suodattaa hakuehdoin maksuvälitysjärjestelmän itseäni koskevia lokitietoja	FR	Backlog
US18	Kauppiaana voin selailla ja suodattaa hakuehdoin maksuvälitysjärjestelmän käytöstä veloitettuja ja maksatettavia maksuja perusteineen	FR	Backlog
US19	Ylläpitäjänä voin selailla ja suodattaa hakuehdoin maksuvälitysjärjestelmän käytöstä kaikilta eri kauppiailta veloitettuja maksuja perusteineen	FR	Backlog
US20	Maksuvälitysjärjestelmänä tarjoan REST API:n kauppiaiden sovelluksille	EPIC	Esi-Alfa-1
US21	Käyttäjänä tahdon, että maksuvälitysjärjestelmä toimii luotettavasti, vakaasti ja ilman vikoja, eikä myöskään järjestelmän jatkokehitys aiheuta epävakautta, joten järjestelmän toiminnallisuudet testataan automatisoidusti	NFR	Backlog
US22	Kehittäjänä tahdon, että maksuvälitysjärjestelmän kehityksessä noudatetaan version- ja konfiguraationhallintaa ja sen kehitys- ja tuotantoversiot on erotettu toisistaan, jotta kehittäjien yhteistyö, ohjelmakoodin versiointi, muutosten hallinta ja varmennus onnistuvat hyvin	NFR	Esi-Alfa-1
US23	Kehittäjänä tahdon, että maksuvälitysjärjestelmää kehitetään ketteriä menetelmiä soveltaen iteraatioittain, mihin käytetään projektinhallintajärjestelmää, jotta tehtävien hahmottaminen, hallinta, arviointi, seuranta ja jakaminen onnistuvat hyvin	NFR	Esi-Alfa-1

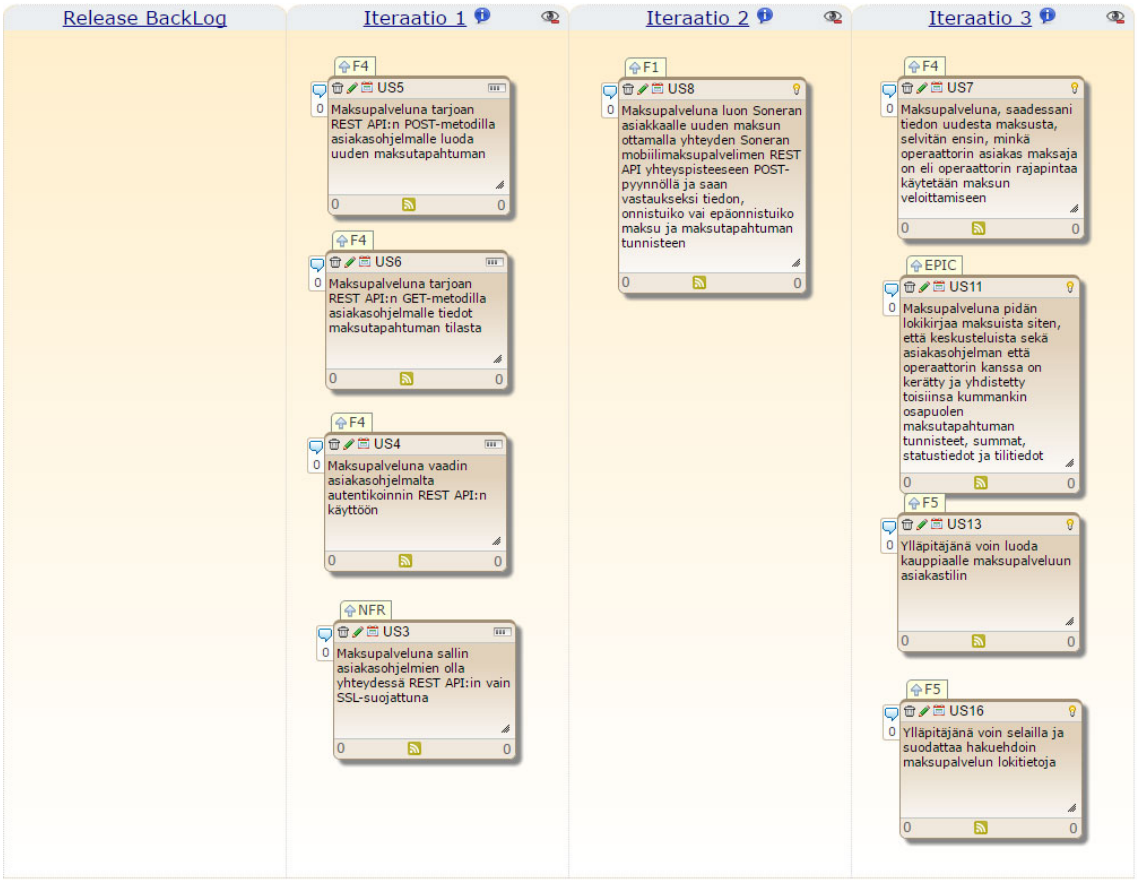
4.5 Projektinhallinta käytännössä

Projektinhallintaan käytettiin web-pohjaista avoimen lähdekoodin Kados-projektinhallintaohjelmaa, joka on tarkoitettu ketteriä projekteja varten. Kados-ohjelmiston käyttö toteutti NFR-käyttäjätarinan US23. Kados asennettiin web-palvelimelle ja käyttäjätarinat kirjoitettiin ”virtuaalisille lapuille” järjestelmän backlogiin. Ensin käyttäjätarinat lajiteltiin ohjelmiston ominaisuuksittain ja samalla otettiin käyttöön EPIC- ja NFR-käyttäjätarinatyypit. Seuraavaksi valittiin käyttäjätarinat Esi-Alfa-1-prototyyppiin (kuvio 3). Tämän jälkeen järjestelmään lisättiin 1-3 vii-

kon iteraatioita Esi-Alfa-1-prototyypille ja käyttäjätarinat sijoitettiin iteraatioihin (kuvio 4).



Kuvio 3. Kuvakaappaus Kados-projektinhallinnasta julkaisujen suunnittelunäkymästä. Osa käyttäjätarinoista oli backlogissa ja osa oli siirretty Esi-Alfa-1-prototyypissä tehtäväksi. EPIC-tarinat näkyivät vihreällä ja NFR-tarinat vaaleanpunaisella. Muilla ”virtuaalisilla lapuilla” oli FR-tarinoita, jotka oli merkitty ominaisuuksittain F1-F5.

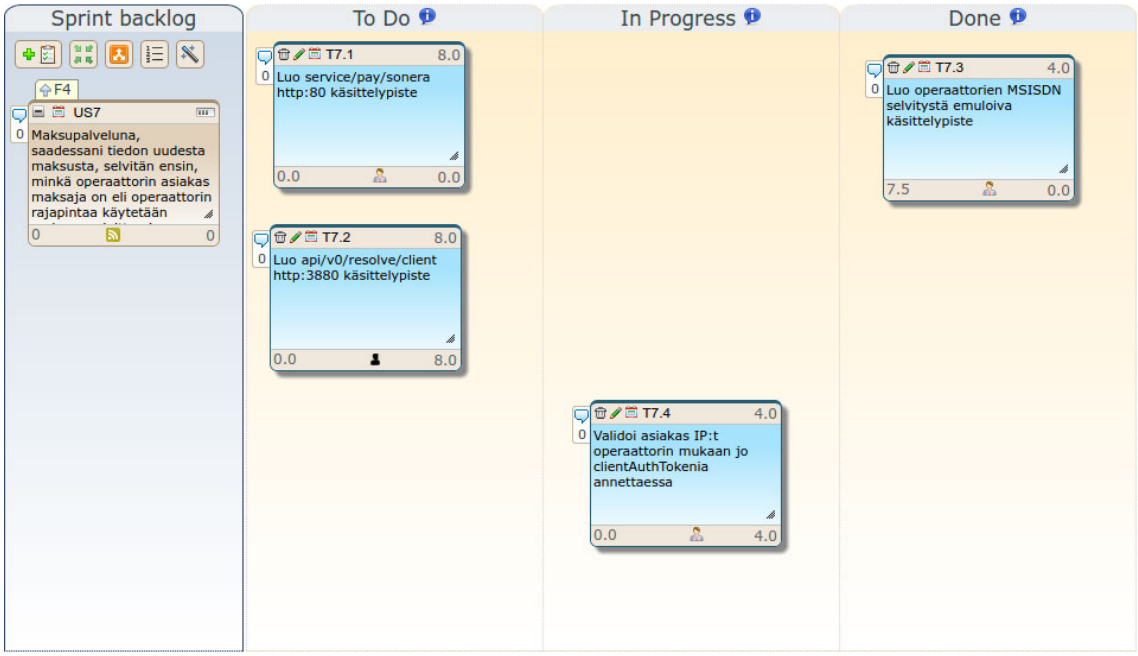


Kuvio 4. Kuvakaappaus Kados-projektinhallinnasta iteraatioiden suunnittelunäkymästä. Esi-alfa-1 julkaisuun valitut käyttäjätarinat sijaitsevat julkaisun backlogissa, josta ne sijoitettiin julkaisun iteraatioihin. Kuvakaappauksen ottamisen jälkeen julkaisun, iteraatioiden ja käyttäjätarinoiden sisältö muuttui projektin edetessä.

Kados tarjosi mahdollisuuden käyttäjätarinoiden priorisointiin suhteuttamalla käyttäjätarinan työmäärä sen liiketoiminta-arvoon, jolloin prioriteetissa korkeimpana olisivat käyttäjätarinat, jotka tuottaisivat suurimman liiketoiminta-arvon pienimmällä työmäärällä. Ominaisuutta ei kuitenkaan käytetty, vaan käyttäjätarinoiden työstäminen aloitettiin siinä järjestyksessä kuin se teknisen toteutuksen kannalta vaikutti kulloinkin mielekkäimmältä. Käyttäjätarinoita oli tarpeen sijoitella uudelleen lähes joka iteraation jälkeen. Esimerkiksi, jos jotain käyttäjätarinan ei saatu kokonaan tehtyä iteraation aikana, se sijoitettiin uudelleen johonkin seuraavaan iteraatioon.

Yksittäisen iteraation näkymässä kuhunkin käyttäjätarinaan oli liitettävissä ”virtuaalisille lapuille” tehtäviä, joista osa syötettiin kunkin iteraation aluksi ja osa

keksittiin vasta iteraation aikana. Tehtäville määriteltiin niiden oletettu aikaresurssien tarve puolen tunnin tarkkuudella. Tehtävät sijaitsivat taululla, jotka oli jaettu tehtävänä – tekeillä – tehty -sarakkeisiin (engl. To Do – In Progress – Done). Kun tehtävä otettiin tehtäväksi, ”lappua” siirrettiin tekeillä-sarakkeeseen. Tehtävän valmistuttua ”lappua” siirrettiin tehty-sarakkeeseen. Toiminnallisuutta on havainnollistettu yhden käyttäjätarinan osalta kuviossa 5. Tehtävään käytetty ja oletettu jäljellä oleva työtuntimäärä merkittiin ”virtuaaliselle lapulle” aikaajoin. Kadoksen raportointiominaisuudet ovat mielestäni heikot, joten niitä ei juuri käytetty.



Kuvio 5. Kuvakaappaus Kados-projektinhallinnasta yksittäisen iteraation näkymästä eli Sprint Backlogista, jossa käyttäjätarinaan kytketyt tehtävät on sijoitettu tilansa mukaisesti tehtävänä – tekeillä – tehty -sarakkeisiin. Kuvakaappauksen ottamisen jälkeen käyttäjätarinoiden ja tehtävien sisältö muuttui projektin edetessä.

4.6 Konfiguraation- ja versionhallinta

Tärkeä osa Drupal-web-sovellusprojekteja on konfiguraation- ja versionhallinta, joille sopiva yhteinen yläkäsite Haikalan ja Mikkosen (2011, 169) mukaan on tuotteenhallinta. Niin konfiguraation- kuin versionhallinnankin voi toteuttaa

usealla eri tavalla. Asiaan on otettava kantaa projektin suunnitteluvaiheessa, jotta toimintatapa on selvä toteutusta tehtäessä alusta alkaen. Konfiguraation- ja versionhallinnalla ei ollut opinnäytetyön aikana tehtävän prototyypin toteuttamisen kannalta suurta merkitystä, koska työtä tehtiin yksin ja ainoastaan paikallisessa kehitysympäristössä, mutta myöhemmässä vaiheessa tiimityöskentelyn ja tuotantokäyttöön otettavan ohjelmistotuotteen ylläpidon ja jatkokehityksen kannalta merkitys on ratkaisevan suuri. Lisäksi Drupal 8 -versio tuo konfiguraation- ja versionhallintaan huomattavia parannuksia (Borchert & Schirwinski 2015, 2–3), joten toimintatapa kannatti ottaa esille ja kokeiltavaksi myös opinnäytetyössä.

Ohjelmistot koostuvat komponenteista, joilla on erilaisia konfiguraatioita. Konfiguraatiolla tarkoitetaan, että komponenteista on erilaisia kokoonpanoja ja komponenteilla on erilaisia asetuksia. Sekä komponentit että konfiguraatiot voidaan versioida. Versio tarkoittaa ”jäädytettyä tilaa” ohjelmiston tiedostoista. Versioon ei tule versioimisen jälkeen tehdä enää muutoksia. Versiointi mahdollistaa vakaan työympäristön, jossa työskentely on hallittua, muutokset voidaan jäljittää ja kehittäjät eivät vahingossa häiritse toistensa työntekoa. Perättäiset versiot eli revisiot muodostavat versiopuun, joka voi tarvittaessa haarautua variaatioiksi esimerkiksi jonkin tietyn ominaisuuden tai korjauksen tekemiseksi. Versiointia hallitaan yleensä versionhallintaohjelmistoilla kuten Git tai Subversion. (Haikala & Mikkonen 2011, 170–174.)

Versionhallinta toteutetaan tiedostopohjaisesti, joten konfiguraatioiden versioiminen vaatii niiden tallentamista tietokannan sijaan tiedostoihin. Konfiguraatioiden hallinta käytännössä edellyttää ohjelmakoodista erillisten konfiguraatiotiedostojen käyttöä. Drupal 8:ssa konfiguraatiota voidaan tallentaa YAML-syntaksia noudattaviin `*.yml`-tiedostoihin. Esimerkiksi Drupalin tietorakenteet ja järjestelmän asetukset ovat konfiguraatioita. Tietoturvasyistä konfiguraatioiden tallennushakemisto kannattaa sijoittaa Drupal-asennuksen juuren ulkopuolelle, mikä vaatii asetusmuutoksen tekemistä. (Borchert & Schirwinski 2015, 14, 37–38; Drupal.org 2017å.)

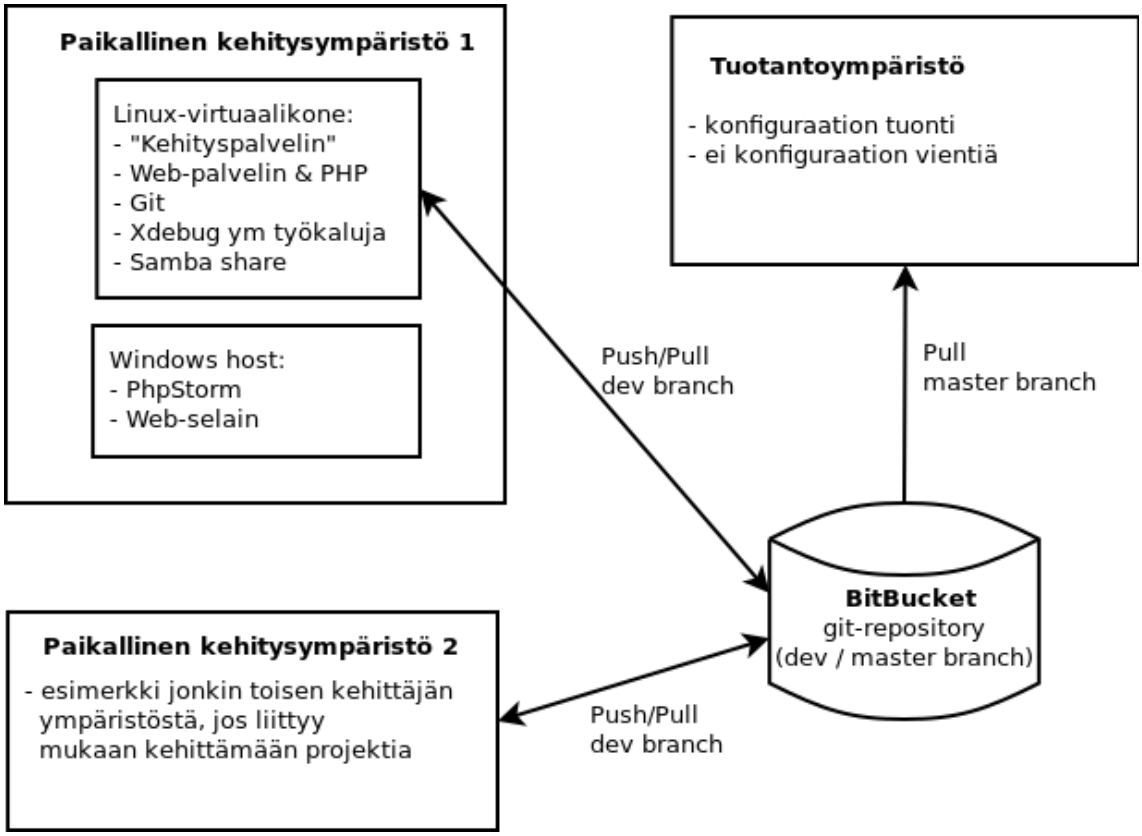
Drupalin konfiguraatiota kuten näkymiä tai sisältötyyppien rakenteita, muutetaan yleensä käyttöliittymän kautta. Konfiguraatiomuutosten versioiminen luonnolli-

sesti edellyttää, että jokaisen konfiguraatiomuutoksen jälkeen muutokset viedään tiedostomuotoon ja tiedostot kommitoidaan versionhallintaan. Sujuva työnkulku vaatii, että Drupalista on tällöin oltava vähintään kehitys- ja tuotantoasennukset, jolloin konfiguraatiomuutokset tehdään vain kehitysversiossa, eikä koskaan tuotantoympäristössä. Jotta käytäntö ei unohtuisi sivuston rakentajilta, voidaan konfiguraatioiden tekeminen tuotantoversiossa estää esimerkiksi ottamalla käyttöön Configuration Read-only mode -moduuli. (Bujisic 2016.) Konfiguraation vienti ja tuonti tiedostomuotoon onnistuu esimerkiksi Drupal Consolella komentoriviltä `drupal config:export` ja `drupal config:import`. Suositeltavan versionhallintaetiikan mukaisesti versionhallintaan ei viedä tietokantavedoksia, käyttäjähakemistoja eli sisältöön liittyviä tiedostoja, eikä kehitys- ja tuotantoversioiden paikallisia asetustiedostoja (Norton 2016). Versionhallinta ei siis korvaa tiedostojen ja tietokantojen varmuuskopiointia.

Edistyneemmässä Drupal 8 -versionhallinnassa käytetään Composer-riippuvuudenhallintatyökalua, jolloin Drupal-projekteissa viedään versionhallintaan core- ja contrib-moduuleiden ja -teemojen sijaan vain `composer.json`-konfiguraatiotiedosto, jossa määritellään projektiin kuuluvat komponentit. Tällöin muutokset kuten lisäykset, päivitykset ja poistot tehdään core- ja contrib-moduuleihin konfiguraatiotiedoston mukaisesti Composeria käyttäen. Composer-pohjaista työnkulkua ei opinnäytetyössä kuitenkaan otettu käyttöön, koska syksyllä 2016 Drupalin virallinen pakettihakemisto ei ollut vielä valmis ja useat contrib-moduulit eivät vielä tukeneet Composerin käyttöä. Lisäksi opinnäytetyöntekijän aikaisempien Composer-käyttökokemusten perusteella erilaisia riippuvuusristiriitoja esiintyi usein ja niiden ratkaiseminen oli työlästä. Drupal-asennusten ja -päivitysten tekemiseen käytettiin Symfony Consoleen perustuvaa Drupal Consolea, joka hyödyntää sisäisesti Composeria.

Projektisuunnitelmassa luonnosteltiin yksinkertainen malli konfiguraation- ja versionhallintaan (kuvio 6), jossa kehitysympäristöt olivat paikallisia ja vain tuotantoympäristö oli internet-palvelimella. Projektin aikana ei kuitenkaan tehty tuotantoympäristön käyttöönottoa, eikä projektissa ollut muita kehittäjiä, joten käytössä olivat vain paikallinen kehitysympäristö ja BitBucket-versionhallinta. Joka iteraation alkaessa versionhallintaan avattiin dev-haara ja iteraation päättyessä

dev-haara yhdistettiin master-haaraan. Jos tuotantoympäristö olisi ollut käytössä, vain master-haara olisi viety tuotantoon. Mallissa tuotantoympäristöstä ei koskaan viedä mitään versionhallintaan.



Kuvio 6. Projektia varten luonnosteltu malli mahdollisimman yksinkertaisesta versionhallinnan työnkulusta eri ympäristöjen välillä. Kehittäjät voivat tarvittaessa käyttää kukin omaa dev-haaraansa.

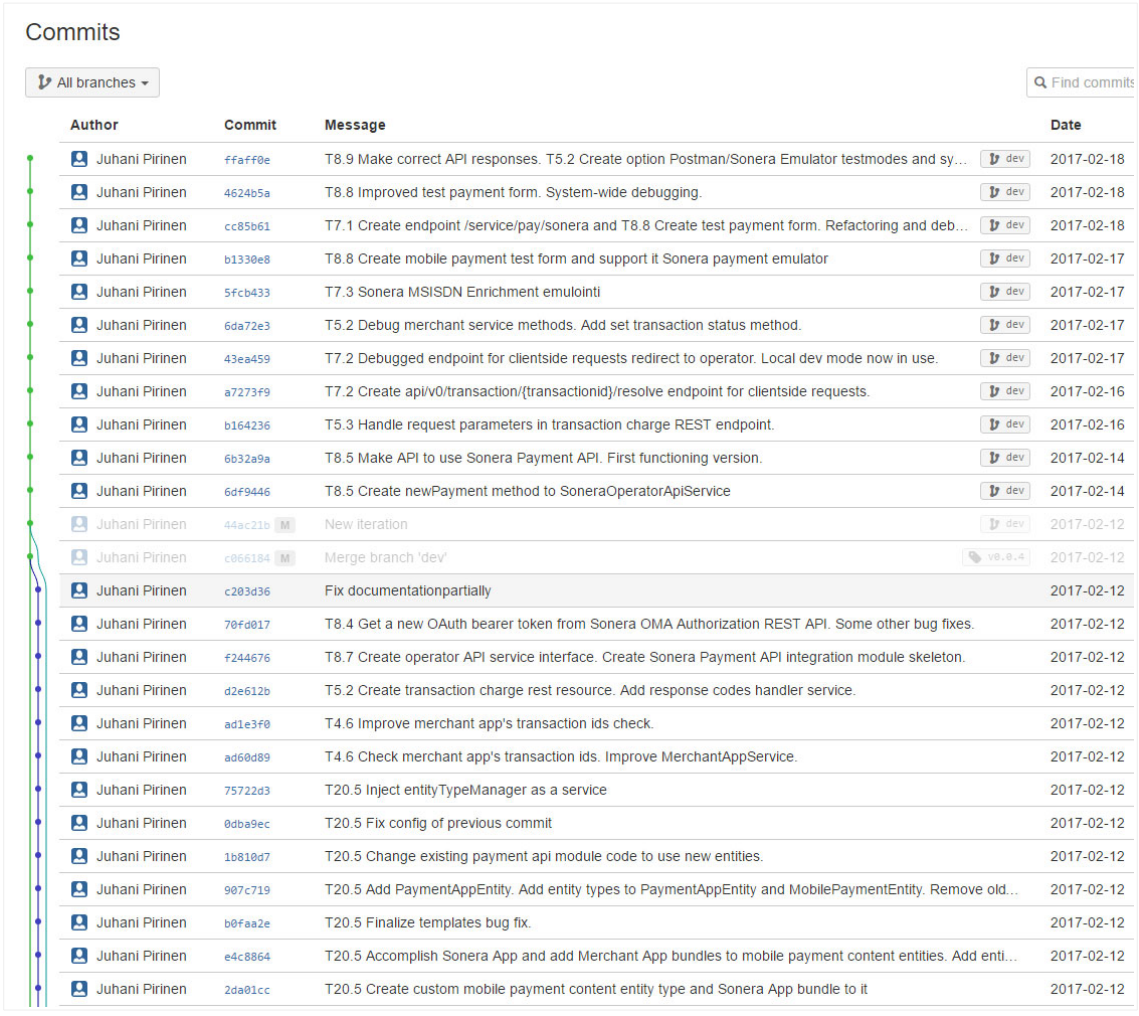
Versionhallinnan käyttöön on tarjolla useita eri työnkulkumalleja. Kehittäjien keskuudessa suosittuja ovat mm. *Git Flow*, joka sopii kun käytössä on useita tuotantoversioita ohjelmistosta (esimerkiksi työpöytäohjelmistot ja käyttöjärjestelmät); *GitHub Flow*, joka sopii kun käytössä on yksi tuotantoversio yksinkertaisemmasta ohjelmistosta (esimerkiksi web-sivut ja -palvelut); ja *GitLab Flow*, joka sopii kun käytössä on yksi tuotantoversio monimutkaisemmasta ohjelmistosta (esimerkiksi suuret web-palvelut) (Pathirage 2016). API-järjestelmistä voi olla useita tuotantoversioita, joten projektin alkuvaiheessa kokeiltiin Git Flowta. Jostakin syystä Git Flowta käytettäessä päädyttiin jatkuvasti suureen määrään

merge-virheitä, joten tilalle otettiin yksinkertainen *Simple Git Branching Model* (Benet 2014), jonka kanssa merge-virheitä ei esiintynyt. Esimerkki työnkulusta sovitettuna Drupal-kehittäjän työasemalle esitetään taulukossa 7. Työnkulkuun on liitetty Drupal Consolen käyttö konfiguraation viemiseksi versionhallintaan.

Taulukko 7. Simple Git Branching Model -työnkulku (Benet 2014) Drupal-kehittäjän Linux-työasemalle soveltaen. Kehitystyö tehdään dev-haarassa ja kehitystyön valmistuttua työ yhdistetään master-haaraan. Kehittäjät voivat tarvittaessa käyttää kukin omaa dev-haaraansa. Komennot suoritetaan Drupal-juuressa, paitsi komento 6 Drupal-konfiguraation synkronisaatiohakemistossa.

#	Komennot	Selitys
1	git checkout master	Siirrytään master-haaraan.
2	git pull origin master	Tuodaan master-haaran sisältö työasemalle, jos ei ajan tasalla.
3	drupal config:import	Tuodaan Drupal-konfiguraatio synkronisaatiohakemistosta, jos siinä on muutoksia.
4	git checkout -b dev	Siirrytään dev-haaraan. Jos dev-haara on jo olemassa, ei käytetä -b valintaa komennossa.
5	Kohdassa 5 koodia työstetään paikallisessa kehitysympäristössä. Muutos versioidaan toteuttamalla vaiheet 6-11.	
6	rm *.yml	Drupal Console ei poista vanhoja konfiguraatiotiedostoja sykronisaatiohakemistosta, joten poistetaan ne käsin, jotta konfiguraation vienti menee oikein. Ei poisteta .htaccess-tiedostoa.
7	drupal config:export	Viedään Drupal-konfiguraatio synkronisaatiohakemistoon.
8	git add --all .	Lisätään mahdolliset uudet tiedostot. Viitattava Git-juureen.
9	git commit -m "T1.2 Make these changes"	Viedään muutokset paikalliseen versionhallintaan.
10	git fetch origin git rebase origin/dev git rebase origin/master	Pidetään molemmat haarat ajantasalla. Käytetään rebase-komentoa pitämään koodi toimivana, merge helppona ja historia siistinä. Jos dev-haara on uusi, eikä sinne ei ole vielä koskaan viety mitään, sille ei anneta rebase-komentoa.
11	git push origin dev	Viedään muutokset versionhallintapalvelimelle. Valinnainen.
12	Jos kehitystyö on valmis, siirrytään kohtaan 13. Muutoin palataan jatkamaan kohtaan 5.	
13	git checkout master	Siirrytään master-haaraan.
14	git pull origin master	Tuodaan master-haaran sisältö työasemalle, jos ei ajan tasalla.
15	drupal config:import	Tuodaan Drupal-konfiguraatio synkronisaatiohakemistosta, jos siinä on muutoksia.
16	git merge --no-ff dev	Liitetään dev-haara master-haaraan siten, että dev-haara jää näkyviin master-haarasta erillisenä ”kuplana”.
17	git push origin master	Viedään muutokset master-haaraan versionhallintapalvelimelle.
18	git tag 1.0.0-RC1 git push origin 1.0.0-RC1	Tarvittaessa merkitään tägillä master-haaraan esimerkiksi versionumero ja viedään versionhallintaan. Valinnainen.

Ohjelmistojen versionhallintaan tehdyt kommit-viestit ovat sekavaa luettavaa, jos niiden rakenne ei ole yhtenäinen. Projektia varten kommit-viesteille muodostettiin rakenne, joka aloitettiin käyttäjätarinaan liittyvän tehtävän koodilla, jota seurasi kommittia kuvaava verbi imperatiivimuodossa. Lausetta jatkettiin lyhyellä kuvauksella, joka vastasi kysymykseen ”mitä?”. (Beams 2014; Borchert & Schirwinski 2015, 5.) Rakennetta noudattaen kommit-viesti saattoi kuulua esimerkiksi näin: `T5.3 Handle request parameters in transaction charge REST endpoint`. Kuvakaappaus kommit-viesteistä versionhallinnassa on esitetty kuviossa 7.



Kuvio 7. Kuvakaappaus BitBucket-versionhallintajärjestelmän käyttöliittymästä. Kuvassa näkyvät versionhallintaan tehdyt kommitit. Meneillään olevan iteraation kommitit olivat vielä dev-haarassa, kun taas edellisen iteraation kommitit oli jo yhdistetty master-haaraan ja merkitty versiotägillä.

5 Projektin toteutus

5.1 Iteraatio 1: API-kokeilu Drupal core- ja contrib-moduuleilla

Iteraatiossa 1 otettiin tavoitteeksi kokeilla, miten pitkälle maksuvälitysjärjestelmän toteutus oli mahdollista olemassa olevilla Drupal core- ja contrib-moduuleilla, ilman ohjelmointia eli omia custom-moduuleita. Näin pyrittiin selvittämään, missä määrin niihin voidaan projektissa tukeutua ja mitä omia custom-moduuleita tarvitaan niiden lisäksi. Selvää oli, että oma custom-moduuli tarvittiin ainakin mobiilioperaattoreiden integroimiseksi, mutta kiinnostavaa oli selvittää tilanne maksuvälitysjärjestelmän oman rajapinnan toteuttamisen kannalta.

Drupal-projekteissa yleensä, kun vaatimuksia on ensin määritelty, on seuraavaksi tarpeen kartoittaa projektin näkökulmasta, millä Drupal-moduuleilla mikäkin toiminnallisuus voidaan parhaiten toteuttaa. Mitä pienemmällä määrällä moduuleja selvitään, sen parempi. (Hakimzadeh, Melançon & Nordin 2011, 87–88.) Drupal-projekteissa kannattaa pyrkiä käyttämään mahdollisimman pitkälle käyttövalmiita core- ja contrib-moduuleita. Core-moduulit ovat hyvin tuettuja ja vakaita (Drupal.org 2016aq), kun taas contrib-moduuliprojektien aktiivisuus ja laatu vaihtelee paljon (Hakimzadeh ym. 2011, 88–90). Moduuleiden onnistunut valinta edellyttää kehittäjältä hyvää Drupal-asiantuntemusta.

Omia custom-moduuleita tarvitaan, kun projekti vaatii toiminnallisuuksia, joihin valmiiden moduuleiden ominaisuudet eivät riitä tai ne eivät vastaa täysin haluttua (Lorétan 2011, 501). Koska custom-moduuleiden kehitys vaatii projektilta aina resursseja, saattaa joissain tapauksissa olla järkevää harkita, voisiko vaatimuksissa hieman joustaa siten, että toiminnallisuuden toteuttaminen onnistuisi mahdollisimman pitkälle käyttämällä uudelleen jo olemassa olevia moduuleita.

Iteraation aluksi siihen valittiin tavoitteeseen sopivat käyttäjätarinat ja niille liitettiin tehtävät, jotka on esitetty taulukossa 8. Osa tehtävistä lisättiin iteraation kuluessa. Iteraatio 1:n alkaessa kehitysympäristö oli jo asennettu, Drupal 8.0.x oli

asennettu minimal-asennusprofiililla ja tarvittavat työkalut kuten Drupal Console, Drush, Git, PHPStorm ja Postman olivat käyttövalmiina.

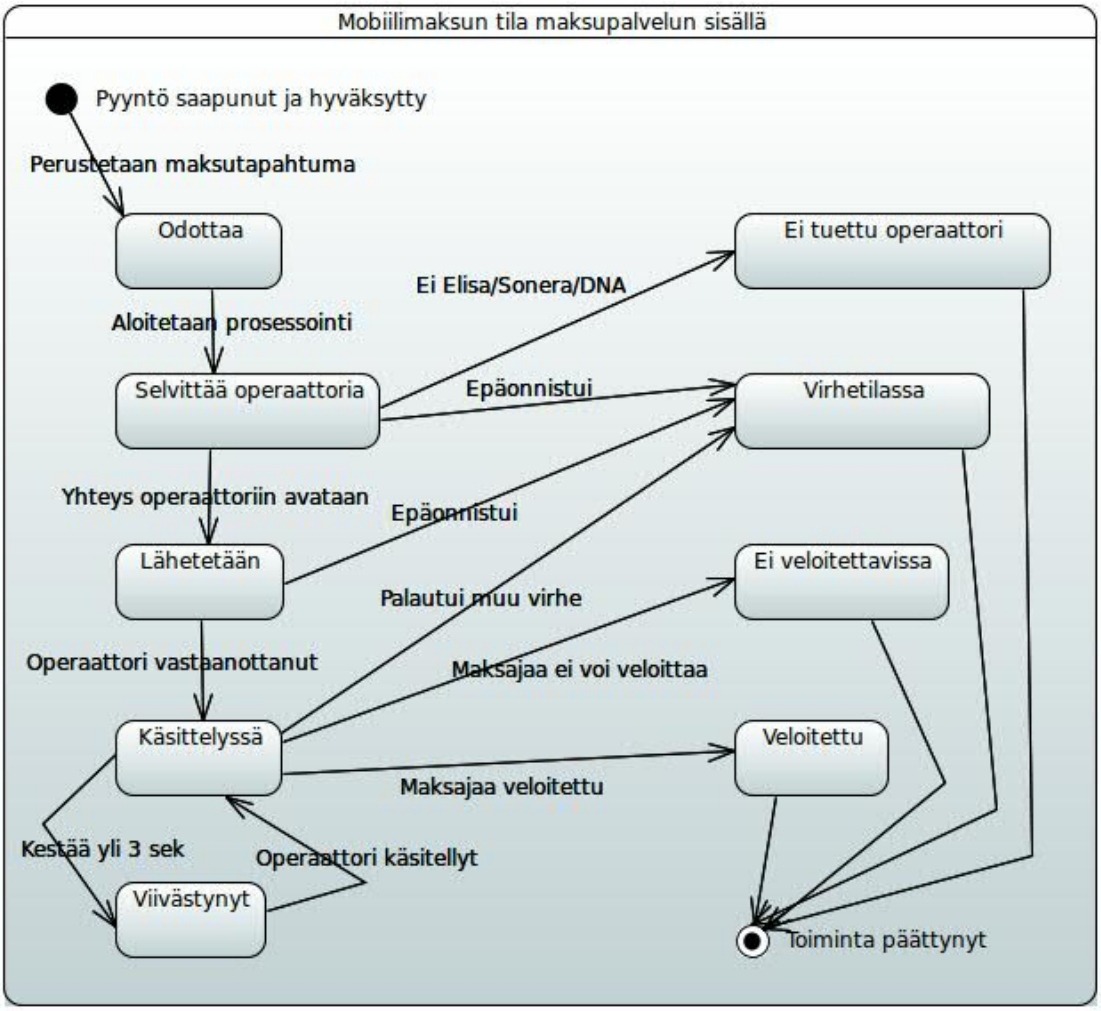
Taulukko 8. Iteraatio 1:n tavoite, siihen valitut käyttäjätarinat ja kuhunkin käyttäjätarinaan liittyvät tehtävät.

Iteraatio 1:n tavoite: Kokeillaan, miten pitkälle maksuvälitysjärjestelmän API:n toteutus on mahdollinen olemassa-olevilla Drupal core- ja contrib-moduuleilla, ilman ohjelmointia.	
Käyttäjätarinat:	**Tehtävät:**
US4 Maksuvälitysjärjestelmänä vaadin kauppiaan sovellukselta autentikoinnin REST API:n käyttöön (FR)	T4.1 Tutki käyttövalmiit autentikointimenetelmät (Oauth2, Basic Auth, ym) T4.2 Ota käyttöön jokin käyttövalmiista autentikointimenetelmistä T4.3 Testaa autentikoinnin toimivuus
US5 Maksuvälitysjärjestelmänä tarjoan REST API:n POST-metodilla kauppiaan sovelluksen luoda uuden maksutapahtuman (FR)	T5.1 Maksutapahtuma-sisältötyyppi T5.2 Testaa maksutapahtuman luonti REST API:n kautta
US6 Maksuvälitysjärjestelmänä tarjoan REST API:n GET-metodilla kauppiaan sovellukselle tiedot maksutapahtuman tilasta (FR)	T6.1 Testaa maksutapahtuman tietojen saanti REST API:n kautta T6.2 Maksutapahtuman View Rest Export
US20 Maksuvälitysjärjestelmänä tarjoan REST API:n kauppiaan sovellukselle (EPIC)	T20.1 Suunnittele maksuvälitysjärjestelmän REST API:n alustava versio T20.2 Drupal moduulien asennus

Ensimmäisenä tehtävänä käyttöön otettiin core-moduulit (tehtävä T20.2) RESTful Web Services, Serialization, Views ja Views UI, jotka mahdollistivat mm. ytimen REST-ominaisuuksien ja JSON-serialisoinnin käytön sekä näkymien REST Export -toiminnallisuuden kokeilemisen. Moduuleiden asentaminen onnistui Drupal Consolea käyttäen komentoriviltä Drupal-asennuksen juuressa helposti, esimerkki: `drupal module:install rest serialization views views_ui`.

Maksutapahtuman sisältötyypin (engl. content type) (tehtävä T5.1) luomista varten koetettiin ensin hahmottaa, mitä eri tiloja maksulla voisi olla järjestelmän sisällä. Papyrus-mallinnusohjelmaa käyttäen luotiin UML-tilakonekaavio maksun tiloista (kuvio 8). Seuraavaksi luotiin uusi sisältötyyppi Payment ja asennettiin

sitä varten moduulit Field UI ja Options. Sisältötyyppiin konfiguroitiin kentät, jotka on esitetty taulukossa 9.



Kuvio 8. Mobiilimaksun tila maksuvälitysjärjestelmän sisällä UML-tilakonekaaviona. Mallinnus osoittautui iteraatiossa 4 osittain epärelevantiksi.

Taulukko 9. Payment-sisältötyypin custom-kentät kuvauksineen ja määrittelyineen. Myöhemmässä vaiheessa sisältötyypin nimi ja rakenne muuttuivat.

Kentän nimi	**Kentän tyyppi**	**Kuvaus ja määrittely**
title	Title	Node-sisältötyyppiä käytettäessä title-kenttä on pakollinen, eikä sitä ole mahdollista poistaa. En keksinyt tälle vielä tässä vaiheessa kentälle käyttötarkoitusta. Koska kenttää ei saa jättää tyhjäksi, siihen voi toistaiseksi laittaa sisällöksi vaikkapa ”mobiilimaksu”.
field_state	List (text)	Maksun tila. Vaadittu kenttä. Arvoksi sallittu vain yksi avain-arvo-pareista. Oletusarvo avain 0. 0|Pending 1|Resolving 2|Sending 3|Processing 4|Delayed 5|Charged 6|Unchargeable 7|Unsupported 8|Error
field_operator	List (text)	Maksun veloittava operaattori. Vaadittu kenttä. Arvoksi sallittu vain yksi avain-arvo-pareista. Oletusarvo avain 0. 0|Unresolved 1|Unsupported 2|Tieto 3|Elisa 4|Sonera
field_end_user	Text (plain)	Mobiilioperaattorin asiakkaan matkapuhelinnumero.
field_request	Text (plain, long)	Loki maksun pyynnöistä, jos virheitä.
field_response	Text (plain, long)	Loki maksun vastauksista, jos virheitä.
field_payment_amount	Number (decimal)	Maksun summa kahden desimaalin tarkkuudella. Vaadittu kenttä. Maksimiarvo 60,00.
field_payment_ description	Text (plain)	Maksun kuvaus, joka näkyy puhelinlaskulla. Vaadittu kenttä.
field_payment_ reference_code	Text (plain)	Maksun viitekoodi.
field_payment_currency	List (text)	Maksun valuutta ISO 4217 mukaan. Oletusarvo EUR. Vaadittu kenttä.
field_payment_amount_ charged	Number (decimal)	Onnistuneesti veloitettu maksun summa kahden desimaalin tarkkuudella.

Jotta maksutapahtuma voitiin luoda REST-API:n kautta, Drupal edellytti autentikoinnin käyttöönottoa (tehtävät T4.1, T4.2 ja T4.3). Koska autentikoinnin toiminta REST-vientinäkymässä vaati uudempaa Drupal 8.2.x versiota, piti Drupal-ydin päivittää tähän versioon. Drupal Consolen uusinkaan versio ei vielä tukenut Drupal-ytimen päivityksiä, joten tehtävään piti käyttää Drushia.

Drupal coren mukana tuleva Basic Auth -autentikointimenetelmän RFC 2069 (IETF 1997) tarjoava moduuli toimi moitteettomasti (tehtävä T4.3). Maksutapahtuman luominen (tehtävä T5.2) Postman-testityökalua käyttäen onnistui Drupalin node-REST-resurssia kutsuen polussa `/entity/node`. Sisältönodea luotaessa JSON-arvoille oli käytettävä pakollisia title- ja type-kenttiä sekä `field_`-määreellä alkavia kenttiä (kuvio 9). Sessio piti ensin avata kutsumalla polkua `/rest/session/token` mikä palautti token-arvon, joka varsinaisessa REST-luontipyynnössä oli sijoitettava `X-CSRF-Token` HTTP-otsikkokentän arvoksi. Ominaisuudella pyritään estämään CSRF-hyökkäykset (OWASP 2016). Paluuviestinä Drupal palautti automaattisesti kokonaisen luodun sisältönoden eli pyyntö sisälsi noden kaikki mahdolliset kentät.

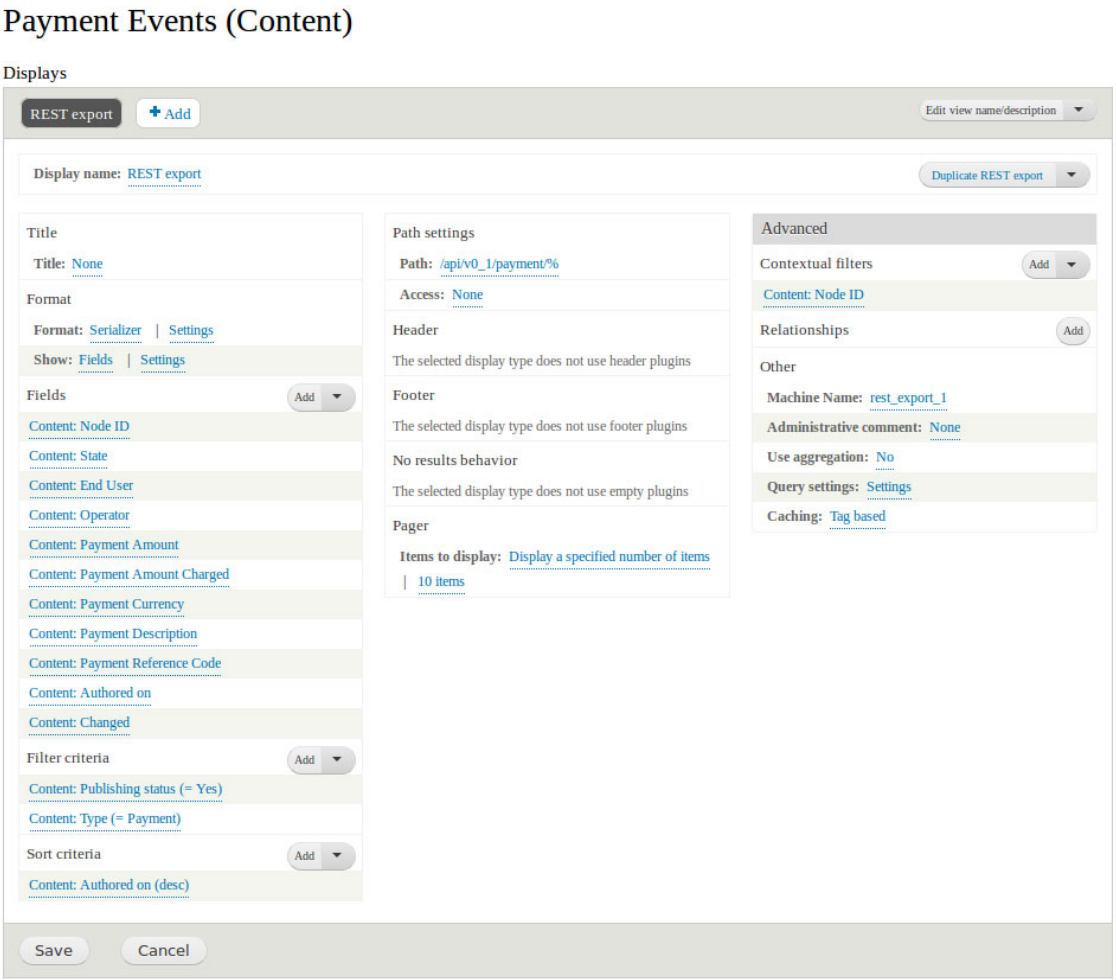
```
{
    "title": [
    {
    "value": "mobiilimaksu"
    }
  ],
    "type": [
    {
    "target_id": "payment"
    }
  ],
    "field_end_user": [
    {
    "value": "tel:+3580000000000"
    }
  ],
    "field_payment_amount": [
    {
    "value": "15.90"
    }
  ],
  "field_payment_description": [
    {
    "value": "Onko taas perjantai, pikku kaniini?"
    }
  ]
}
```

Kuvio 9. Esimerkki payment-sisältötyyppiä vastaavan sisällön luontiin tarvittavasta JSON-koodista Drupal-ytimen node-REST-resurssia käytettäessä.

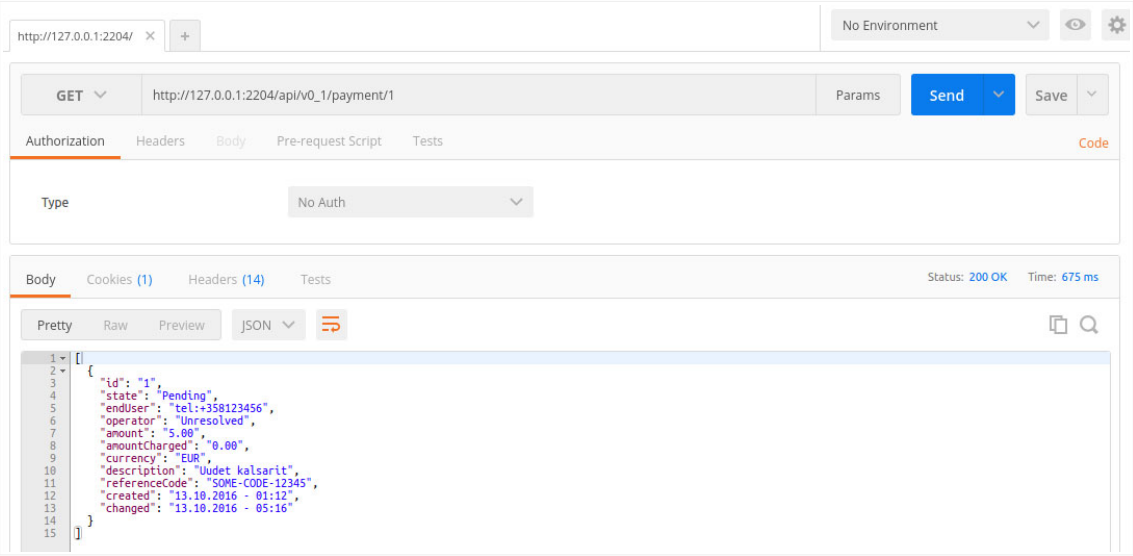
Myös muita autentikointimenetelmiä tutkittiin (tehtävä T4.1). Simple OAuth contrib-moduuli tarjosi Basic Authia kehittyneempää OAuth2 Bearer Token -autentikointia RFC 6749 (IETF 2012), mutta tätä ei jostakin syystä saatu toimimaan. Lisäksi moduulin käyttöliittymä ei sellaisenaan vaikuttanut sopivalta maksuvälitysjärjestelmän käyttöön. Toinen contrib-moduuli, OAuth2 Server vaikutti lupaavalta, mutta sen Drupal 8 -version kehitys oli vielä täysin kesken ja projektiin oli viimeksi koskettu yli vuosi sitten. Myöskään saatavilla ollutta OAuth 1.0 -standardin mukaista contrib-moduulia ei kokeiltu, koska OAuth 1.0 -standardi on OAuth 2.0 -standardia vanhempi ja mutkikkaampi ja siten enää harvoin API:eissa nykyisin käytetty (Siriwardena 2014, 91). Näin ollen Esi-Alfa-1-prototyypissä käytetään vain Basic Auth -moduulia (tehtävä T4.2).

Maksutapahtumien REST-vientinäkymä (tehtävä T6.2) luotiin ja määriteltiin yksittäisen maksutapahtuman esittämiseen vastauksena kyselyyn järjestelmään tallentuneista maksutapahtumista (kuvio 10). Näkymä on yleensä tarkoitettu

useamman sisältönoden esittämiseen, joten viennin JSON-notaatio oli taulukkomuotoinen. REST-vientinäkymän käyttö oli ainoa ytimen tarjoama keino muotoilla vastauksesta halutunlainen, joten siksi sitä käytettiin myös yksittäisen maksutapahtuman esittämiseen (tehtävä T6.1). Jotta REST-vientinäkymä näyttäisi kullekin käyttäjälle vain heidän omia sisältöjään (tehtävä T4.3), näkymälle piti asettaa suhde (engl. relationship) parhaillaan sisäänkirjautuneeseen käyttäjään. Muista pääsyoikeuden hallintaan tarkoitetuista contrib-moduuleista, kuten Node View Permissions, ACL tai Content Access, ei yksikään tukenut REST-vientinäkymän pääsyoikeuksien hallintaa.

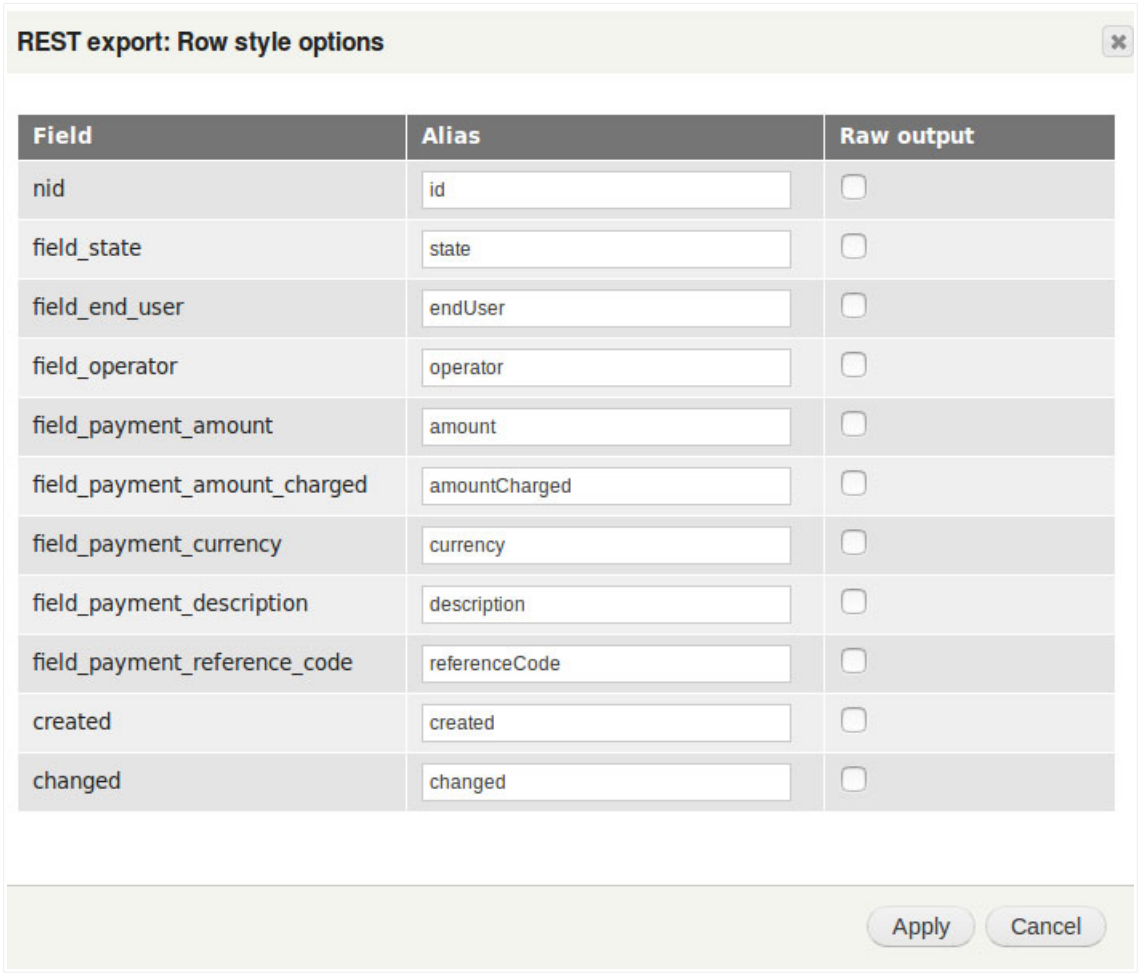


Kuvio 10. Views UI -moduulin tarjoama käyttöliittymä näkymän REST-viennin konfigurointiin. Tässä työvaiheessa näkymän käyttöoikeutta rajoittavaa suhdemääritystä ei vielä oltu tehty.



Kuvio 11. Kuvakaappaus Postman-testityökalusta. View-moduulin REST export toiminnolla pyyntöihin sai generoitua myös maksuvälitysjärjestelmän API:n kannalta kelvollisia JSON-vastauksia.

Maksuvälitysjärjestelmän REST API:n suunnittelun kannalta (tehtävä T20.1) jouduttiin toteamaan, etteivät Drupal core- ja contrib-moduulien tarjoamat REST-ominaisuudet sellaisenaan sovellu käytettäväksi maksuvälitysjärjestelmän toteuttamiseen. Iteraatiossa 1 havaittiin, että uutta maksutapahtumaa luotaessa REST-pyynnöt ja -vastaukset sisälsivät runsaasti kenttiä, joita kyllä tarvittaisiin esim. decoupled Drupal -tyyppisessä ratkaisussa sisällönhallintajärjestelmän back-end- ja front-end-puolten erottamiseksi toisistaan, mutta maksuvälitysjärjestelmän API:n kannalta valtaosa kentistä oli tarpeettomia. Sisältöä luotaessa käytettäviä `/entity/node` ja `/rest/session/token` polkuja ei ollut mahdollista sijoittaa uudelleen API:a varten polkuun `/api/v1/payment`. Myöskin coren tapa nimetä kentät ja mahdollisuus kenttien nimeämiseen uudelleen vain REST-vientinäkymässä (kuvio 12) oli API:n suunnittelun kannalta ei-toivottavaa. Sen sijaan jo aiemmin luotujen maksutapahtumien pyytäminen ja vastauksen saaminen REST-vientinäkymän avulla toimi hyvin (kuvio 11). Jälkikäteen iteraatiossa 4 huomattiin, että Services contrib-moduuli jäi iteraatiossa 1 kokonaan huomaamatta ja testaamatta, millä olisi saattanut olla vaikutusta iteraatiosta 1 tehtäviin johtopäätöksiin (taulukko 10).



Kuvio 12. View-moduulin REST export toiminto mahdollisti Drupal-kenttien nimeämisen uudelleen, jotta REST-vastauksesta saatiin halutun kaltainen. Samankaltaista toimintoa ei löytynyt Drupal-ytimestä sisällön luomisessa tarvittavien pyyntöjen ja niihin saatavien vastausten siistimiseen.

Taulukko 10. Itsearvio iteraatio 1:n tuloksista.

Yhteenveto iteraatio 1:n tuloksista	
Ajankäyttö:	• Toteutunut ajankäyttö: 35.5 h
Vahvuudet:	• Drupal-ytimen REST-toiminnallisuudet maksutapahtumia kyseltäessä olivat maksuvälitysjärjestelmän API:n kannalta pääosin käyttökelpoisia. • Drupal 8:n konfiguraationhallinta mahdollisti projektin pitämisen jatkuvasti versionhallinnassa. • Drupal Console, Drush ja Postman osoittautuivat suhteellisen päteviksi työkaluiksi Drupal- ja API-kehityksessä.
Heikkoudet:	• Simple Oauth contrib-moduuli ei soveltunut käytettäväksi sellaisenaan, eikä Oauth2 autentikointia saatu toimimaan. • Drupal-ytimen REST-toiminnallisuudet maksua luotaessa eivät olleet maksuvälitysjärjestelmän API:n kannalta käyttökelpoisia. • Maksun tilan määrittely tehtiin ennen riittävän tarkkaa tutustumista kaikkien operaattoreiden API:hin, joten tilakonekaavio osoittautui myöhemmin osittain virheelliseksi. • Versionhallinnan käyttö konfiguraatiomuutosten versioimisessa oli turhan pikkutarkkaa. • Tehdyt sisältötyyppimääritykset eivät tulisi sopimaan yhteen joskus myöhemmin kehitettävän katevaraus-maksutapahtumatyypin kanssa. • Services contrib-moduuli jäi iteraatiossa käsittelemättä kokonaan, millä olisi voinut olla vaikutusta iteraation tuloksista tehtäviin johtopäätöksiin.
Johtopäätökset:	• Drupal 8 -ytimen REST-ominaisuudet olivat toimivia ja sopivat oletettavasti hyvin siihen käyttöön, johon ne oli tarkoitettu: sisällönhallintaan, eritoten back-end- ja front-end-puolien erottamiseen toisistaan. Tähän niitä ei kuitenkaan tarvita maksuvälitysjärjestelmä-prototyypissä. • Drupal 8 coren REST-ominaisuuksia voi käyttää maksuvälitysjärjestelmän API:ssa vain, jos niiden toiminnallisuuksia mukauttaa ja laajentaa omilla custom-moduuleilla. • Tällä hetkellä saatavilla olevista contrib-moduuleista ei maksuvälitysjärjestelmän ydinominaisuuksia toteuttaessa liene apua.

5.2 Iteraatio 2: maksuvälitysjärjestelmän tekninen suunnittelu

Maksuvälitysjärjestelmän teknistä suunnittelua tehtiin jonkin verran kaikissa iteraatioissa ja vähän jo ennen iteraatioiden aloittamista, mutta iteraatiossa 2 keskityttiin ainoastaan tekniseen suunnitteluun (taulukko 11). Edellisen iteraation tulokset auttoivat hahmottamaan prototyypin vaatimaa toteutustapaa riittävästi

ja myöskin kaikilta mobiilioperaattoreilta saatiin API-dokumentaatiot lisätietoineen vasta tämän iteraation alkuun mennessä, joten nyt teknisessä suunnittelussa pystyttiin tekemään projektin jatkon kannalta välttämätön suurempi harppaus. Vaikka operaattoreiden API-vertailu tehtiin iteraation 2 aikana, se sijoitettiin jälkikäteen esitutkimuskappaleeseen opinnäytetyöraportin rakenteen selkiyttämiseksi.

Taulukko 11. Iteraatio 2:n tavoite, siihen valitut käyttäjätarinat ja kuhunkin käyttäjätarinaan liittyvät tehtävät.

Iteraatio 2:n tavoite: Tavoitteena oli hahmottaa ja mallintaa maksuvälitysjärjestelmän tekninen toiminta Esi-Alfa-1-prototyypin laajuudessa tarkkuudella, joka olisi riittävä ohjelmointityön aloittamiseksi.	
Käyttäjätarinat:	**Tehtävät:**
US20 Maksuvälitysjärjestelmänä tarjoan REST API:n kauppiaiden sovelluksille (EPIC)	T20.1 Suunnittele maksuvälitysjärjestelmän REST API:n alustava versio

Ensimmäiseksi pyrittiin hahmottamaan maksutapahtuman kulku maksuvälitysjärjestelmän näkökulmasta tavalla, joka olisi yhteinen kaikkien operaattoreiden liittymäasiakkaiden mobiilimaksuja käsitellessä. Maksuvälitysjärjestelmän tapauksessa tämä tarkoitti sen selvittämistä, mitä pyyntöjä ja vastauksia operaattoreiden mobiilimaksu-API:en kanssa keskusteleminen vaati ja mitä pyyntöjä ja vastauksia kauppiaiden sovellusten tulisi vaihtaa maksuvälitysjärjestelmän API:n kanssa maksutapahtuman toteuttamiseksi. Opinnäytetyön rajauksen mukaisesti tarkasteluun otettiin vain ”yhden askeleen veloitus”-tyypin maksutapahtuma, mikä jokaiseen API-kuvaukseen oli nimetty ja toteutettu hiukan eri tavalla: Soneran API:ssa se oli ”One-Step Charge”, Elisan API:ssa ”Direct billing” ja Tieto Connectionin API:ssa ”Direct debit” (Elisa yritysasiakaspalvelu 2015; Sonera 2016b; Tieto Connection / VAS Center 2014).

Maksuvälitysjärjestelmän toiminnan kuvaamisen artefaktiksi valittiin UML-sekvenssikaavio (kuvio 13 ja taulukko 12), koska tarkoitus oli selvittää käyttöskenaario, jolla järjestelmää potentiaalisesti käytettäisiin. Sekvenssikaavio sopii monimutkaisten käyttötapahtumien logiikan selvittämiseksi, kunhan kaaviota ra-

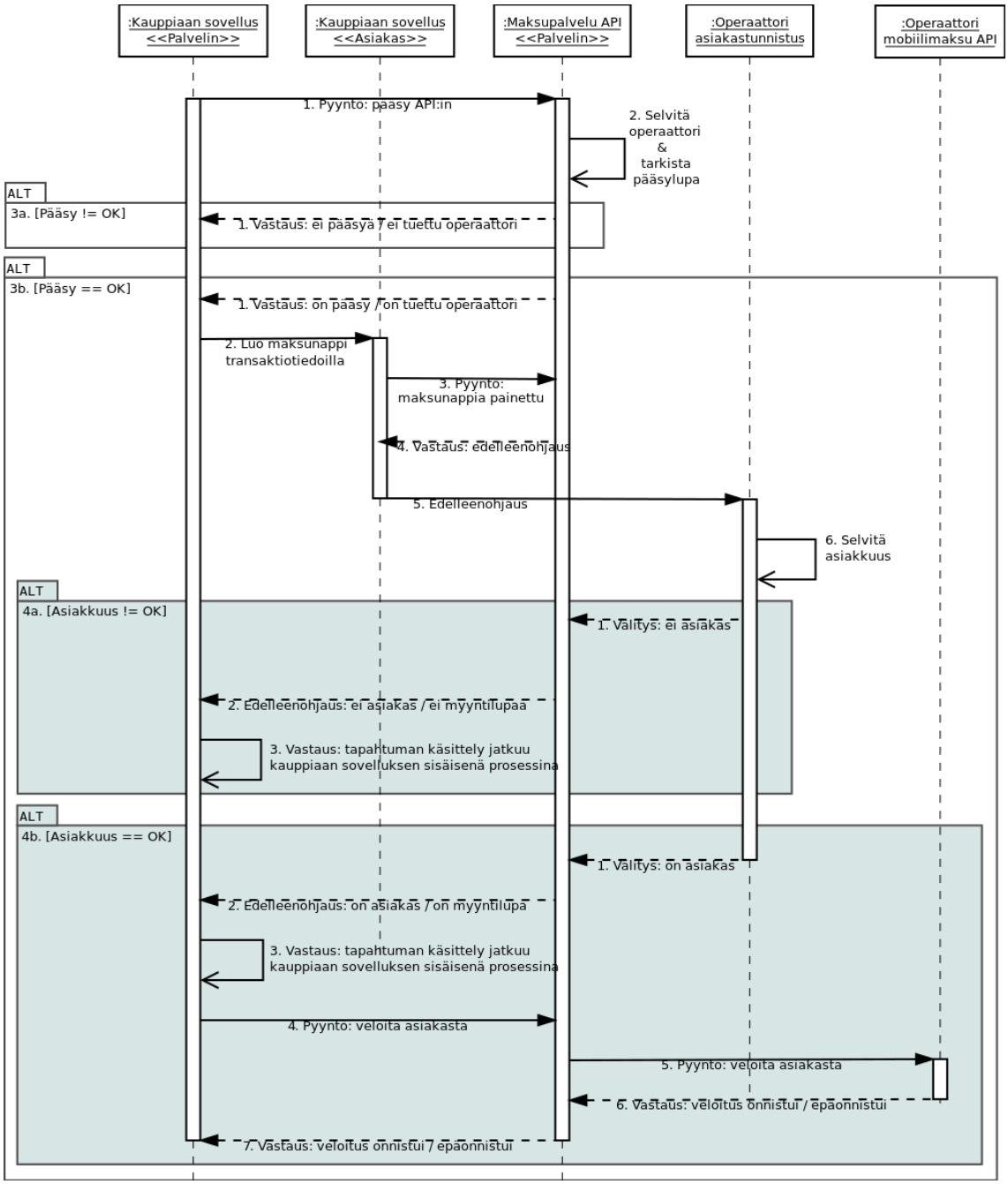
jataan jonkin tietyn osa-alueen mallintamiseen, ettei siitä tule liian laajaa. (Ambler 2002, 349.) Operaattoreiden API-kuvaukset asettivat jo selkeät reunaehdot sille, minkälainen käyttöskenaario voisi olla. Haasteena oli mallintaa maksuvälitysjärjestelmä siten, että kaavion esittämä malli olisi yhteensopiva kaikkien operaattoreiden API:en kanssa.

Haasteelliseksi maksuvälitysjärjestelmän toiminnan hahmottamisessa osoittautui etenkin se, että maksuvälitysjärjestelmän API:a täytyi pystyä käyttämään sekä palvelin- että asiakaspuolen pyyntöjen käsittelemiseen. Asiakaspuolen pyyntöjen käsittelemisen tarve johtui siitä, että ennen kuin maksutapahtuma voidaan toteuttaa, täytyy mobiilioperaattorin tunnistaa liittymäasiakkaansa eli loppukäyttäjä. Kun Mobiilimaksu-nappia painetaan kauppiaan sovelluksessa, lähtee loppukäyttäjältä asiakaspuolen HTTP-pyyntö, joka tulee kierrättää operaattorin asiakasselvitysjärjestelmän kautta. Loppukäyttäjän HTTP-pyynnön käydessä mobiilioperaattorin palvelimella, lisätään pyynnön HTTP-otsikoihin vähintään loppukäyttäjän matkapuhelinnumero, mutta osa operaattoreista lisää pyyntöön myös muita, transaktiokohtaisia tietoja. HTTP-pyyntö täytyy kierrättää maksuvälitysjärjestelmän kautta kahteen kertaan siten, että ensimmäisellä kerralla pyynnöstä tulee selvittää IP-osoitteen perusteella loppukäyttäjän operaattori ennen pyynnön ohjaamista jonkin operaattorin käsiteltäväksi ja toisella kerralla pyynnön palatessa operaattorilta maksuvälitysjärjestelmään on HTTP-otsikoita tarpeen käsitellä maksutapahtuman toteuttamiseksi tarvittavien tietojen tallentamiseksi maksuvälitysjärjestelmään, kauppiaan sovelluksen tarvitsemien tietojen lisäämiseksi pyyntöön ja vain maksuvälitysjärjestelmän tietoon tarkoitettujen otsikoiden poistamiseksi. Lopuksi HTTP-pyyntö tulee ohjata kauppiaan sovelluksen palvelinpuolelle.

Asiakkuuden selvitysjärjestelmä oli eri API-kuvauksissa nimetty ja toteutettu hiukan eri tavalla: Soneran tapauksessa toiminnallisuus ei ollut osa julkista API-kuvausta vaan tätä varten oli erillinen ”MSISDN Enrichment”-dokumentaatio, Elisan API:ssa toiminto tunnettiin nimellä ”MSISDN Resolution” ja Tieto Connectionin API:ssa ”IP Billing Request” (Elisa yritysasiakaspalvelu 2015; Sonera 2016a; Tieto Connection / VAS Center 2014). Elisan dokumentaatio kuvasi toiminnallisuuden selkeimmin ja kattavimmin ja auttoi myös ymmärtämään toisten operaattoreiden teknistä kuvausta. Vain Elisan dokumentaatiosta kävi ilmi, että

operaattorin puolella pyyntö ohjataan edelleen maksuvälitysjärjestelmään välityspalvelinteknologialla, joten sen ei tarvitse välillä käydä loppukäyttäjän selaimessa. (Elisa yritysasiakaspalvelu 2015.) Jos maksuvälitysjärjestelmä ei käytä välityspalvelinteknologiaa, vaihtoehdoksi jää HTTP 302 Found -edelleenohjauksien käyttäminen, mikä edellyttää sitä, että loppukäyttäjän selaimen tai HTTP-pyynnön tekevän kirjaston on pystyttävä seuraamaan niitä.

Vasta kun loppukäyttäjä on selvitetty, maksuvälitysjärjestelmä voi tehdä palvelinpuolen pyynnön maksun veloittamiseksi. ”Yhden askeleen veloitus”-tyyppiä käyttäen tämä on varsin suoraviivaista. Maksuvälitysjärjestelmä välittää kauppiaan sovelluksen palvelinpuolelta tulevat veloituspyynnöt edelleen loppukäyttäjän operaattorin mobiilimaksu-API:iin. Tässä tapauksessa maksuvälitysjärjestelmä toimii kuten ”normaali API” eli kauppiaan sovellus keskustelee palvelinpuolella vain maksuvälitysjärjestelmän kanssa, eivätkä kauppiaan sovelluksen HTTP-pyynnöt käy operaattorin palvelimella asti kääntymässä.



Kuvio 13. ”Yhden askeleen veloitus”-maksutapahtuman kulku kuvattuna UML-sekvenssikaaviona.

Taulukko 12. ”Yhden askeleen veloitus”-maksutapahtumaa esittävän kaavion (kuvio 13) kuvaus. Opinnäytetyön aikana ei voitu kokeilla operaattorin palvelua käyttäen taulukossa vaaleanharmaalla taustalla esitettyjä vaiheita, joten niiden kuvaukset perustuvat olettamuksiin.

Pää-tapahtumat	**Alitapahtumat**	**Toimija**	**Kuvaus**
1. Pyyntö: Pääsy API:iin		Kauppiaan palvelin	Kauppiaan sovellus tunnistautuu maksuvälitysjärjestelmässä ja kertoo samalla asiakaspuolensa eli loppukäyttäjän IP-osoitteen.
2. Selvitä operaattori		Maksuvälitysjärjestelmä	Maksuvälitysjärjestelmä tutkii, onko mobiilimaksutapa saatavana IP-osoitteen perusteella.
3a. Maksutapaa ei tarjota	1. Ei tuettu operaattori	Maksuvälitysjärjestelmä	Maksuvälitysjärjestelmä vastaa kauppiaan sovellukselle, että mobiilimaksutapa ei ole saatavissa tälle loppukäyttäjälle.
3b. Maksutapa tarjotaan	1. Tuettu operaattori	Maksuvälitysjärjestelmä	Maksuvälitysjärjestelmä vastaa kauppiaan sovellukselle, että mobiilimaksutapa on saatavissa tälle loppukäyttäjälle. Kauppiaan sovellukselle luodaan transaktiotunnus, joka on voimassa rajoitetun ajan kauppiaan sovelluksen sekä asiakas- että palvelinpuolen IP-osoitteille.
	2. Luo maksunappi	Kauppiaan palvelin	Kauppiaan sovelluksen palvelinpuoli näyttää sovelluksen asiakaspuolella loppukäyttäjälle Mobiilimaksu-napin.
	3. Pyyntö: maksunappia painettu	Kauppiaan asiakas	Loppukäyttäjä painaa Mobiilimaksu-nappia, joka laukaisee pyynnön maksuvälitysjärjestelmään, minkä mukana on maksuvälitysjärjestelmältä saatu transaktiotunnus.
	4. Vastaus: edelleenohjaus	Maksuvälitysjärjestelmä	Maksuvälitysjärjestelmä ohjaa pyynnön edelleen mobiilioperaattorin asiakasselvityspalveluun asettamalla HTTP-otsikon 302 Found.
	5. Pyyntö: edelleenohjaus	Kauppiaan asiakas	Loppukäyttäjän pyyntö tehdään operaattorin asiakaselvityspalveluun.
	6. Selvitä asiakkuus	Mobiilioperaattori	Mobiilioperaattori selvittää, onko loppukäyttäjä liittymäasiakas. Osa operaattoreista saattaa jo tässä vaiheessa selvittää myös myyntiluvan, osa saattaa selvittää asian vasta myöhemmin.
4a. Myyntilupaa ei ole	1. Pyyntö: välitys	Mobiilioperaattori	Mobiilioperaattori muokkaa HTTP-otsikoita ja välittää pyynnön edelleen maksuvälitysjärjestelmään: loppukäyttäjää ei tunnistettu ja/tai myyntilupaa ei ole.
	2. Pyyntö: edelleenohjaus	Maksuvälitysjärjestelmä	Maksuvälitysjärjestelmä muokkaa HTTP-otsikoita ja edelleenohjaa pyynnön kauppiaan sovelluksen palvelinpuolelle asettamalla HTTP-otsikon 302 Found ja välittämällä tiedon, että loppukäyttäjää ei tunnistettu ja/tai myyntilupaa ei ole. Transaktiotunnuksen voimassaolo päättyy.
	3. Vastaus	Kauppiaan palvelin	Kauppiaan sovelluksen palvelinpuoli generoi vastauksen asiakaspuolelleen oman logiikkansa mukaisesti.
4b. Myyntilu-	1. Pyyntö:	Mobiili-	Mobiilioperaattori muokkaa HTTP-otsikoita ja vä-

pa on	välitys	operaattori	littää pyynnön edelleen maksuvälitysjärjestelmään: loppukäyttäjä tunnistettiin liittymäasiakkaaksi ja samalla osa operaattoreista myöntää myös myyntiluvan.
	2. Pyyntö: edelleenohjaus	Maksuvälitys-järjestelmä	Maksuvälitysjärjestelmä tallentaa ja muokkaa HTTP-otsikoita ja ohjaa pyynnön edelleen kauppiaan sovelluksen palvelinpuolelle asettamalla HTTP-otsikon 302 Found ja välittämällä tiedon, että loppukäyttäjä tunnistettiin.
	3. Vastaus	Kauppiaan palvelin	Kauppiaan sovelluksen palvelinpuoli generoi vastauksen asiakaspuolelleen oman logiikkansa mukaisesti.
	4. Pyyntö: veloitus	Kauppiaan palvelin	Kauppiaan sovellus pyytää maksuvälitysjärjestelmää veloittamaan maksun tunnistetulta loppukäyttäjältä.
	5. Pyyntö: veloitus	Maksuvälitys-järjestelmä	Maksuvälitysjärjestelmä pyytää mobiilioperaattoria veloittamaan maksun tunnistetulta loppukäyttäjältä.
	6. Vastaus	Mobiili-operaattori	Operaattori vastaa maksuvälitysjärjestelmälle veloituksen onnistumisesta tai epäonnistumisesta.
	7. Vastaus	Maksuvälitys-järjestelmä	Maksuvälitysjärjestelmä vastaa kauppiaan sovellukselle veloituksen onnistumisesta tai epäonnistumisesta.

Maksuvälitysjärjestelmän tuli tarjota myös oma API kauppiaiden sovelluksille. Toimeksiantajan kanssa oli sovittu, että API:sta tehtäisiin REST-muotoinen, mutta tehtäväksi ennen API:n toteuttamista jäi selvittää, minkälainen on hyvä API ja miten se suunnitellaan. Googlen johtavan ohjelmistoinsinöörin Joshua Blochin (2007) mukaan hyvän API:n tuntomerkkejä ovat:

- Helppo oppia
- Helppo käyttää, jopa ilman dokumentaatiota
- Vaikea käyttää väärin
- Riittävän tehokas tyydyttämään vaatimukset
- Helppo laajentaa
- Käyttäjien kannalta tarkoituksenmukainen
- API:a käyttävää koodia on helppoa lukea ja ylläpitää.

API:eja on karkeasti ottaen kahta eri tyyppiä: yksityisiä ja julkisia. Yksityiset API:t ovat vain yrityksen omassa ja sopimuksen nojalla yrityksen asiakkaiden ja yhteistyökumppaneiden käytössä, kun taas julkisen API:n käyttäjäksi on periaatteessa pääsy kenellä tahansa. (Brail, Jacobson & Woods 2012, 7.) Maksuvälitysjärjestelmän API sijoittuu yksityisten API:en kategoriaan. API:n suunnittelussa on syytä huomioida myös API:n kohderyhmä, joita on karkeasti ottaen kahta eri tyyppiä: kehittäjät ja loppukäyttäjät (Brail ym. 2012, 53–56). Sekaannuksen välttämiseksi on tarkennettava, että Brail ym. tarkoittavat loppukäyttäjillä käyttäjäryhmää, jota tässä opinnäytetyössä kutsutaan kauppiaiksi. Maksuvälitysjärjestelmän API on tarkoitettu sekä kehittäjille että kauppiaille, joten API:n suunnittelussa on syytä huomioida sen helppokäyttöisyys ja myöhemmässä vaiheessa esim. laatia eri dokumentaatio kummallekin kohderyhmälle. Kehittäjät ovat niitä, jotka ohjelmoivat maksuvälitysjärjestelmän API-integraation esimerkiksi verkkokauppajärjestelmään, kun taas kauppiaat ovat esimerkiksi verkkokauppajärjestelmää käyttäviä verkkokauppiaita, jotka tekevät sopimuksen maksuvälitysjärjestelmän kanssa API:n käytöstä ja joille maksuvälitysjärjestelmän on tilitettävä myynnit ja joiden on pystyttävä maksuvälitysjärjestelmää käyttäen selvittämään esimerkiksi virhemaksuja ja tekemään hyvityksiä. Kauppiaiden tarvitsemien ominaisuuksien kehittäminen ei kuitenkaan kuulu Esi-Alfa-1-prototyypin laajuuteen, joten tässä vaiheessa API:n suunnittelussa huomioidaan lähinnä vain kehittäjät.

Kuten esimerkiksi palomuurit ja kuormantasaajat, myös API-yhdyskäytävät (engl. API Gateway) – jollainen maksuvälitysjärjestelmäkin on – lisäävät tietoverkkoliikenteeseen yhden uuden abstraktiotason, jonka tarkoitus on tuoda jokin parannus tai uusia ominaisuuksia helpommin käyttäjien saataville. Yleensä API-yhdyskäytävät mm. yksinkertaistavat jonkin monimutkaisen käyttötapauksen, huolehtivat suorituskyvyn parantamiseksi välimuistista, parantavat ratkaisujen tietoturvaa ja huolehtivat lokitietojen keräämisestä sekä muuntavat dataa eri muotoihin. Yhdyskäytävät pyrkivät vähentämään yhdyskäytävää käyttävien järjestelmien kuormitusta ja kompleksisuutta sekä järjestelmien kehittäjien työmäärää. (Brail ym. 2012, 106–108.) Esi-Alfa-1-prototyypin laajuudessa edellä mainituista pyrittiin toteuttamaan vain Mobiilimaksu-maksutavan käyttöönoton yksinkertaistaminen.

Brail ym. (2012, 54) suosittelevat teknologiavalinnaksi modernille API:lle rakenteeksi pragmaattista REST:iä (engl. Pragmatic REST), sillä se on helposti opittavissa, käytettävissä ja laajennettavissa; dataformaatiksi JSON:ia sekä pyynnöissä että vastauksissa, sillä kehittäjien on helppoa tuottaa ja käyttää sitä; ja tietoturvaratkaisuksi OAuth-autentikaatiota, sillä se tarjoaa monipuoliset käyttömahdollisuudet, eikä salasanoja ole tarpeen levittää laajasti internetissä. Suositukseen ja sen perusteisiin oli helppo yhtyä ja se vastasi myös toimeksiantajan kantaa, joten muita vaihtoehtoja kuten SOAP ja XML ei ryhdytty tutkimaan. Esi-Alfa-1-prototyypin tapauksessa REST-API JSON-dataformaatilla voitiin toteuttaa, mutta OAuthin sijaan käytetään Basic Authia, koska opinnäytetyön laajuuden vuoksi tietoturvaominaisuuksien kehittäminen sovellukseen oli rajattu työn ulkopuolelle.

REST on lyhenne sanoista engl. ”Representational State Transfer”, jolla tarkoitetaan HTTP-protokollaa käyttävää arkkitehtuurityyliä asiakas-palvelin-järjestelmien väliseen viestintään internetissä yhtenäistä rajapintaa käyttäen, joka identifioi käytettävät tilattomat resurssit URI-polun perusteella (Fielding 2000). Yksi REST:n ideoista on valjastaa HTTP-metodit käsittelemään REST-resursseja CRUD-operaatioin eräässä mielessä hiukan samoin kuin kyseessä olisi tietokanta. HTTP-metodit POST, GET, PUT ja DELETE vastaavat REST-käytössä operaatioita CREATE, READ, UPDATE ja DELETE. (Brail ym. 2012, 60.) REST:n soveltamisessa käytäntöön API-suunnittelussa on useita tapoja, joista suosituin lienee pragmaattinen lähestymistapa, mutta suosittu on myös HATEOAS (engl. Hypermedia as the Engine of Application State), minkä toteuttamiseen Drupal-ytimen mukana tulee HAL-moduuli (Brail ym. 2012, 61; Drupal.org 2017ä). Pragmaattisen lähestymistavan periaatteisiin kuuluu mm. että URI-malli on pidettävä helppona käyttää, keksiä ja laajentaa; URI-polkuun sijoitettavat parametrit on oltava standardeja ja helposti arvattavia; käytettävä dataformaatti on määriteltävä selkeästi; HTTP-vastauskoodeja on käytettävä ja huolehdittava että koodi kuvaa sitä vastaavaa tilaa; ja kaikki muu kuten tietoturvaan ja reititykseen liittyvä informaatio tulisi piilottaa URI-polusta HTTP-otsikoihin. (Brail ym. 2012, 62-64.)

Maksuvälitysjärjestelmän API-kuvaus laadittiin Esi-Alfa-1-prototyypin laajuudessa (taulukot 13–17). Esimerkiksi READ-operaatiot maksutapahtumien tai trans-

aktiotunnusten noutamiseksi eivät kuuluneet prototyypin vaatimuksiin. Suunnittelussa pyrittiin käyttämään pragmaattista lähestymistapaa, vaikka asiakaspuolen API:n osalta tässä ei täysin voitukaan onnistua. Erikoista oli, että pyyntöjä maksujärjestelmän API:iin täytyi tehdä niin kauppiaan sovelluksen palvelin- ja asiakaspuolen kuin operaattorinkin ja että kukin polku käytti joko HTTP tai HTTPS protokollaa ja oli varattu vain tietyn tahon käyttöön. Käyttöskenaariosta tuli sen verran mutkikas, että vaikka maksuvälitysjärjestelmän API pyrittäisiin dokumentoimaan hyvin ja tekemään se mahdollisimman helpoksi käyttää, API:n integroiminen saattaa olla kaikesta huolimatta haasteellista osalle kehittäjistä. Maksuvälitysjärjestelmässä ei myöskään ole mahdollista päästä puhtaaseen REST-toteutukseen, jossa resurssit ovat tilattomia, koska maksutapahtuman tilan täytyy voida muuttua ja tilan muuttumista on välttämätöntä seurata palvelinpuolella ja seuraaminen on suositeltavaa myös asiakaspuolella (Tutorialspoint 2017d). Esimerkiksi jos asiakasta ei ole tunnistettu tai jos transaktiotunnus on jo käytetty, resurssin tila on erilainen kuin silloin, jos asiakas on tunnistettu ja transaktiotunnus on käyttämätön. Iteraation 2 tulokset on arvioitu taulukossa 18.

Taulukko 13. API-kuvaus pyynnöstä transaktiotunnuksen luomiseksi kauppiaan sovellukselle. Vastaa taulukon 12 päätapahtumaa 1.

Transaktiotunnuksen luominen kauppiaan sovellukselle	
Kuvaus	Ennen kuin kauppiaan sovelluksen palvelinpuoli luo Mobiilimaksu-napin asiakaspuolelle, tarvitaan kauppiaan sovelluksen sekä asiakas- että palvelinpuolen käyttöön maksutapahtumakohtainen tunnus. Tunnusta pyydettäessä varmistetaan alustavasti, että maksutapa voidaan tarjota loppukäyttäjälle. Transaktiotunnus on voimassa vain rajallisen ajan ja sitä voidaan käyttää vain yhden maksun tekemiseen.
Pyynnön tekijä	Kauppiaan sovelluksen palvelinpuoli
Käyttölupa	• Kauppiaan sovelluksen Drupal-käyttäjätunnus • Kauppiaan sovelluksen palvelimen IP-osoite
Edellytykset	• Kauppiaan sovelluksen palvelinpuolella on Drupalin käyttäjätunnuksen ja salasanan pohjalta luotu Basic Auth token. • Kauppiaan sovelluksen palvelimen kiinteä IP on määritelty maksuvälitysjärjestelmän API:iin. • Kauppiaan sovellus on määritelty operaattoreiden päässä.
Polku	api/v0/transaction

HTTP protokolla	HTTPS			
HTTP portti	3443			
HTTP metodi	POST			
Sisältötyyppi	application/json			
Auktorisointi	HTTP Basic Authentication			
Pyyntö (JSON)	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	clientIp	string (255)	Loppukäyttäjän IP-osoite.	ei
	appId	string (255)	Sovellus, johon maksu liittyy.	ei
Vastaus (JSON), onnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	clientIp	string (255)	Loppukäyttäjän IP-osoite.	ei
	appId	string (255)	Sovellus, johon maksu liittyy.	ei
	transactionId	string(255)	Transaktiotunnus	ei
	TransactionId Expire	unix timestamp	Aika, jolloin tunniste vanhenee.	ei
	HTTP vastauskoodi: 201 Created			
Vastaus (JSON), epäonnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	clientIp	string (255)	Loppukäyttäjän IP-osoite.	kyllä
	appId	string (255)	Sovellus, johon maksu liittyy.	kyllä
	error	string (255)	Virhe	ei
	errorMessage	string (2048)	Virheviesti	kyllä
	HTTP vastauskoodi tilanteesta riippuen: 400 Bad Request, 401 Unauthorized tai 403 Forbidden.			

Taulukko 14. API-kuvaus pyynnöstä asiakasselvityksen käynnistämiseksi loppukäyttäjälle. Vastaa taulukon 12 päätapahtuman 3b alitapahtumia 3-5.

Loppukäyttäjälle tehtävän operaattorin asiakasselvityksen käynnistäminen				
Kuvaus	Loppukäyttäjän asiakasselvitys käynnistyy maksuvälitysjärjestelmän API:a kutsumalla, joka edelleenohjaa loppukäyttäjän HTTP-pyynnön operaattorin API:iin.			
Pyynnön tekijä	Kauppiaan sovelluksen asiakaspuoli			
Käyttölupa	• Voimassaoleva transaktiotunnus • Loppukäyttäjän IP-osoite			
Edellytykset	• Kauppiaan sovelluksen palvelinpuoli on noutanut transaktiotunnuksen. • Loppukäyttäjän IP-osoite on sama kuin kauppiaan sovelluksen palvelinpuolen ilmoittama IP-osoite transaktiotunnusta noudettaessa. • Maksupalveluun on tallennettu kauppiaan sovelluksen palvelinpuolen URL, jonne pyyntö lopulta reitittyy. • Asiakaspuolen saadessa maksuvälitysjärjestelmältä 302 Found -vastauksen, asiakaspuolen on siirryttävä annettuun operaattorin tai kauppiaan palvelimen osoitteeseen.			
Polku	api/v0/transaction/{transactionid}/resolve			
HTTP protokolla	HTTP			
HTTP portti	80			
HTTP metodi	GET			
Sisältötyyppi	ei			
Auktorisointi	ei			
Pyyntö (GET-parametri)	ei			
302 Found -vastaus (GET-parametrit), onnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	transactionId	string(255)	Transaktiotunnus	ei
	HTTP 302 Location: operaattorin asiakasselvitys			
302 Found -vastaus (GET-parametrit), epäonnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	transactionId	string(255)	Transaktiotunnus	ei
	resolved	string(3)	Arvo: ”NOK”	ei
	error	string (255)	Virhe	ei
	errorMessage	string (2048)	Virheviesti	kyllä

	HTTP 302 Location: kauppiaan sovelluksen palvelinpuoli			
Vastaus	Vastauksen pyyntöön antaa kauppiaan sovelluksen palvelinpuoli.			

Taulukko 15. API-kuvaus operaattorin asiakasselvitykseen käytetyltä välityspalvelimelta tulevan pyynnön käsittelystä. Vastaa taulukon 12 päätapahtuman 4a alitapahtumia 1-3 ja päätapahtuman 4b alitapahtumia 1-3.

Operaattori palauttaa asiakasselvityksen tulokset				
Kuvaus	Pyynnön saapuessa operaattorin lisäämät HTTP-otsikot poistetaan ja pyyntöön lisätään tieto asiakasselvityksen tuloksesta. Seuraavaksi pyyntö ohjataan edelleen kauppiaan palvelimen palvelinpuolelle. Tätä kuvausta ei anneta maksuvälitysjärjestelmän käyttäjille, vaan maksuvälitysjärjestelmän ylläpito sopii tästä operaattorin kanssa.			
Pyynnön tekijä	Operaattori			
Käyttölupa	• Operaattorin ilmoittama IP tai IP-avaruus			
Edellytykset	• Operaattorin kanssa on sovittu asiakasselvitysten tulosten toimittamisesta. • Kauppiaan sovelluksen palvelinpuoli on noutanut transaktiotunnuksen. • Loppukäyttäjän IP-osoite on sama kuin kauppiaan sovelluksen palvelinpuolen ilmoittama IP-osoite transaktiotunnusta noudettaessa. • Maksupalveluun on tallennettu kauppiaan sovelluksen palvelinpuolen URL, jonne pyyntö lopulta reitittyy. • Asiakaspuolen saadessa maksuvälitysjärjestelmältä 302 Found -vastauksen, asiakaspuolen on siirryttävä annettuun kauppiaan palvelinpuolen osoitteeseen.			
Polku	service/pay/{operatorName}			
HTTP protokolla	HTTP			
HTTP portti	80			
HTTP metodi	GET			
Sisältötyyppi	ei			
Auktorisointi	ei			
Välitetty pyyntö (GET-parametri)	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	transactionId	string(255)	Transaktiotunnus	ei
	Jos operaattori ei välitä edelleen alkuperäisen pyynnön GET-para-			

	metriä, vähintään loppukäyttäjän IP-osoitteen tulee selvitä pyynnöstä.			
Välitetty pyyntö (HTTP-otsikko)	Eri operaattorit välittävät eri tietoja HTTP-otsikoissa. Jokainen operaattori kuitenkin ilmoittaa vähintään loppukäyttäjän matkapuhelinnumeron. Jokaisen operaattorin virhevastaus on myöskin yksilöllinen.			
302 Found -vastaus (GET-parametrit), onnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	transactionId	string(255)	Transaktiotunnus	ei
	resolved	string(3)	Arvo: ”OK”	ei
	HTTP 302 Location: kauppiaan sovelluksen palvelinpuoli			
302 Found -vastaus (GET-parametrit), epäonnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	transactionId	string(255)	Transaktiotunnus	ei
	resolved	string(3)	Arvo: ”NOK”	ei
	error	string (255)	Virhe	ei
	errorMessage	string (2048)	Virheviesti	kyllä
	HTTP 302 Location: kauppiaan sovelluksen palvelinpuoli			
Vastaus	Vastauksen pyyntöön antaa kauppiaan sovelluksen palvelinpuoli.			

Taulukko 16. API-kuvauksen ohje pyynnöstä ”yhden askeleen veloitus”-maksutapahtuman luomiseksi. Vastaa taulukon 12 päätapahtuman 4b alitapahtumia 4 ja 7.

Transaktiotunnuksen luominen kauppiaan sovellukselle	
Kuvaus	Kun loppukäyttäjä on tunnistettu, kauppiaan sovelluksen palvelinpuoli voi käyttää maksuvälitysjärjestelmää loppukäyttäjän veloittamiseen.
Pyynnön tekijä	Kauppiaan sovelluksen palvelinpuoli
Käyttölupa	• Kauppiaan sovelluksen Drupal-käyttäjätunnus • Kauppiaan sovelluksen palvelimen IP-osoite
Edellytykset	• Kauppiaan sovelluksen palvelinpuolella on Drupalin käyttäjätunnuksen ja salasanan pohjalta luotu Basic Auth token. • Kauppiaan sovelluksen palvelimen kiinteä IP on määritelty maksuvälitysjärjestelmän API:iin. • Kauppiaan sovellus on määritelty operaattoreiden päässä. • Loppukäyttäjä on operaattorin tunnistama. • Loppukäyttäjälle on operaattorin antama myyntilupa (ei tässä vaiheessa päde kaikkiin operaattoreihin).

	• Voimassaoleva transaktiotunnus.			
Polku	api/v0/transaction/{transactionId}/charge			
HTTP protokolla	HTTPS			
HTTP portti	3443			
HTTP metodi	POST			
Sisältötyyppi	application/json			
Auktorisointi	HTTP Basic Authentication			
Pyyntö (JSON)	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	Payment Amount	decimal(2)	Pyydetty maksun määrä	ei
	Payment Currency	string(3)	Maksun valuutta	ei
	Payment Description	string(255)	Kauppiaan kuvaus maksutapahtumalle (näkyy puhelinlaskulla)	ei
	Payment Reference Code	string(255)	Kauppiaan järjestelmän viitekoodi	kyllä
Vastaus (JSON), onnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	Payment Amount	decimal(2)	Pyydetty maksun määrä	ei
	Payment Amount Charged	decimal(2)	Veloitettu maksun määrä	ei
	Payment Currency	string(3)	Maksun valuutta	ei
	Payment Description	string(255)	Kauppiaan kuvaus maksutapahtumalle (näkyy puhelinlaskulla)	ei
	Payment Reference Code	string(255)	Kauppiaan järjestelmän viitekoodi	kyllä
	clientTel	string(255)	Loppuasiakkaan matkapu-	ei

			helinnumero	
	Payment Gateway	string(255)	Käytetty gateway, esim. ”Sonera”	ei
	Payment State	string(255)	Maksutapahtuman tila: ”Charged”	ei
	HTTP vastauskoodi: 201 Created			
Vastaus (JSON), epäonnistunut	**Parametri**	**Tietotyyppi**	**Kuvaus**	**Valinnainen**
	Payment Amount	decimal(2)	Pyydetty maksun määrä	ei
	Payment Amount Charged	decimal(2)	Veloitettu maksun määrä	ei
	Payment Currency	string(3)	Maksun valuutta	ei
	Payment Description	string(255)	Kauppiaan kuvaus maksutapahtumalle (näkyy puhelinlaskulla)	ei
	Payment Reference Code	string(255)	Kauppiaan järjestelmän viitekoodi	kyllä
	clientTel	string(255)	Loppuasiakkaan matkapuhelinnumero	ei
	Payment Gateway	string(255)	Käytetty gateway, esim. ”Sonera”	ei
	paymentState	string(255)	Maksutapahtuman tila: ”Error”	ei
	error	string (255)	Virhe	ei
	errorMessage	string (2048)	Virheviesti	kyllä
	HTTP vastauskoodi tilanteesta riippuen: 400 Bad Request, 401 Unauthorized tai 403 Forbidden.			

Taulukko 17. API:n omat error-avaimen arvoksi laitettavat virhekoodit, jotka täydentävät HTTP-tilakoodin antamaa informaatiota. Virhekoodin lisäksi errorMessage-avaimeen saatetaan sijoittaa tarkempikin virhekuvaus, mutta tämä poistettaneen käytöstä tuotantoversiossa.

HTTP-tilakoodi	**Oma API-virhekoodi**	**Kuvaus**
400 Bad Request	4001 Bad request	Pyyntö on virheellinen tai puutteellinen.
401 Unauthorized	4012 Authentication with operator failed	Kauppiaan sovelluksen tunnistaminen operaattorin päässä ei onnistunut.
	4013 Expired transaction id	Transaktiotunnus ei ole voimassa.
	4014 Bad app id	Kauppiaan sovelluksen tunnus on väärin tai sovellus on epäaktiivinen.
	4015 Bad client IP	Loppukäyttäjän IP ei kelpaa tai vastaa aiemmin ilmoitettua.
	4016 Bad transaction id	Transaktiotunnus on virheellinen tai virhetilassa.
	4018 Unresolved client	Transaktiotunnusta ei voi käyttää, koska asiakasta ei ole tunnistettu.
403 Forbidden	4030 Unsupported mobile operator	Loppukäyttäjän IP ei mahdollista Mobiilimaksua.
	4031 Unchargeable payment	Loppukäyttäjän veloittaminen ei onnistunut.
	4032 Refused service category	Veloitus kuului palvelukategoriaan, johon loppukäyttäjällä on voimassa palveluesto.
	4033 System error	Järjestelmävirhe

Taulukko 18. Itsearvio iteraatio 2:n tuloksista.

Yhteenveto iteraatio 2:n tuloksista	
Ajankäyttö:	• Toteutunut ajankäyttö: 47.5 h
Vahvuudet:	• Maksuvälitysjärjestelmän toiminta saatiin mallinnettua. • Maksuvälitysjärjestelmän API saatiin luonnosteltua. • Maksuvälitysjärjestelmän API:n teknologiavalinnat tehtiin perustellusti.
Heikkoudet:	• Käsitys mobiilimaksun asiakasselvityksen toimintaperiaatteista on lähempänä hypoteesia kuin varmaa tietoa. • Eri operaattoreiden API:t poikkeavat toisistaan ja etenkin asiakasselvitysominaisuuden dokumentointi voisi olla kehittäjien kannalta selkeämpää.
Johtopäätökset:	• Tekninen dokumentaatio saatiin nyt riittävälle tasolle, jotta toteutuksen ohjelmointi voitiin aloittaa seuraavissa iteraatioissa. • Mobiilimaksun toteutustapa on suhteellisen yksinkertainen, mutta sen toimintatavan hahmottaminen vaatii hyvää HTTP-protokollan ja monien eri teknologioiden tuntemusta.

5.3 Iteraatio 3: sovellusten autentikointi maksuvälitysjärjestelmän rajapinnassa

Iteraatiossa 3 tavoitteeksi otettiin toteuttaa kauppiaan sovellusten autentikointi maksuvälitysjärjestelmän rajapinnassa. Jotta tähän päästiin, täytyi iteraatiossa toteuttaa sovellusten tarvitsemat tietorakenteet, REST-rajapinta ja autentikointitoiminnallisuus. Iteraatioissa 1 ja 2 tehtyjen johtopäätösten pohjalta, toteutus edellytti oman custom-moduulin ohjelmointia.

Toteutus päätettiin aloittaa sovellusten autentikoinnista, koska se on toiminnallisuus, jolla maksutapahtuma maksuvälitysjärjestelmässä alkaa (taulukko 13). Tällöin kauppiaan sovellus tunnistautuu maksuvälitysjärjestelmän API:n kautta ja saa yksilöllisen transaktio-id:n. Tätä osatoiminnallisuutta kuvaavat käyttäjätarinat US4 ja US13 otettiin tehtäväksi tässä iteraatiossa (taulukko 19). Moduulin toteutuksessa pyrittiin käyttämään mahdollisimman paljon Drupal Console -komentorivityökalun valmisohjelmakoodia generoivia ominaisuuksia.

Taulukko 19. Iteraatio 3:n tavoite, siihen valitut käyttäjätarinat ja kuhunkin käyttäjätarinaan liittyvät tehtävät.

Iteraatio 3:n tavoite: Mobile Payment API -custom-moduulin toteutuksen aloittaminen: kauppiaan sovellus käynnistää maksutapahtuman maksuvälitysjärjestelmän API:n kautta. Onnistunut tapahtuma palauttaa sovellukselle transaktiotunnuksen, jota myös kauppiaan sovelluksen asiakaspuoli voi käyttää.	
Käyttäjätarinat:	**Tehtävät:**
US4 Maksuvälitysjärjestelmänä vaadin kauppiaan sovellukselta autentikoinnin REST API:n käyttöön (FR)	T4.4 Luo api/v0/transaction https:3443 rest-käsittelypiste T4.5 Generoi transaktiotunnus, joka on tietyn aikaa voimassa tietylle IP:lle T4.7 Basic Auth käyttö api/v0/transaction https:3443 käsittelypisteessä T4.9 Validoi kauppias-palvelin IP:t T4.10 Validoi App ID:t (id olemassa, aktiivinen ja kuuluu käyttäjälle)
US13 Ylläpitäjänä voin luoda kauppiaalle maksuvälitysjärjestelmään sovelluksen (FR)	T13.1 Sisältötyyppi kauppiaan sovellukselle T13.2 Kauppiaan sovelluksen kytkentä käyttäjätunnukseen, johon basic auth käyttö perustuu

Uuden REST-resurssin luonti (tehtävä T4.4) edellytti ensin moduulin luomista. Mobile Payment API -moduuli luotiin suorittamalla Drupal-asennuksen juuressa komento `drupal generate:module`, jonka esittämiin kysymyksiin vastaamalla työkalu generoi polkuun `modules/custom/mobile_payment_api` moduulin tiedostot ja hakemistot, jotka mainitaan taulukossa 20. Consolen komennot rakentavat automaattisesti myös välimuistin uudestaan, jotta Drupal 8 noteeraa tehdyt tiedostomuutokset – tässä tapauksessa uuden moduulin. Lisäksi Drupal 8 konfiguroitiin ns. kehitystilaan (Drupal.org 2017ö), jossa välimuisti oli pois päältä. Myös Chromium-selainta käytettiin ilman välimuistia developer-sivupaneeli avoinna. Käytännössä edellä mainitut toimetkaan eivät riittäneet, vaan kehityksen eri vaiheissa välimuisti piti toisinaan rakentaa uudestaan manuaalisesti, mikä onnistui Consolen komennolla `drupal cache:rebuild all`.

Drupal 8:ssa core-moduulit on sijoitettu core-hakemistoon ja ydintä laajentaville moduuleille on varattu modules-hakemisto. Custom-, contrib- ja features-hakemistonimien käyttö modules-hakemistossa on hyvä tapa, sillä näihin sijoitettavien moduulien alkuperä ja päivityskäytäntö poikkeavat toisistaan. (Drupal.org 2017aa.)

Taulukko 20. Drupal Consolen komennoilla generoidun custom-moduulin Mobile Payment API sisäinen tiedostorakenne.

Polku moduulin juureen	**Tiedosto**	**Käyttötarkoitus**
	`mobile_payment_api.module`	Moduulin ohjelmakoodi
	`mobile_payment_api.info.yml`	Moduulin konfiguraatio
	`mobile_payment_api.services.yml`	Moduulin palveluiden konfiguraatio
`src`	`MerchantAppService.php`	MerchantAppService-palvelu
`src/Tests`	`LoadTest.php`	Drupal Consolen generoimia yksikkötestejä
`src/Plugin/rest/resource`	`MobileTransactionsRestResource.php`	Plugin REST-resurssille

REST-resurssit ovat Drupal 8:ssa plugineja. Tehtävä jatkui generoimalla moduuliin uusi REST-resurssi-plugin nimellä MobileTransactionsRestResource käyttäen komentoa `drupal generate:plugin:rest:resource`. API-suunnitelman (taulukot 13–17) mukaisesti resurssi oli POST-tyyppinen ja sijoitettiin polkuun `api/v0/transaction`. Resurssi otettiin käyttöön komennolla `drupal rest:enable` ja sille sallittiin POST-tila, jonka käytössä oleva dataformaatti oli JSON ja autentikaation tarjoaja Basic Auth. Generoitu tiedosto mainitaan taulukossa 20. REST-resurssi on annotaatiopohjainen plugin (Drupal.org 2017ab). Drupal Console 1.0.0-rc6:n dummy-valmisohjelmakoodi ei toiminut sellaisenaan, vaan REST-resurssin annotaatioon `uri_paths`-avaimen alitason karttaan tarvittiin avain-arvopari `"https://www.drupal.org/link-relations/create" = "/api/v0/transaction"`, jotta REST-resurssin custom-polku toimisi (kuvio 14).

```
/**
 * Provides a resource to get view modes by entity and bundle.
 *
 * @RestResource(
 *   id = "mobile_transactions_rest_resource",
 *   label = @Translation("Mobile transactions rest resource"),
 *   uri_paths = {
 *     "canonical" = "/api/v0/transaction",
 *     "https://www.drupal.org/link-relations/create" =
"/api/v0/transaction"
 *   }
 * )
 */
```

Kuvio 14. Esimerkki annotaatiosta REST-resurssi-pluginia luotaessa. Annotaatio generoi reitin dynaamisesti, joten reittiä ei tarvita moduulin `*.routing.yml`-tiedostoon (Garcia 2014).

Jotta moduuliin oli mahdollista ohjelmoida kauppiaan sovelluksen autentikointitoiminnallisuus, täytyi kauppiaan sovelluksesta olla tietyt tiedot palvelussa. Näin ollen sovelluksia varten luotiin Drupal-ylläpitoliittymässä uusi node-sisältötyyppi

(tehtävä T13.1), jonka rakenne on kuvattu taulukossa 21. Sovelluksen yksilöivä tunnus App Id on sama kuin node id, joten se tiedetään vasta kun sovelluksen tiedot on ensimmäisen kerran tallennettu Drupaliin. Niinpä siitä ei voitu tehdä omaa kenttäänsä. Lopuksi järjestelmään luotiin vielä testikäyttäjätili, jolle tehtiin kauppiasrooli käyttöoikeuksineen (tehtävä T13.2). Vain sovellusten käyttöön tarkoitettuja sovellusroolia käyttäviä sovelluskäyttäjätilejä ei tehdä tähän prototyyppiin, eikä tästä ole vielä olemassa käyttäjätarinaakaan.

Taulukko 21. App-sisältötyyppi kauppiaan sovelluksen määrittelemiseksi maksuvälitysjärjestelmässä.

Kentän nimi	**Kentän tyyppi**	**Määrittely**
App Name	Title	Sovelluksen nimi. Node-entiteettiä käytettäessä title-kenttä on pakollinen.
App Active	Boolean	Onko sovellus käytössä vai ei. Vaadittu kenttä. Vaihtoehdot 0|Off ja 1|On. Oletusarvo 1.
App Customer Service TEL	Telephone number	Sovelluksen asiakaspalvelun matkapuhelinnumero.
App Customer Service URL	Link	Sovelluksen asiakaspalvelun www-osoite. Vain ulkoisia linkkejä. Linkkiteksti ei käytössä.
App Description	Text (plain)	Palvelun kuvaus.
App Server IP	Text (plain)	Sovelluksen IP-osoite. Vaadittu kenttä. HUOM: kentän tyyppi vaihdettava tyyppiin, joka validoi IP-osoitteet.
App Server Endpoint	Link	Sovelluksen www-osoite, jonne edelleenohjataan Mobiilimaksu-napin painaminen. Vaadittu kenttä.
App URL	Link	Sovelluksen asiakaspalvelun www-osoite. Vain ulkoisia linkkejä. Linkkiteksti ei käytössä.
App User	Entity reference	Käyttäjätili, jolla sovelluksen käyttöoikeus autentikoidaan.
App Service Group	List (text)	Sovelluksen palveluryhmä. Vaadittu kenttä. Viestintäviraston määräyksen 35 R/2016 M mukainen määrittely. 1|General services 2|Consulting and ordering services 3|Entertainment services 4|Adult services Oletusarvo 1.

Kauppiaan sovellukseen liittyviä eri toimintoja varten luotiin uusi palvelu `mobile_payment_api.merchantapp`. Tämä onnistui kätevästi Drupal Consolen komennolla `drupal generate:service`. Generoidut tiedostot mainitaan taulukossa 20. Palvelun luokkaan `MerchantAppService` kirjoitettiin `userHasAccessRight`-metodi (tehtävä T4.10), jolla tarkistetaan Drupalin `QueryFactory`-luokan GET-metodin avulla, onko annettua `App Id`:tä vastaava sovellus olemassa ja aktiivinen palvelussa, täsmääkö sovelluksen IP tietokantaan tallennetun kanssa (tehtävä T4.9) ja onko annetulla käyttäjällä oikeus palvelun käyttöön sovelluksen nimissä. Luokan palvelu `entity.query` saatiin käyttöön `mobile_payment_api.merchantapp`-palvelussa lisäämällä se argumentiksi palvelumäärittelytiedostoon `mobile_payment_api.services.yml` (kuviot 15 ja 16).

```
services:
  mobile_payment_api.merchantapp:
    class: Drupal\mobile_payment_api\MerchantAppService
    arguments: ['@entity.query']
```

Kuvio 15. Esimerkki `mobile_payment_api.services.yml`-tiedostosta, joka määrittelee uuden palvelun `mobile_payment_api.merchantapp` ja injektoi siihen argumenttina annetun `entity.query`-palvelun.

```
<?php

namespace Drupal\mobile_payment_api;

use Drupal\Core\Entity\Query\QueryFactory;

/**
 * Class MerchantAppService.
 *
 * @package Drupal\mobile_payment_api
 */
class MerchantAppService {

    protected $entityQuery;

  /**
   * Constructor.
   */
  public function __construct(QueryFactory $entityQuery) {
      $this->entityQuery = $entityQuery;
  }

}
```

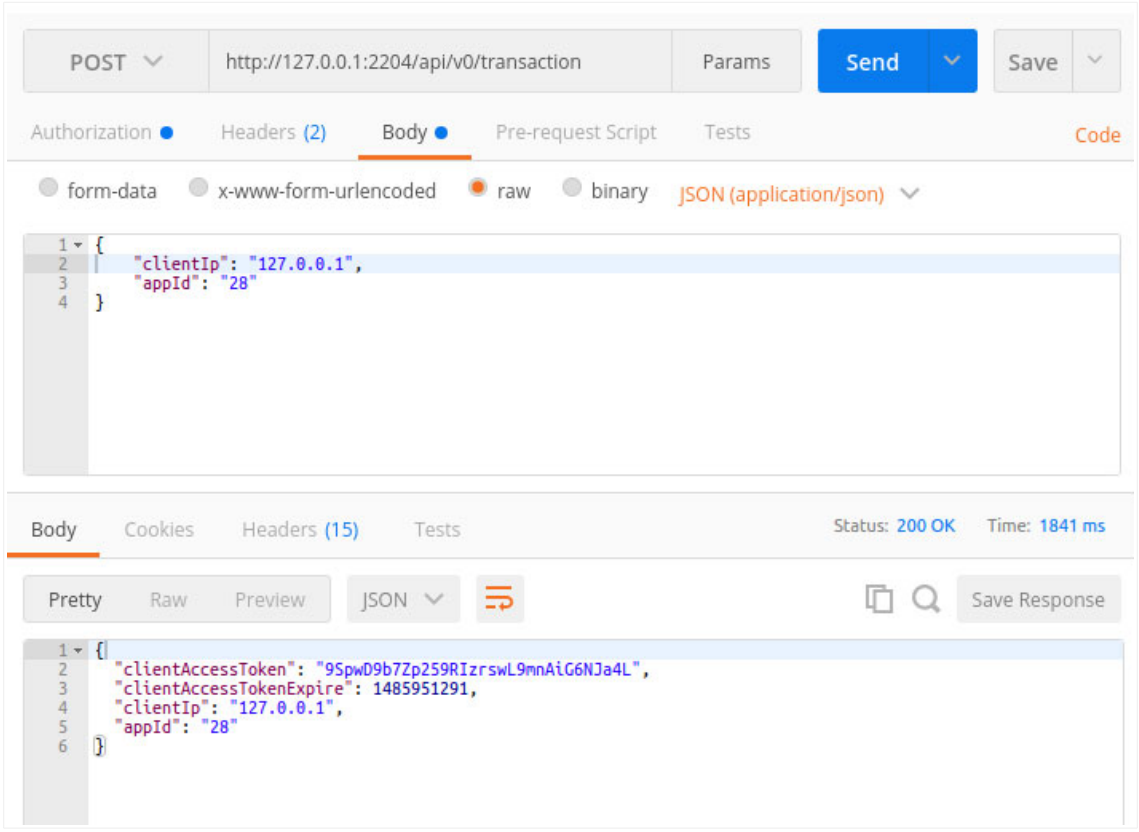
Kuvio 16. Esimerkki `MerchantAppService`-palvelun luokasta ja `entity.query`-palvelun konstruktori-injektiosta siihen.

Tämän jälkeen palvelun metodia voitiin käyttää REST-resurssin pluginista käsin sovelluksen autentikoimiseen, mikä tapahtui järjestyksessä Drupal-käyttäjätunnuksen basic auth -autentikoinnin jälkeen. Sovelluksen IP-osoitteen saaminen vaati sen lukemista Symfonyn Http Foundation -komponentin `RequestStack`-luokan `request_stack`-palvelun `getClientIp`-metodilla, joka on esimerkki siitä, kuinka Drupal 8 -kehittäjä tarvitsee myös Symfony-tuntemusta jo yksinkertaisemmissakin ohjelmointitehtävissä. Jos sovellus ei autentikoitunut, plugin laitettiin tekemään REST-vastaus API-kuvauksen mukaisella virhekoodilla. Käyttäjätunnuksen autentikoinnin epäonnistumiseen REST-vastauksen antaminen ei onnistunut, sillä tämä käsiteltiin jo ennen REST-resurssiin pääsyä basic auth -autentikointitarjoajassa (engl. authentication provider). Asian korjaaminen saattaisi vaatia oman custom-autentikointitarjoajan laatimisen (Drupal.org 2017ac).

Taulukko 22. Transaction-sisältötyyppi kauppiaan sovelluksen maksutapahtumalle. Seuraavassa iteraatiossa 4 sisältötyyppi muutettiin Merchant Transaction -nipuksi ja siihen lisättiin State-kenttä.

Kentän nimi	**Kentän tyyppi**	**Määrittely**
Transaction Id	Title	Transaktiotunnus. Node-entiteettiä käytettäessä title-kenttä on pakollinen.
App Id	Entity reference	Entiteetti-viittaus app-nodeen. AppId on sama kuin App-entiteetin Id.
Client Ip	Text (plain)	Kauppiaan sovellusta käyttävän asiakkaan IP-osoite.
Transaction Id Expire	Timestamp	Transaktiotunnuksen voimassaoloaika
State	List (text)	Maksutapahtuman tilat on esitetty taulukossa 25.

Jotta transaktiotunnus saatiin luotua (tehtävä T4.5), tarvittiin entiteetti sen tallentamiseksi tietokantaan. Transaktioita varten luotiin node-sisältötyyppi Transaction, jonka rakenne on esitetty taulukossa 22. Koska transaktiotunnuksia ei ole tarkoitus katsella, eikä muokata käyttöliittymässä, olisi parempi tapa ollut luoda niitä varten uusi entiteettityyppi. REST-resurssin pluginiin lisättiin toiminto transaktio-noden luomiseksi, mikä toteutettiin try-catch-lohkoja käyttäen. Jos noden luominen ei onnistunut, palautettiin `EntityStorageException`-poikkeus, joka johti API-kuvauksen mukaiseen REST-virhevastaukseen. Transaktiotunnuksen generointiin muokattiin PHP-leikettä, joka löytyi Stack Overflow -yhteisöstä (Scott 2012). Transaktiotunnuksen voimassaoloajaksi määritettiin Unix-aikaleima yhden tunnin päähän. Kauppiaan sovelluksen ilmoittamalle asiakas-IP:lle `Client Ip` tehtiin vain kevyt syötteen muototarkistus, vaikka haluttu toiminnallisuus edellytti IP:n vertaamista suomalaisten mobiilioperaattoreiden IP-osoiteavaruuksiin. IP-osoiteavaruuksien määrittely kannattaa kuitenkin tehdä vasta moduuleissa, jotka integroivat operaattoreiden mobiilimaksu-API:t, jonka jälkeen tehtävää (tehtävä T4.5) voidaan jatkaa. Myös transaktiotunnuksen määräajan umpeutumisen tarkistaminen ja vanhentuneiden transaktiotunnusten poistaminen jäivät tehtäväksi myöhemmin. Esimerkki iteraatiossa syntyneen moduulin toiminnallisuuden testaamisesta Postman-työkalua käyttäen on esitetty kuviossa 17. Yhteenveto iteraation tuloksista on esitetty taulukossa 23.



Kuvio 17. Moduulin toiminnallisuuden testaus Postman-työkalulla. Yläosassa näkyy tehty HTTP-pyyntö ja alaosassa maksuvälitysjärjestelmän API:n vastauksena antama transaktiotunnus ja sen vanhenemisaika Unix-aikaleimana. Tätä ennen palveluun oli tehty kauppiaalle sovellusasetus, joka sai App Id:n 28 ja kytkentä basic auth -menetelmällä autentikoitavaan Drupal-käyttäjätunnukseen. Kuvakaappauksen ottamisen jälkeen maksuvälitysjärjestelmän API-kuvaus muuttui (taulukot 13-17).

Taulukko 23. Itsearvio iteraatio 3:n tuloksista

Yhteenveto iteraatio 3:n tuloksista	
Ajankäyttö:	• Toteutunut ajankäyttö: 29.0 h
Vahvuudet:	• Mobile Payment API -moduuli luotiin. • API-kuvauksen mukainen REST-resurssi saatiin luotua ja otettua käyttöön. • Kauppiaan sovelluksen autentikointi toimii. • Kauppiaan sovellukselle luodaan transaktiotunnus. • Kauppiaan sovellus voidaan määritellä palveluun. • Maksuvälitysjärjestelmän API -kuvauksen mukaisia virheilmoituksia pyritään käyttämään. • Palvelut otettiin käyttöön REST-resurssi-pluginissa tehdas-injektiolla (Bosch 2016) ja moduulin omassa merchantapp-palvelussa konstruktori-injektiolla (Symfony.com 2017).
Heikkoudet:	• Parempi toteutustapa tehdä kauppiaan sovelluksen autentikointi, olisi luoda järjestelmään uusi autentikointitarjoaja, jossa olisi myös OAuth2-tuki. • Transaktiotunnuksia ei ole tarkoitus katsella tai muokata käyttöliittymän kautta, joten parempi toteutustapa olisi luoda niille oma entiteettityyppi, eikä käyttää node-entiteettityyppiä. • Kauppiaan sovelluksen ilmoittamaa asiakas-IP:tä ei tarkisteta, kuuluuko se suomalaisten mobiilioperaattoreiden IP-osoitevaruuksiin. • API ei anna aika tarkalleen oikeaa virheilmoitusta, eikä niissä käytetä HTTP-tilakoodeja oikein. • Virheilmoitukset kannattaisi määritellä muualla kuin koodin seassa.
Johtopäätökset:	• Laaditun API-kuvauksen mukainen maksuvälitysjärjestelmän API on mahdollista luoda Drupalilla custom-moduuleja kehittämällä. • Drupal Console nopeuttaa työskentelyä merkittävästi. • Drupal-kehityksen käytäntöjen oppiminen ottaa aikaa.

5.4 Iteraatio 4: sovellusten autentikointi operaattorin rajapinnassa

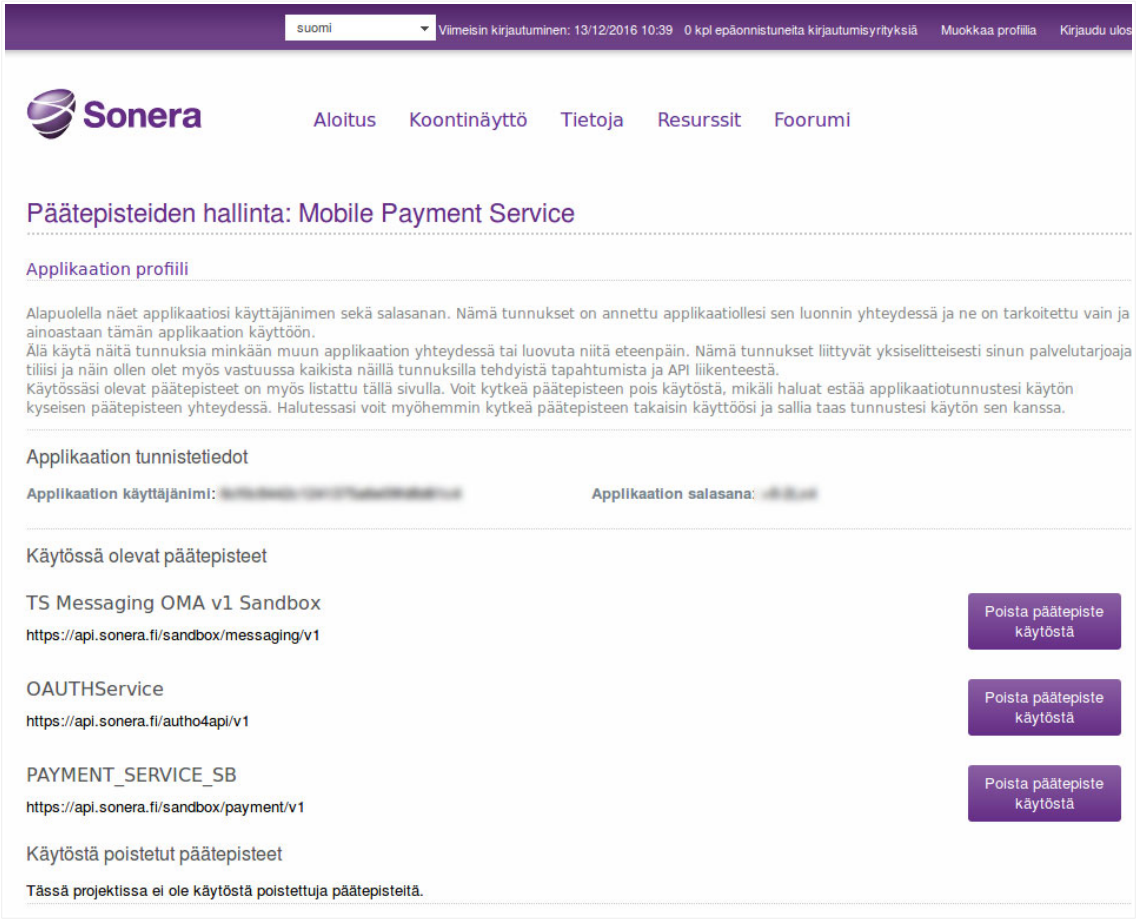
Iteraatiossa 4 tavoitteeksi otettiin toteuttaa sovelluksen autentikointi operaattorin mobiilimaksu-API:ssa. Sovellus autentikoituu ensin maksuvälitysjärjestelmän omassa rajapinnassa, mikä toteutettiin iteraatiossa 3. Iteraatioon otettiin käyttäjätarinat US4, US5, US8 ja US20, joihin liitetyt tehtävät tämän iteraation osalta on listattu taulukossa 24.

Taulukko 24. Iteraatio 4:n tavoite, siihen valitut käyttäjätarinat ja kuhunkin käyttäjätarinaan liittyvät tehtävät.

Iteraatio 4:n tavoite: Tavoitteena oli toteuttaa sovellusten autentikointi operaattorin rajapinnassa maksuvälitysjärjestelmän kautta. Onnistunut tapahtuma palauttaa Soneran OMA Authorization REST API:sta OAuth2 Bearer Token -tunnisteen.	
Käyttäjätarinat:	**Tehtävät:**
US4 Maksuvälitysjärjestelmänä vaadin kauppiaan sovellukselta autentikoinnin REST API:n käyttöön (FR)	T4.6 Service transaktiotunnuksen tarkistamiseen. T4.8 Vastaa api/v0/transaction pyyntöihin API-kuvausta vastaavalla tavalla.
US5 Maksuvälitysjärjestelmänä tarjoan REST API:n POST-metodilla kauppiaan sovelluksen luoda uuden maksutapahtuman (FR)	T5.1 Maksutapahtuma-sisältötyyppi. T5.2 Testaa maksutapahtuman luonti REST API:n kautta. T5.3 Luo "yhden askeleen veloitus"-tyypin maksujen tekemiseen REST-päätepisteen api/v0/transaction/{transactionid}/charge.
US8 Maksuvälitysjärjestelmänä luon loppukäyttäjälle, joka on Soneran tai Tele Finlandin liittymäasiakas, uuden maksun ottamalla yhteyden Soneran mobiilimaksupalvelimen API:iin ja saan vastaukseksi tiedon, onnistuiko vai epäonnistuiko maksu ja maksutapahtuman tunnisteen (FR)	T8.2 Hanki pääsy opaali-portaaliin. Pystytä Sandbox appsi. T8.4 Moduuli autentikoi Soneran palveluun. T8.7 Rajapinta operaattoreiden mobiilimaksu-API-palveluille.
US20 Maksuvälitysjärjestelmänä tarjoan REST API:n kauppiaiden sovelluksille (FR)	T20.5 Luo custom-sisältöentiteetti maksuvälitysjärjestelmän API:n sisällöille. T20.6 Luo API:n virhevastausten generointiin palvelu.

Iteraatio aloitettiin luomalla sovellus eli applikaatio Soneran Opaali-portaalissa (tehtävä T8.2). Applikaation luominen oli hyvin yksinkertainen prosessi, jossa toimeksiantajan palveluntarjoajaprofiilista klikattiin ”Luo applikaatio”-nappia ja annettiin applikaation nimi. Tämän jälkeen sandbox-ympäristön päätepisteet (kuvio 18) olivatkin jo käyttövalmiita. Palvelu generoi automaattisesti applikaation käyttäjätunnuksen ja salasanan, jotka talletettiin maksuvälitysjärjestelmän muistiin. Vasta applikaatiota luotaessa oivalsin, että Viestintäviraston määräyksen 35 R/2016 M (Viestintävirasto 2016b) mukaiset estoluokat määritetään operaattorin päässä jo sovelluksen luontivaiheessa sovelluskohtaisesti. Jokaista maksuvälitysjärjestelmän sovellusta varten on luotava oma applikaatio myös operaattorin palvelussa. Tässä vaiheessa aiempi olettamukseni siitä, että esto-

luokat olisivat maksutapahtumakohtaisia ja että samaa applikaatiota voisi käyttää usealle eri sovellukselle osoittautui vääriksi, joten maksuvälitysjärjestelmä vaati tietorakenteiden eli sisältöentiteettien osalta jonkin verran uudelleensuunnittelua.



Kuvio 18. Kuvakaappaus Sonera Opaali-portaaliin luodun ”Mobile Payment Service”-applikaation hallintaliittymästä. Maksuvälitysjärjestelmän sandbox-ympäristöä varten tarvittiin ”OAUTHService”- ja ”PAYMENT_SERVICE_SB”-päätepisteet. ”TS Messaging OMA v1 Sandbox”-päätepiste oli maksuvälitysjärjestelmässä tarpeeton, joten se poistettiin käytöstä kuvakaappauksen ottamisen jälkeen ”Poista päätepiste käytöstä”-nappia painamalla.

Drupal 8:ssa sisältöentiteetteihin tallennetaan dataa, jota on tarkoitus lisätä ja muokata tuotantoympäristössä, esimerkiksi käyttäjätunnukset. Vaikka esimerkiksi kauppiaan sovelluksen tietojen ja tunnusten määrittelyssä eräässä mielessä onkin kysymys konfiguraation tekemisestä, käytettiin tarkoitukseen sisältö-

eikä konfiguraatioentiteettejä, sillä kauppiaan sovellus tultaisiin määrittelemään kauppiaan tai API:n ylläpitäjän toimesta tuotantoympäristössä. Valitun työnkulun mukaisesti konfiguraatiomuutoksia saa tehdä vain kehitysympäristössä ja muutokset tulee tallentaa myös versionhallintaan.

Kustomoitujen entiteettityyppien käyttö Drupalissa on perusteltua, jos tallennettava data esimerkiksi ei ole websisältöä (kuten websivuja), entiteettien käsittelyyn ei tarvita Drupal-ytimen tai node-sisältötyypin tarjoamaa työnkulkua, sisällön ei haluta löytyvän hakukoneista, datatietueita ei ole tarpeen listata valikoissa tai datan käsittelyyn tarvitaan hyvin mukautettu käyttöliittymä (Zamor 2017). Maksuvälitysjärjestelmän tiedonvarastointitarpeita ajatellen, jokainen perustelu sopi jossain määrin käyttötapaukseen.

Viimeistään tässä vaiheessa osa lukijoista saattaa miettiä, eikö Drupal 8:ssa voisi vain tehdä suoraan tietokantataulua oman moduulin käyttöön? Ei, koska Drupalin ja vastaavien alustojen käytön kantava idea on, että kaikessa – kuten tiedon varastoinnissa – pyritään käyttämään mahdollisimman pitkälle alustan omaa API:a. Se tarjoaa korkealle abstraktiotasolle kehitettyjä useita fiksuja vaihtoehtoja datan tallentamiseen kuten esimerkiksi *Entity API:n* sisällöille ja konfiguraatioille, *State API:n* järjestelmäympäristön tilalle yksittäisinä avain/arvo-pareina, *TempStore API:n* HTTP-pyyntöjen välillä säilytettävälle yksityiselle ja jaetulle datalle ja *Cache API:n* järjestelmän suorituskyvyn parantamiseen (Garfield 2015). Vaikka Drupal 8 käyttääkin Symfony-komponentteja, se on omaksunut Symfonyn Doctrine ORM-oliorelaatiomallinnusteknologiasta käyttöönsä vain PHP-annotaatiot (Drupal.org 2015). PHP-kieli ei itsessään sisällä annotaatioita, toisin kuin esimerkiksi Java tai C# (PHP.net Wiki 2014). Yhteydet tietokantaan hoidetaan esimerkiksi Entity API:ssa query builder -kyselynrakentajateknologialla, eikä SQL-tietokantakyselyiden käyttöä tueta suoraan. ORM-pohjainen ratkaisu on harkinnassa seuraavaan Drupal-ytimen versioon 9. (Drupal.org 2015; Garfield 2009; Garfield 2015.)

Edellä esitetyillä perusteilla node-sisältötyyppien käytöstä päätettiin luopua (tehtävä T5.1) ja siirtyä custom-sisältöentiteettien käyttöön. Mobile Payment API -moduulin tarvitsemille sisältöentiteeteille luotiin uudet entiteettityypit `MobilePaymentEntity` ja `PaymentAppEntity` (tehtävä T20.5) Drupal Consolen

komennolla `drupal generate:entity:content`. Molemmille entiteettityypeille otettiin käyttöön tuki nipuille (engl. bundle), joka mahdollisti useita variaatioita samoista entiteettityypeistä. Moduuli täytyi ensin poistaa ennen kustomoidun entiteettityypin generointia moduuliin, koska entiteettityypin tarvitsemat tietokantataulut luotiin vasta moduulia asennettaessa uudelleen. Drupal Consolen valmisohjelmakoodi ei toiminut sellaisenaan, vaan se vaati useita pieniä korjauksia. Osa korjauksista piti tehdä ennen moduulin asentamista uudelleen, muuten moduuli lakkasi toimimasta ja toimimatonta moduulia ei pystynyt poistamaan Drupalin ylläpitoliittymän, eikä Drupal Consolen avulla.

Koodi generoi myös yksinkertaisen käyttöliittymän, joka mahdollistaa nippujen ja kenttien lisäämisen ja muokkaamisen. `MobilePaymentEntity`-sisältöentiteettityypeiksi tehtiin Transaction-sisältötyypistä (taulukko 22) Merchant Transaction -nippu ja Payment-sisältötyypistä (taulukko 9) Sonera One-Step Charge -nippu. `PaymentAppEntity`-sisältöentiteettityypeiksi tehtiin App-sisältötyypistä Merchant App -nippu. Sisältöentiteetiksi muutettaessa entisten sisältötyyppien tietorakenteisiin tehtiin useita muutoksia ja parannuksia. Esimerkiksi iteraatiossa 1 esitetyt maksutapahtuman tilat (kuvio 8) osoittautuivat käytännön kannalta epärelevanteiksi. Uudet maksutapahtuman tilat otettiin käyttöön Merchant Transaction -nipussa (taulukko 25). Maksutapahtuman tilojen mallintamisesta uudeksi tilakonekaavioksi olisi tässä vaiheessa ollut hyötyä ohjelmakoodin rakenteen syntetisoimiseksi, mutta tästä luovuttiin aikataulullisista syistä, mikä tosin saattoi seuraavassa iteraatiossa 5 johtaa suurempaan määrään bugeja.

Taulukko 25. Merchant Transaction -nipun State-kentän sallitut arvot kuvaavat maksutapahtuman tilaa sen eri vaiheissaan. Kun prosessi on päättynyt, sitä ei voida enää jatkaa, vaan kauppiaan sovelluksen on käynnistettävä uusi maksutapahtuma. Vaaleanharmaalla taustalla olevat tilat eivät kuuluneet opinnäytetyössä tehtävän Esi-Alfa-1-prototyypin laajuuteen.

Tila	**Prosessi**	**Selitys**
Uninitiated	Kesken	Loppukäyttäjällä on Mobiilimaksu-maksutavan tukema IP-osoite, joten kauppiaan sovelluksen käyttöön on luotu maksutapahtumatunnus, mutta loppukäyttäjää ei ole vielä tunnistettu operaattorin asiakasselvityspalvelussa, eikä loppukäyttäjän veloituslupaa ole annettu.
Initiated	Kesken	Loppukäyttäjä on tunnistettu operaattorin asiakasselvityspalvelussa ja loppukäyttäjän veloituslupa on annettu.
Reserved	Kesken	Loppukäyttäjältä on otettu katevaraus ”usean askeleen veloitus”-ominaisuudella. Ei käytössä Esi-Alfa-1-prototyypissä.
Committed	Päättynyt	Loppukäyttäjän katevaraus on veloitettu ”usean askeleen veloitus”-ominaisuudella. Ei käytössä Esi-Alfa-1-prototyypissä.
Released	Päättynyt	Loppukäyttäjän katevarauksen veloittamisesta on luovuttu ”usean askeleen veloitus”-ominaisuudella. Ei käytössä Esi-Alfa-1-prototyypissä.
Charged	Päättynyt	Loppukäyttäjää on veloitettu ”yhden askeleen veloitus”-ominaisuudella.
Error	Päättynyt	Maksutapahtuma on joutunut virhetilaan, eikä sitä enää voida jatkaa.

Jotta kauppiaiden sovellukset voitiin autentikoida Soneran palvelussa (tehtävä T8.4), tehtiin Mobile Payment -sisältöentiteetille uusi nippu Sonera App, jonka rakenne on esitetty taulukossa 26. Koska sovellukset tulee ensin luoda käyttöliittymää käyttäen Soneran Opaali-kehittäjäportaalissa, eikä suoraa API-liittymää tätä varten ole toistaiseksi saatavissa, jää Sonera App -nipun entiteettien luominen myös maksuvälitysjärjestelmän puolella API:n ylläpitäjän tehtäväksi. Näin ollen viittaus kauppiaan määrittelemään sovellukseen oli syytä tehdä Sonera App -sisältöentiteetissä. Jos tuotteen kehitys jatkuu opinnäytetyön jälkeen, jokainen uusi integroitava mobiilimaksu-API tulee tarvitsemaan oman App-nippunsa, jolle tehdään yksilöllinen rakenne.

Taulukko 26. Sonera App -nippu kauppiaiden sovellusten integroimiseksi Soneran mobiilimaksu-API:iin. Seuraavassa iteraatiossa 5 lisättiin nippuun kenttä Api client resolver.

Kentän nimi	**Kentän tyyppi**	**Määrittely**
Name	name	Sovelluksen nimi. Käytetään samaa nimeä kuin Sonera Opaalissa.
API username	Text (plain)	Sovelluksen käyttäjätunnus Soneran API:ssa. Huom: vaihdettava myöhemmin kryptatuksi kentäksi tai tiedostoksi.
API password	Text (plain)	Sovelluksen salasana Soneran API:ssa. Huom: vaihdettava myöhemmin kryptatuksi kentäksi tai tiedostoksi.
API token	Text (plain)	Sovelluksen OAuth Bearer token Soneran API:ssa. Huom: vaihdettava myöhemmin kryptatuksi kentäksi tai tiedostoksi.
Api token expire	Timestamp	Sovelluksen OAuth Bearer token voimassaolon päättymisaika Unix-aikaleimana.
API oauth endpoint	URL	Soneran antama URL sovelluksen autentikoinnille.
API payment endpoint	URL	Soneran antama URL sovelluksen maksuliikenteelle.
Merchant app	Entity reference	Entiteetti-viittaus kauppiaan sovellukseen.
Api client resolver	URL	Soneran antama sovelluksen asiakasselvityksen URL.

Iteraatio 4 sisälsi myös useita pienempiä prototyypin toimintaa ja rakennetta kohentavia tehtäviä, jotka tehtiin ennen Sonera-integraatiota. `MerchantAppService`-palveluun tehtiin parannuksia kauppiaan sovelluksen ja kauppiaan sovelluksen asiakkaan autentikointiin mm. transaktiotunnuksen tarkastamiseksi (tehtävä T4.6). Maksuvälitysjärjestelmän API-kuvauksessa (taulukot 13–17) oli esitetty virheilmoituksille useita virhekoodeja helpottamaan tulevaa kauppiaiden sovellusten integroimista ja ylläpitoa, mutta nämä oli implementoitu REST-vastauksiksi iteraatiossa 3 epämääräisellä tavalla. Moduuliin lisättiin `ResponseCodesService`-palvelu (tehtävä T20.6) sekä onnistuneiden että epäonnistuneiden REST-vastausten kodifiointiin, mutta aikataulullisista syistä ohjelmakoodin kirjoittaminen jäi keskeneräiseksi, joten kaikkiin virhetiloihin ei vieläkään aina saada tarkkaa vastausta (tehtävä T4.8). ”Yhden askeleen veloitus”-maksutapahtuman toteutus (tehtävät T5.2 ja 5.3) vaati uuden

`api/v0/transaction/{transactionid}/charge` REST-päätepisteen luomisen maksuvälitysjärjestelmään, mikä toteutettiin pitkälti samalla tavalla kuin iteraatiossa 3 toteutettu päätepiste. Uutena ominaisuutena resurssin polusta kerättiin 32-merkkiä pitkä `transactionId`-tunnus, jolla yksittäinen maksutapahtuma identifioitiin.

Iteraation lopuksi päästiin toteuttamaan Soneran mobiilimaksu-API:n integraatiota. Maksuvälitysjärjestelmän Esi-Alfa-1-prototyypin toteutuksessa perimmäinen tavoite oli toteuttaa ”Yhden askeleen veloitus”-tyyppinen Mobiilimaksu-maksutapahtuma. Lisähaasteena oli, että vaikka moduuliin integroidaan ensiksi vain Soneran mobiilimaksu-API, se tulisi suunnitella siten, että valmius muidenkin operaattoreiden mobiilimaksu-API:en käyttöönottoon olisi mahdollista ilman merkittävää tarvetta uudelleensuunnittelulle. Näin ollen kaikkien operaattoreiden mobiilimaksu-API:t täytyi tuntea ensin hyvin, mihin pyrittiin iteraatiossa 2.

Ongelma ratkaistiin järjestelmän sisäisellä ohjelmointirajapinnalla ja ohjelmakomponenttirakenteella (tehtävä T8.7). Jokaisen integroitavan operaattorin palvelusta päätettiin tehdä oma custom-moduuli ja kaikki moduulit päätettiin sijoittaa Mobile Payment API -moduulin kanssa yhteiseen Mobile Payment -moduuliryhmään. Jokaisen operaattorin mobiilimaksu-API:n integroiva moduuli sai riippuvuudekseen Mobile Payment API -moduulin. Mobile Payment API -moduuliin määriteltiin palvelurajapinta (kuvio 19), joka operaattori-integraatio-moduuleiden tulisi toteuttaa. Näin jokaisen operaattorin mobiilimaksurajapinta näyttäytyy maksuvälitysjärjestelmän sisällä maksuvälitysjärjestelmälle samanlaisena, samoilla metodeilla käytettävänä palveluna. Soneran mobiilimaksu-API:n integroimista varten generoitiin Sonera Mobile Payment API integration -moduuli ja siihen tehtiin palveluluokka `SoneraOperatorApiService`, joka toteutti `OperatorApiServiceInterface`-palvelurajapinnan. Uusi moduuli lisättiin PHPStormissa samaan projektiin Mobile Payment API -moduulin kanssa sisältöjuuriasetusta muuttamalla.

```php
<?php

namespace Drupal\mobile_payment_api;

/**
 * Interface OperatorApiServiceInterface.
 *
 * @package Drupal\mobile_payment_api
 */
interface OperatorApiServiceInterface {

    /**
     * Method to check, does the IP belong to operator's mobile payment IP pool
     *
     * @return boolean Results of checking
     */
    public function belongsToOperatorIpPool($ip);

    /**
     * Method to save details received from operator's msisdn resolver service to handle the transaction in continue
     *
     * @param $transactionid
     * @param $endUser
     * @param null $operatorTransactionID (optional)
     * @return boolean Results of making the setting, true or false
     */
    public function setOneStepChargeTransactionDetails($transactionid, $endUser, $operatorTransactionID = NULL);

    /**
     * Method to get operator's entity, that contains data for one-step-charge transaction
     *
     * @param $transactionid
     * @return entity|boolean Returns entity if succeed, otherwise returns false
     */
    public function getOneStepChargeTransactionEntity($transactionid);

    /**
     * Method to resolve operator's client; implements merchant's client interaction with operator
     *
     * @param string $transactionid 32-chars length id of a single transaction event
     * @return string|boolean Returns path to operator msisdn resolver for this client if succeed, otherwise returns false
     */
    public function resolveOperatorClient($transactionid);

    /**
     * Method to make one-step-charge with this operator's client
     * Pre-requirements:
     * - merchant transaction status is "Initiated"
     * - merchant's client is recognized
     * - one_step_charge mobile_payment_entity is created
```

```
     *
     * @param string $transactionid 32-chars length id of a
single transaction event
     * @param float $amount Charging amount with 2 decimals in
currency
     * @param string $currency 3-chars length currency code (ISO
4217)
     * @param $description Max 255-chars length description of
transaction, will be shown in client's phone invoice
     * @param null $chargingInfoCode Max 255-chars length
reference code (optional)
     * @return array Results of charging client
     */
    public function createOneStepCharge($transactionid, $amount,
$currency, $description, $chargingInfoCode = NULL);
}
```

Kuvio 19. PHP-koodina maksuvälitysjärjestelmän OperatorApiServiceInterface-palvelurajapinta, jonka jokaisen operaattori-integraatiomoduulin tuli toteuttaa. Kommentit ovat PHPDoc-standardin mukaisessa DocBlock-muodossa (PhpDocumentor 2017). On syytä huomata, että rajapinta sisältää merkittäviä suunnitteluvirheitä: esimerkiksi se ei saisi koskaan vaatia operaattori-integraatiomoduulin app- tai transaction-entiteettejä käsiteltäväksi moduulin ulkopuolella.

Soneran mobiilimaksu-API käytti loppukäyttäjän veloittamiseen OMA Payment REST API -rajapintaa, joka vaati Oauth 2.0 Bearer token -autentikoinnin käyttöä. Token oli pyynnöt valtuuttava merkkijono, joka lisättiin HTTP-otsikon Authorization-avaimen arvoksi maksua tehtäessä, millä pyynnöt valtuutetaan, esimerkki: `Authorization: Bearer <token value>`. Uuden tokenin noutamiseen (tehtävä T8.4) ja käytöstä poistamiseen käytettiin OMA Authorization REST API -rajapintaa. (Sonera 2016b; 2016c.) Sonera Payment API integration -moduulissa tämä toteutettiin yksityisenä metodina, jota kutsuttiin `createOneStepCharge`-metodista käsin. Pyynnön toteuttamiseen käytettiin Drupalin `http_client`-palvelua, joka oli Guzzle HTTP-asiakasinstanssi. Token oli voimassa vain 10 minuuttia, jonka aikana voitiin käyttää samaa tokenia useiden pyyntöjen valtuuttamiseen. Tokenin vanhennuttua oli pyydettävä uusi token. Sekä token että voimassaolon päättymisaika tallennettiin Sonera App -entiteettiin, joten uusi token pyydettiin vasta, kun vanhan tokenin vanhenemiseen oli aikaa 10 sekuntia. Iteraation 4 tulokset on esitetty taulukossa 27. Maksun veloittaminen jäi seuraavassa iteraatiossa toteutettavaksi.

Taulukko 27. Itsearvio iteraatio 4:n tuloksista.

Yhteenveto iteraatio 4:n tuloksista	
Ajankäyttö:	• Toteutunut ajankäyttö: 24.0 h
Vahvuudet:	• Oman sovelluksen käyttöönotto onnistui vaivatta Soneran Opaali-portaalissa. • Mobiilimaksu API -moduulin entiteetti-tietorakenteet saatiin käyttökelpoiselle kehitystasolle. • Bearer token -valtuuden nouto ja uudelleennouto Soneran rajapinnasta onnistui. • Kaikkien operaattoreiden mobiilimaksu-API:t saadaan käyttöön samaa rajapintaa käyttäen. • Operaattoreiden API-integraatioiden kapselointi erillisiksi moduuleiksi helpottaa niiden käyttöönottoa, käyttöä ja jatkokehitystä maksuvälitysjärjestelmässä. • Toteutuksessa käytettiin kaikkialla riippuvuuden injektiota, eikä globaalin Drupal-luokan staattisia metodeita.
Heikkoudet:	• Sonera mobiilimaksu-API:n käyttäjätunnus, salasana ja bearer token tallennettiin salakirjoittamattomassa muodossa tietokantaan; parempi käytäntö olisi ollut tallentaa tiedot suoraa HTTP-dokumenttijuuren ulkopuoliseen levytilaan salakirjoitetussa muodossa. • Kaikkia maksuvälitysjärjestelmän vaatimia syötteiden tarkistuksia ei tehty. • Ei käytetty `entity.query`-palvelun edistyneempiä ominaisuuksia, jotka olisivat mahdollistaneet mm. ehtoryhmät ja SQL-kielestä tutut liitokset. • Yksikkötestejä ei kirjoitettu, vaan testaus tehtiin vain manuaalisesti Postman-työkalulla. • Drupal Consolen valmisohjelmakoodi sisälsi virheitä. • Uutta tilakonekaaviota maksutapahtuman tiloista ei tehty, mikä saattoi johtaa suurempaan määrään ohjelmointivirheitä seuraavassa iteraatiossa 5. • OperatorApiServiceInterface-palvelurajapinta ei saisi koskaan vaatia operaattori-integraatiomoduulin app- tai transaction-entiteettejä käsiteltäväksi moduulin ulkopuolella.
Johtopäätökset:	• Drupal-ydin näyttää sisältävän kaikki prototyypin tarvitsemat ominaisuudet, joten toteutuksen tekeminen oli suhteellisen vaivatonta, vaikkakin aloittelijan kannalta haastavaa. • Drupal Console -työkalu nopeutti työskentelyä, vaikka valmisohjelmakoodi vaatikin korjailua.

5.5 Iteraatio 5: maksutapahtuman toteutus

Iteraatiossa 5 tavoitteeksi otettiin Esi-Alfa-1-prototyypin viimeistely. Prototyyppi jo autentikoitui Soneran palveluun, mutta ”yhden askeleen veloitus”-maksutapahtuman toteuttaminen Soneran OMA Payment REST API -rajapintaa käyttäen oli vielä tehtävänä. Liittymäasiakkaat SIM-kortin perusteella tunnistavaan Soneran MSISDN Enrichment -palveluun ei ollut pääsyä maksutta Opaalin sandbox-ympäristön käyttäjällä, mutta vähintään tämän toiminnallisuuden simulointi tuli sisällyttää prototyyppiin. Prototyypin REST API:n kokeileminen oli jo mahdollista mm. Postman-työkalua käyttäen, mutta prototyypin oheen tuli toteuttaa myös Mobiilimaksu-napin sisältävä lomake, jolla maksuvälitysjärjestelmän toimintaa voisi kokeilla loppukäyttäjän ja verkkokauppiaan näkökulmasta. Näin ollen iteraatioon 5 valikoitiin käyttäjätarinat US5, US7 ja US8 (taulukko 28).

Valitettavasti ohjelmistotestien suunnittelua ja kirjoittamista (käyttäjätarina US21) ei tiukan aikataulun vuoksi voitu sisällyttää mukaan viimeiseenkään iteraatioon edes yksikkötestauksen muodossa, mikä on merkittävä puute. Toisaalta ohjelmistotestaus oli tarkoitus aloittaa varsinaisesti vasta alfa-tason prototyypeissä, kun ohjelmiston kaikki MVP-julkaisua varten tarvittavat ominaisuudet ovat prototyypissä jo mukana ja koodipohjaa on ensin refaktoroitu. Esi-alfa-1:ssä oli tarkoitus olla vain tärkeimmät ydinominaisuudet, mikä vastaa lähinnä ns. teknologiademon toteutusta.

Taulukko 28. Iteraatio 5:n tavoite, siihen valitut käyttäjätarinat ja kuhunkin käyttäjätarinaan liittyvät tehtävät.

Iteraatio 5:n tavoite: Iteraation tavoitteena oli Esi-Alfa-1-prototyypin loppuun saattaminen ja viimeistely. Tärkeintä oli luoda puuttuvat toiminnallisuudet eli ”yhden askeleen veloitus”-maksutapahtuman tekeminen ja Soneran MSISDN-selvityksen simulointi. Tehtäväksi otettiin myös prototyypin demokäytön mahdollistaminen Postman-työkalun lisäksi Mobiilimaksu-napin sisältävällä lomakkeella.	
Käyttäjätarinat:	**Tehtävät:**
US5 Maksuvälitysjärjestelmänä tarjoan REST API:n POST-metodilla kauppiaan sovelluksen luoda uuden maksutapahtuman (FR)	T5.2 Testaa maksutapahtuman luonti REST API:n kautta T5.3 Luo "yhden askeleen veloitus"-tyypin maksujen tekemiseen REST-päätepisteen api/v0/transaction/{transactionid}/charge
US7 Maksuvälitysjärjestelmänä saadessani tiedon uudesta maksusta, selvitän ensin, minkä operaattorin asiakas maksaja on eli operaattorin rajapintaa käytetään maksun veloittamiseen (FR)	T7.1 Luo service/pay/sonera HTTP-päätepiste T7.2 Luo api/v0/transaction/{transactionid}/resolve HTTP-päätepiste T7.3 Luo Soneran MSISDN-selvitystä simuloiva päätepiste T7.4 Validoi asiakas-IP:t operaattorin mukaan jo transaction-id:tä annettaessa
US8 Maksuvälitysjärjestelmänä luon loppukäyttäjälle, joka on Soneran tai Tele Finlandin liittymäasiakas, uuden maksun ottamalla yhteyden Soneran mobiilimaksupalvelimen API:iin ja saan vastaukseksi tiedon, onnistuiko vai epäonnistuiko maksu ja maksutapahtuman tunnisteen (FR)	T8.5 Moduuli tekee one-step chargen Soneran palveluun T8.8 Mobiilimaksu-testilomake T8.9 API vastaa pyyntöön, onnistuiko vai eikö onnistunut maksutapahtuma

Iteraatio aloitettiin ”yhden askeleen veloitus”-ominaisuuden toteutuksella (tehtävä T8.5), mikä vaati myös korjauksia ja täydennyksiä maksuvälitysjärjestelmän REST-päätepisteeseen `api/v0/transaction/{transactionid}/charge` pyyntöjen ja vastausten käsittelyyn (tehtävä T5.3). Mobile Payment API -moduulin tarjoaman `OperatorApiServiceInterface`-rajapinnan toteuttavaan `SoneraOperatorApiService`-palveluun kirjoitettiin toiminnallisuudet `createOneStepCharge`-metodille. Drupalin-ytimen mukana tulleen Guzzle-kirjaston käyttö Soneran API:n kutsumiseksi sijoitettiin omaan yksityiseen metodiinsa. Pyynnön piti vastata täysin tarkasti OMA Payment REST API -kuvausta, muuten vastauksena oli erilaisia HTTP-virheilmoituksia; jopa virhe 500 Internal Server Error esiintyi, kun JSON-rakenteessa oli pieni virhe. Guzzlen debug-toiminnosta ei ollut apua monimutkaisen JSON-rakenteen ja -sisällön sekä HTTP-otsikkotietojen saamiseksi täysin kohdilleen, joten avuksi kirjoitettiin oma

`testi.php`-skripti, joka tallensi HTTP Header- ja Raw Body -tiedot kehityspalvelimen levytilaan. Vertaamalla pyyntöä Soneran API-kuvauksen esimerkkipyyntöön, saatiin pyyntö lopulta kohdilleen. Jos Soneran API-dokumentaatio ei olisi sisältänyt esimerkkipyyntöjä ja -vastauksia JSON-muodossa, tehtävässä olisi kestänyt paljon kauemmin. Java-kielellä kirjoitettuihin Soneran esimerkkikoodeihin en kuitenkaan tutustunut. Debuggauksessa ei ollut apua Soneran Opaali-portaalin QueryDumpForPartner-raportista, josta näkyi tietoja sekä onnistuneista että epäonnistuneista pyynnöistä, kunhan pyynnön URL-polku ja palveluun autentikoituminen oli ensin kohdillaan, jotta dataa pääsi tallentumaan Soneran puolella. Pyynnön polku sisälsi MSISDN:n eli matkapuhelinnumeron muodossa `tel:+35840123456`, mutta Guzzle osasi koodata URL:n erikoismerkit heksadesimaaliarvoiksi oma-aloitteisesti. Try-Catch-lohkorakenteen käyttäminen oli Guzzlen kanssa välttämätöntä, jotta HTTP-virheet eivät olisi kaataneet ohjelman suoritusta. Operaattorilta saadut tiedot tallennettiin Sonera one-step charge -entiteetteihin ja maksu merkittiin tilansa mukaisesti veloitetuksi tai epäonnistuneeksi Merchant Transaction -entiteetteihin. Maksuvälitysjärjestelmän API:n vastaukset pyrittiin hiomaan sekä onnistuneiden että epäonnistuneiden vastausten osalta lähelle iteraatiossa 2 tehtyä API-suunnitelmaa, jota kylläkin oli muokattu käyttötarkoitusta paremmin vastaavaksi myös myöhempien iteraatioiden aikana.

Seuraavaksi toteutettavaksi otettiin MSISDN Enrichment -palvelun simulointi (tehtävä T7.3), jotta sen käyttämiseksi tarvittavat toiminnallisuudet (tehtävät T7.1, T7.2 ja T7.4) saatiin tehtyä maksuvälitysjärjestelmään. Maksuvälitysjärjestelmä käyttää kaikessa liikenteessä suojattua HTTPS-protokollaa, mutta MSISDN Enrichment -palvelun käyttö tuli toteuttaa suojaamattomalla HTTP-protokollalla (Sonera 2016a), jotta liikenteen reitittäminen edelleen ja pyynnön muokkaus reitityksen eri vaiheissa saataisiin toimintaan. Tämä vaati Drupalin ajamista samanaikaisesti kahdella eri HTTP-protokollalla ja kahdessa eri portissa. Apache2 HTTP-palvelimessa otettiin käyttöön SSL-moduuli itseallekirjoitetulla sertifikaatilla ja SSL-portti määriteltiin palvelimen Virtual Host -asetuksiin. Itseallekirjoitetun sertifikaatin vuoksi Guzzlen HTTP-pyynnöissä täytyi kehitysympäristön osalta falsivoida verify-optio, jotta sertifikaatin alkuperää ei tarkasteta; muuten HTTPS-yhteys ei toiminut. Drupalin asetukset Esi-Alfa-1-prototyypin

vaatimusten täyttämiseksi eivät muutoin tarvinneet muutoksia. Vasta MVP-versio tuotteesta vaatisi tarkan pääsynhallinnan, jossa vaaditaan kaikille poluille suojattu yhteys, mihin voitaisiin tehdä erikseen määritettyjä poikkeuksia. MVP-version korkea tietoturvataso vaatisi, että pääsynhallinnasta olisi hyvä tehdä monikerroksinen mm. siten, että käytetään niin palomuuria, HTTP-palvelimen pääsynhallintaa kuin vielä Drupalin omia protokolla-, portti- ja IP-tarkistuksiakin.

Kun loppukäyttäjä painaa Mobiilimaksu-nappia, kutsutaan operaattorista riippumatta ensimmäiseksi maksuvälitysjärjestelmän `api/v0/transaction/{transactionid}/resolve`-päätepistettä. Päätepiste luotiin (tehtävä 7.2) Drupal Consolen `drupal generate:controller`-komennolla ResolveClientController-kontrolleriksi Mobile Payment API -moduuliin (kuviot 20 ja 21). Päätepisteen käyttöön ei tehty REST-pluginia kuten kahden edellisen luodun päätepisteen tapauksessa, koska päätepistettä käytetään vain ohjaamaan pyyntö edelleen jonkin operaattorin MSISDN-selvityspalveluun. Koska ainakin Tieto Connectionin IP Connectivity Charging API:ssa (Tieto Connection / VAS Center 2014) selvityspalvelun URI-polku vaikutti olevan sovelluskohtainen, täytyi maksuvälitysjärjestelmässä kaikkien operaattoreiden osalta selkeyden ja varmuuden vuoksi tallentaa selvityspalvelun URL joka sovellukselle erikseen Payment App Entity -entiteetteihin ja noutaa se `ResolveClientController`-kontrollerissa. Kontrolleriin lisättiin mm. transaktiotunnuksen ja asiakas-IP:n tarkistus. Käyttöön otettiin myös `/service/error-`fallback-osoite tilanteita varten, joissa transaktiotunnusta ei pystytä validoimaan, jolloin myöskään edelleenohjaus ei ole mahdollinen mihinkään suuntaan. Loppukäyttäjän IP-osoitteen kuuluminen tuettujen operaattoreiden IP-osoiteavaruuksiin tarkastetaan jo aiemmassa vaiheessa, kun kauppiaan sovelluksen palvelinpuoli noutaa transaktiotunnuksen maksuvälitysjärjestelmästä, joten tässä vaiheessa asiakas-IP:tä vain verrattiin jo olemassaolevaan Merchant Transaction -entiteetin tietoon.

```
<?php
namespace Drupal\mobile_payment_api\Controller;

use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Symfony\Component\HttpFoundation\RequestStack;

/**
 * Class ResolveClientController.
 *
 * @package Drupal\mobile_payment_api\Controller
 */
class ResolveClientController extends ControllerBase {

    protected $request;
    protected $user;

    public function __construct(RequestStack $requestStack,
AccountProxyInterface $currentUser) {
        $this->request = $requestStack;
        $this->user = $currentUser;
    }

    /**
     * {@inheritdoc}
     */
    public static function create(ContainerInterface $container)
{
        return new static(
            $container->get('request_stack'),
            $container->get('current_user')
        );
    }

    /**
     * forwardToOperator
     *
     * @param $transactionid
     * @return
     */
    public function forwardToOperator($transactionid) {
        // do something here
        return;
    }

}
```

Kuvio 20. Esimerkki kontrollerin luomisesta ja tehdasinjektiosta palveluiden käyttämiseksi kontrollerissa. Create-tehdasmetodin on palautettava palvelut samassa järjestyksessä palvelusäiliöstä kuin niiden luokat on annettu konstruktorille.

```
mobile_payment_api.resolve_client_controller_forwardtooperator:
  path: '/api/v0/transaction/{transactionid}/resolve'
  defaults:
    _controller:
'\Drupal\mobile_payment_api\Controller\ResolveClientController::f
orwardToOperator'
    _title: 'forwardToOperator'
  requirements:
    _permission: 'access content'
```

Kuvio 21. Esimerkki kontrollerin tarvitsemasta `mobile_payment_api.routing.yml`-tiedostosta, joka kytkee määritellyn staattisen polun kontrollerin metodiin ja välittää kontrollerille polusta transactionid-parametrin.

Asiakasselvitystietojen välittyminen päätepisteeltä toiselle koetettiin toteuttaa ensin HTTP-pyynnön osoitetietoja muokkaamalla. Web-selaimet eivät kuitenkaan säilytä HTTP 302 Found -edelleenohjausvastauksissa lisättyjä otsikkotietoja edelleenohjaavissa HTTP-pyynnöissä, eikä teknisesti haastavampaa pyynnön proxy-edelleenvälitystä pystytty tiukan aikataulun vuoksi tekemään. JavaScriptiä ei otsikkotietoja muokkaavaan edelleenohjaukseen voida käyttää, koska operaattoritkaan eivät sitä tähän käytä ja koska HTTP-pyyntöjä MSISDN-selvitykseen voi tulla myös muista sovelluksista kuin web-selaimista. Niinpä MSISDN Enrichment -palvelun simulointi (tehtävä T7.3) toteutettiin yksinkertaisesti HTTP 302 Found -edelleenohjauksilla lisäämällä URL-polkuun tieto fiktiivisen Soneran asiakkaan matkapuhelinnumerosta GET-parametrina `x-up-calling-line-id`. Simuloinnissa matkapuhelinnumeroa ei myöskään kryptattu. Tarkoitusta varten tehtiin `SoneraResolverResultsController`-kontrolleri, joka vastasi sekä simuloinnista että `/service/pay/sonera`-päätepisteestä. Myös transaktiotunnus muutettiin `api/v0/transaction/{transactionid}/resolve`-päätepisteessä GET-parametriksi. Varmuutta GET-parametrin edelleenvälityksen toimivuudesta ei ollut Soneran ja Tieto Connectionin mobiilimaksu-API:en osalta, koska tietoa tästä ei onnistuttu löytämään dokumentaatiosta, mutta Elisan mobiilimaksu-API:n dokumentaatiossa yhden tunnistetiedon käyttö GET-parametrina oli selitetty hyvin seikkaperäisesti (Elisa yritysasiakaspalvelu 2015, 6–8), joten se uskallettiin ottaa käyttöön myös Soneran vastaavaa palvelua simuloitaessa.

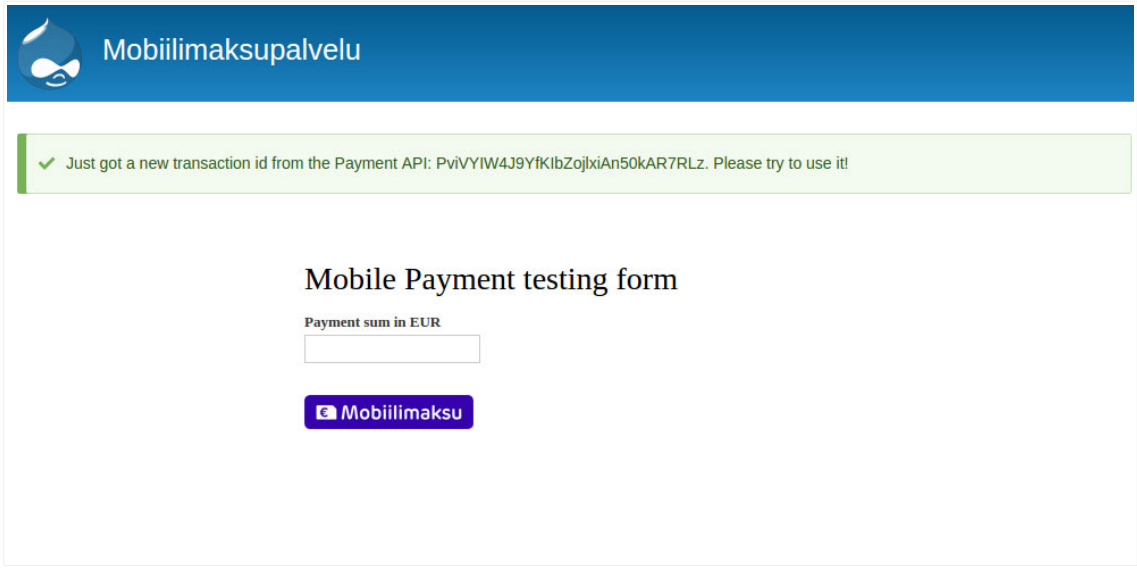
Operaattorilta tulevien vastausten käsittelyyn maksuvälitysjärjestelmässä luotiin `/service/pay/sonera`-päätepiste (tehtävä T7.1), jossa MSISDN luettiin talteen ja jonka polun muoto oli Soneran määrittämä (Sonera 2016a). Vastauksena Sonera toimitti vain MSISDN:n eli matkapuhelinnumeron, kun Elisan ja Tieto Connectionin vastauksissa maksuvälityspalvelu sai lisäksi mm. operaattorin generoiman transaktiotunnuksen. Sonera toimitti transaktiotunnuksen vasta, kun maksutapahtuma oli toteutettu eli Soneralla pelkkä matkapuhelinnumero riitti liittymäasiakkaan yksilöimiseksi, kunhan asiakas oli ensin selvitetty. Päätepisteessä poistettiin `x-up-calling-line-id`-GET-parametri ja lisättiin `resolved`-GET-parametri välittämään tieto kauppiaan sovellukselle, oliko loppukäyttäjälle myyntilupaa vaiko ei. Transaktion statusta muutettiin vastaavasti Merchant Transaction -entiteettiin. Tiedot lähtivät edelleen HTTP 302 Found -ohjauksen URL-polussa kauppiaan sovelluksen palvelinpuolen tulkittaviksi. Edelleenohjausketjun debuggaukseen käytettiin Chromium-selaimen Developer-sivupaneelin Network-kuunteluominaisuutta, josta voitiin nähdä GET-parametrien ja otsikkotietojen muutokset jokaisen väliaskeleen osalta.

Maksuvälitysjärjestelmän prototyypin demonstrointia varten luotiin uusi Drupal-moduuli Test Merchant (tehtävä T8.8), joka näytti fiktiiviselle loppukäyttäjälle mobiilimaksunapin ja antoi syöttää testisumman, hoiti maksutapahtuman maksuvälitysjärjestelmän API:a käyttäen ja ilmoitti tulokset käyttäjälle uudella sivulla. Lomake muistutti käyttötapausta, jossa asiakas valitsee maksutavan verkkokaupan kassalla. Lomake yritettiin luoda Drupal Consolen komennolla `drupal generate:form`, mutta kävikin ilmi, että kyseinen komento oli Consolen kehittäjiltä vahingossa tipahtanut pois ainakin tästä Consolen versiosta (1.0.0-rc16). Niinpä lomake tehtiin alusta lähtien Drupal-dokumentaation Form API -ohjeiden (Drupal.org 2017ad) mukaisesti. Drupal-lomakkeen submit-napin korvaaminen Mobiilimaksu-napilla vaati Drupal 8:ssa CSS-tiedoston määrittelemisen moduulin `test_marchant.libraries.yml`-asetustiedostoon (kuvio 22) ja kirjaston liittämisen lomakkeeseen `#attached`-elementillä (Drupal.org 2017ae). Kuvakaappaus testilomakkeesta on esitetty kuviona 23 ja onnistunut esimerkkivastaus lomakkeen lähettämiseen kuviona 24. Jotta lomakkeen toteutus oli nopeampaa, lähetettiin lomake GET-metodilla suoraan maksuvälitysjärjestelmän `api/v0/transaction/{transactionid}/resolve`-päätepisteeseen, jolloin

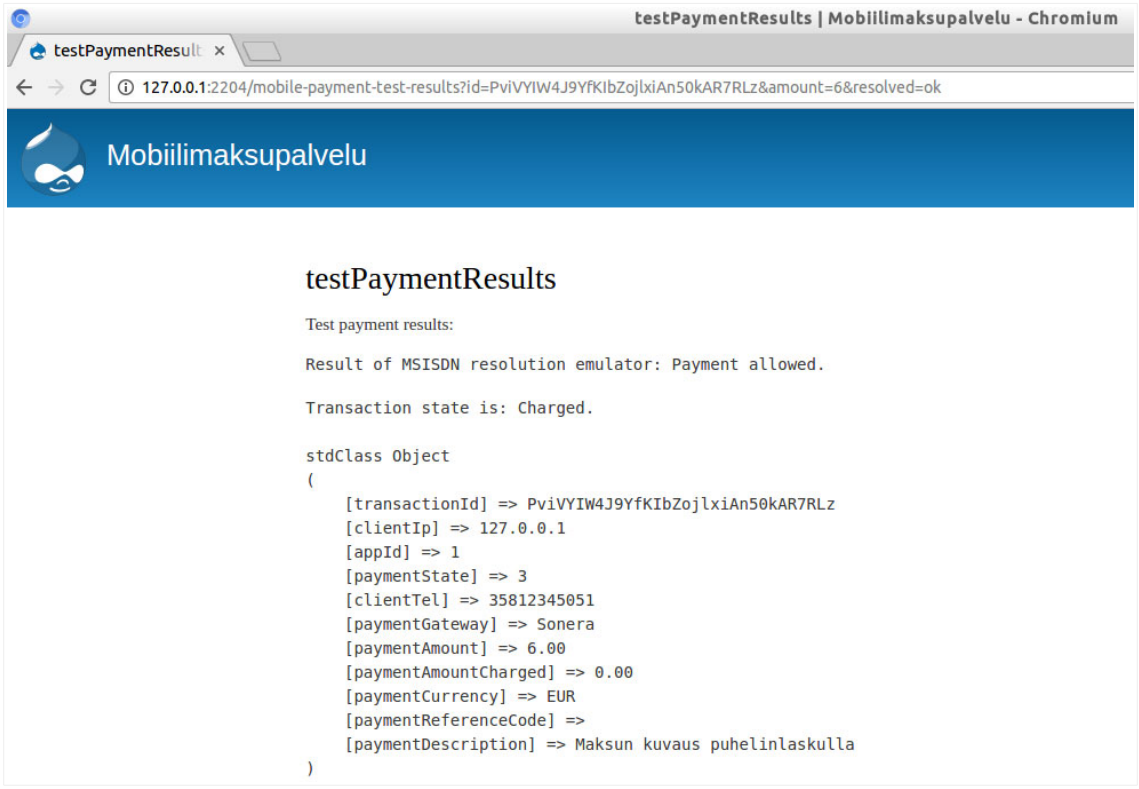
myös maksun summa seurasi GET-parametrina pyynnön mukana. Tuotantokäytössä tarkoitus kuitenkin on, että kauppiaan sovelluksen palvelinpuoli tietää veloitettavan summan viimeistään, kun Mobiilimaksu-nappia painetaan eli summaa ei ole tarkoitus kierrättää maksuvälitysjärjestelmän ja mobiilioperaattorin MSISDN-selvityspalvelun kautta.

```
mobiilimaksu:
  version: 1.x
  css:
    theme:
      css/mobiilimaksu.css: {}
```

Kuvio 22. Esimerkki `test_marchant.libraries.yml`-asetustiedostosta, joka liittää moduuliin `mobiilimaksu`-kirjaston, johon CSS-tiedosto kuuluu.



Kuvio 23. Kuvakaappaus testilomakkeesta maksuvälitysjärjestelmän Mobiilimaksu-maksutavan demonstrointiin. Ilmoitusalueella näkyi tieto transaktiotunnuksesta, joka oli haettu ennen lomakkeen näyttämistä maksuvälitysjärjestelmältä.



Kuvio 24. Kuvakaappaus testilomakkeen vastaussivusta. MSISDN-simulaation tuloksena saatiin myyntilupa. Transaktion tila oli ”Charged” eli veloitus oli onnistunut. Vastaussivulla näytettiin maksuvälitysjärjestelmältä vastaanotettu JSON-vastaus PHP:n `print_r()`-function tulosteena. Osoiterivillä näkyivät GET-parametrit. Soneran mobiilimaksu-API:n vastaukset maksuvälitysjärjestelmä piti omana tietonaan. Kuvassa näkyvä dummy-matkapuhelinnumero oli peräisin Soneran julkisesta sandbox-ympäristön dokumentaatiosta (Sonera 2017).

Iteraation tarkoituksena oli myös Esi-Alfa-1-prototyypin viimeistely. Jotta maksuvälitysjärjestelmän testaaminen testityökaluilla (kuten Postman) olisi helpompaa, tehtiin `MerchantAppService`-palveluun mahdollisuus kytkeä maksuvälitysjärjestelmä testimoodiin, joka ohittaa asiakasselvitysvaiheen (tehtävä T5.2). Testilomakkeen luomisen ja MSISDN-selvityksen simuloinnin ansiosta maksuvälitysjärjestelmän koodia saatiin paranneltua ja debugattua, mutta aikataulullisista syistä prototyypin viimeistely jäi kuitenkin käytännössä tekemättä. Vaikka prototyyppi olikin toimiva, siihen jäi mm. suunnitteluvirheitä, joiden korjaamiseen tarvittaisiin vielä yksi iteraatio. Iteraation tulokset on esitetty taulukossa 29.

Taulukko 29. Itsearvio iteraatio 5:n tuloksista.

Yhteenveto iteraatio 5:n tuloksista	
Ajankäyttö:	• Toteutunut ajankäyttö: 34.0 h
Vahvuudet:	• ”Yhden askeleen veloitus”-mobiilimaksu saatiin toimintakuntoon Soneran API:n kanssa. • Maksuvälitysjärjestelmän toimintaa voi demonstroida sekä REST-työkalulla että maksunapin sisältävällä testilomakkeella. • Drupal saatiin toimimaan ongelmitta sekä HTTP- että HTTPS-protokollalla samanaikaisesti kahdessa eri portissa. • Maksuvälitysjärjestelmän koodi parani iteraation kuluessa ja toteutti sille asetetut tavoitteet.
Heikkoudet:	• Yksikkötestausta ei tehty viimeisessäkään iteraatiossa. • Soneran MSISDN Enrichment -palvelun kanssa teknisesti samanveroista simulointia ei ehditty tekemään proxy-teknologialla. • Maksuvälitysjärjestelmän vastaukset eivät antaneet API-kuvauksen mukaista HTTP-tilakoodia, mikä unohdettiin tehdä. • Maksuvälitysjärjestelmän koodi sisälsi edelleenkin bugeja ja merkittäviä ohjelmointiteknisiä virheitä, jotka olisi välttämätöntä korjata.
Johtopäätökset:	• Esi-alfa-1 prototyyppi saatiin valmiiksi. • Operaattoreiden MSISDN-selvityksen integroiminen maksuvälitysjärjestelmään vaatii palvelun hankkimisen operaattorilta jo kehitysvaiheessa.

6 Tulokset

Projektin tuloksia on hyvä katsoa vielä kokonaisuutena, vaikka niitä käytännössä onkin jo esitelty ja arvioitu iteraatioittain raportin toteutusosuudessa. Opinnäytetyön tuloksena syntyi Drupal 8 -alustalle rakennettu Mobiilimaksujen maksuvälitysjärjestelmän prototyyppi, jossa oli tärkeimmät järjestelmältä vaaditut ydinominaisuudet. Prototyyppiin oli tehty verkkokauppojen ja mobiilisovellusten käyttöön soveltuva REST-rajapinta Mobiilimaksu-maksutavan toteuttamiseksi. Prototyyppiin integroitiin yhden operaattorin mobiilimaksu-API ”yhden askeleen veloitus”-maksutapahtumatyypin osalta ja pyrittiin luomaan valmius muiden operaattoreiden vastaavien mobiilimaksu-API:en lisäämiseksi järjestelmään modulaarisesti. Lisäksi prototyypissä simuloitiin SIM-kortin perusteella tehtävää Mobiilimaksun asiakasselvitystä, jotta prototyypin toiminta saatiin kehitysympäristössä vastaamaan paremmin sen toimintaa tuotantoympäristössä. Simulointi mahdollisti myös Mobiilimaksu-napin sisältävän testilomakkeen luomisen.

Tuloksia voidaan lähestyä useasta eri näkökulmasta. Verkkokauppiaan ja loppukäyttäjän näkökulmasta prototyypin tulokset demonstroitiin iteraation 5 lopussa kuvaamalla testilomakkeen toimintaa kuvakaappauksin (kuviot 23 ja 24). Maksutapa toimi hyvin suoraviivaisesti: kun maksettava summa oli asetettu, painettiin Mobiilimaksu-nappia, jonka jälkeen maksutapahtuma joko onnistui tai epäonnistui.

Maksuvälitysjärjestelmän kaltaisten web-API:en integroinnista kiinnostuneiden kehittäjien näkökulmasta prototyypin tuloksissa keskeistä lienee se, mitä kyselyitä API:iin on tehtävä maksutapahtuman toteuttamiseksi ja mitä vastauksia API antaa. Kehitystyön tuloksena prototyypin toiminta täytti iteraatiossa 2 esitetyn maksuvälitysjärjestelmän API-kuvauksen (taulukot 13–17), minkä perusteella integraation tekeminen pitäisi olla mahdollista. Ketteriä menetelmiä käyttäen dokumentaatio tavallisesti painottuu työn loppuun, joten API-kuvauskin päivitettiin vastaamaan tilaa, jossa API oli prototyypin valmistuttua. Lähinnä HTTP-tilakoodien ja API:n omien virhekoodien käytön osalta prototyypin vastaukset eivät olleet täysin API-kuvauksen mukaisia.

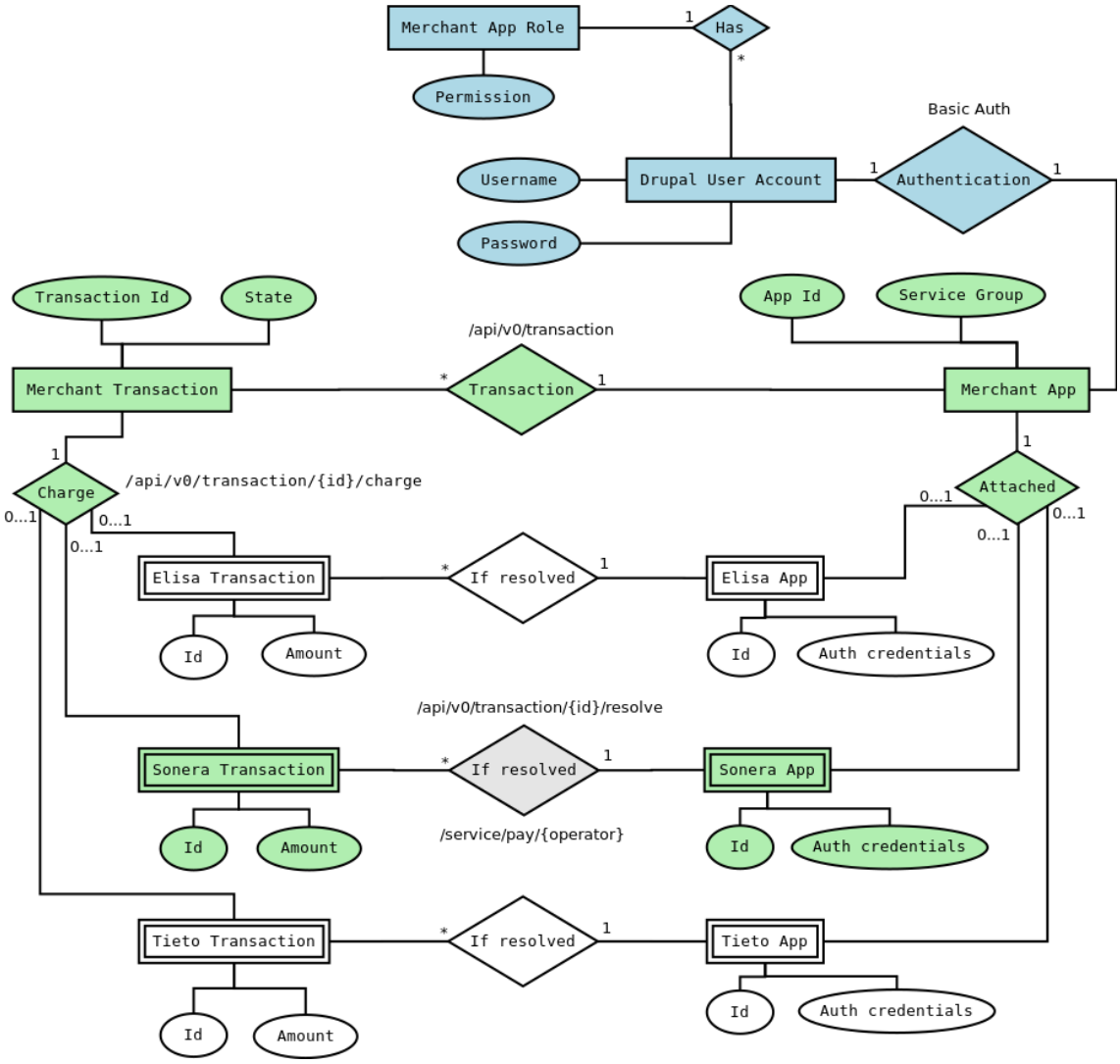
Opinnäytetyön varsinaiselle kohderyhmälle eli Drupal 8 -websovellusten suunnittelusta ja toteuttamisesta kiinnostuneille kehittäjille ei valitettavasti voida esittää projektissa syntynyttä lähdekoodia, sillä työn tulokset kuuluvat toimeksiantajalle. Prototyyppi sisälsi koodia noin 4200 riviä (LOC), josta noin puolet kirjoitettiin käsin ja puolet oli Drupal Consolen generoimaa. Valmisohjelmakoodin suuri osuus selittyy sillä, että Drupal Consolella generoitiin kaksi entiteettityyppiä, joista kumpikin vaati vajaat 1000 koodiriviä. Vaikka koodia ei julkaistakaan, voidaan prototyypin tietorakenteita kuitenkin kuvailla Entity Relationship -kaaviona (kuvio 25) ja ohjelmakoodia moduuleittain UML-luokkakaavioiden muodossa (kuviot 26–28).

Tietorakenteet esiteltiin suurimmaksi osaksi jo opinnäytetyön toteutusosuuden iteraatioiden kuvauksissa erilaisina sisältöentiteettitaulukkoina, joten yksityiskohtaisen tietokantakaavion sijaan kokonaiskuvan järjestelmästä riittää antamaan myös pelkistetympi ER-kaavio (kuvio 25), johon on piirretty rakenteen hahmottamisen helpottamiseksi vain olennaisimmat attribuutit. Maksuvälitysjärjestelmän ylläpitäjän tehtävät uuden sovelluksen lisäämiseksi tietorakenteeseen alkavat Drupal-käyttäjätunnuksen luomisella sovellukselle, jolle annetaan kauppiassovellusrooli. Yhtä käyttäjätunnusta tulisi käyttää vain yhdelle sovellukselle. Seuraavaksi luodaan Merchant App -entiteetti, johon käyttäjätunnus kytketään. Tämän jälkeen sovellus konfiguroidaan operaattoreiden puolella heidän järjestelmiinsä, josta tuloksena saatavat käyttötiedot lisätään Elisa App, Sonera App ja Tieto App -entiteetteihin, jonka jälkeen Mobiilimaksu-maksutapa on sovelluksen osalta maksuvälitysjärjestelmän puolella valmis käytettäväksi. Yhtä transaktiota kohtaan on tarkoitus käyttää vain yhden operaattorin app- ja transaction-entiteettejä, koska loppuasiakas voi käyttää istunnossa vain yhden operaattorin mobiili-internetyhteyttä. Operaattorilta voidaan pyytää transaktion veloittamista vasta, kun Mobiilimaksu-nappia on painettu ja loppukäyttäjän asiakkuus on selvitetty. Asiakkuuden selvittäminen johtaa joko Elisa Transaction, Sonera Transaction tai Tieto Transaction -entiteetin luomiseen maksutapahtumalle. Prototyyppiin toteutettiin edellä kuvatut entiteettirakenteet vain Soneran osalta.

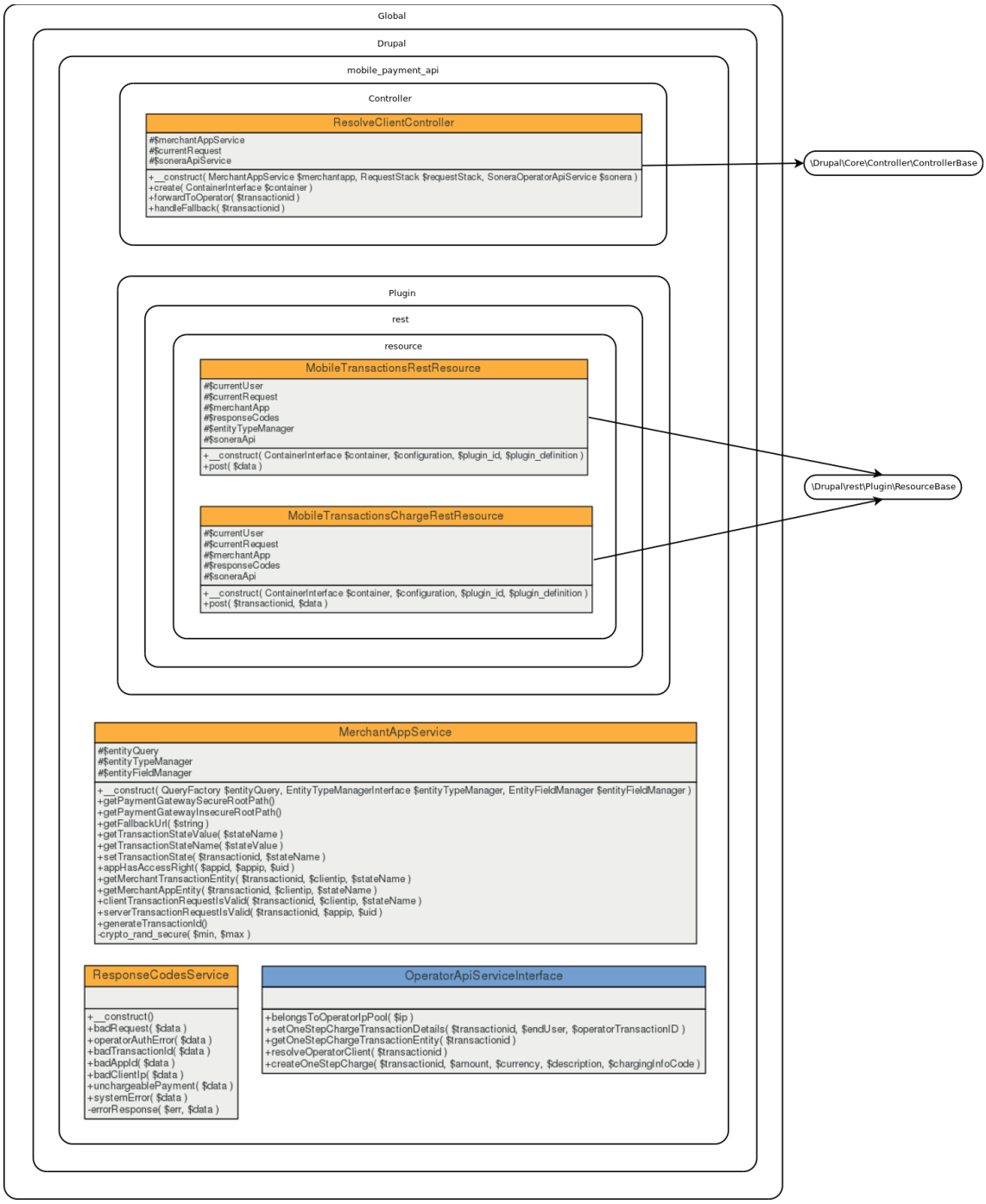
Moduuleiden luokkakaavioista (kuviot 26–28) voidaan havaita, että projektissa tuotettiin koodia paljon enemmän kuin mitä oli mahdollista esittää toteutusosuuden iteraatioiden kuvauksissa. Prototyyppi koostui kolmesta moduulista: *Mobile*

Payment API toteutti järjestelmän pääpiirteittäin, *Sonera Mobile Payment API* lisäsi toteutukseen Sonera mobiilimaksu-API:n integraation ja *Test Merchant* toi saataville yksinkertaisen testilomakkeen järjestelmän kokeilua varten. Mobile Payment API -moduulin `MerchantAppService`-palvelu muodostui järjestelmässä keskeiseksi kirjaston kaltaiseksi luokaksi, johon turvauduttiin jatkuvasti muista luokista käsin. Sen sijaan saman moduulin `ResponseCodesService`-palvelu jäi keskeneräiseksi, koska tarve olisi ollut kehittää siitä API:n kaikki vastaukset muotoileva palvelu. Luokkien metodien ja attribuuttien nimet ovat melko deskriptiivisiä, joten niiden toimintaa ja tarkoitusta ei tässä yhteydessä avata tarkemmin.

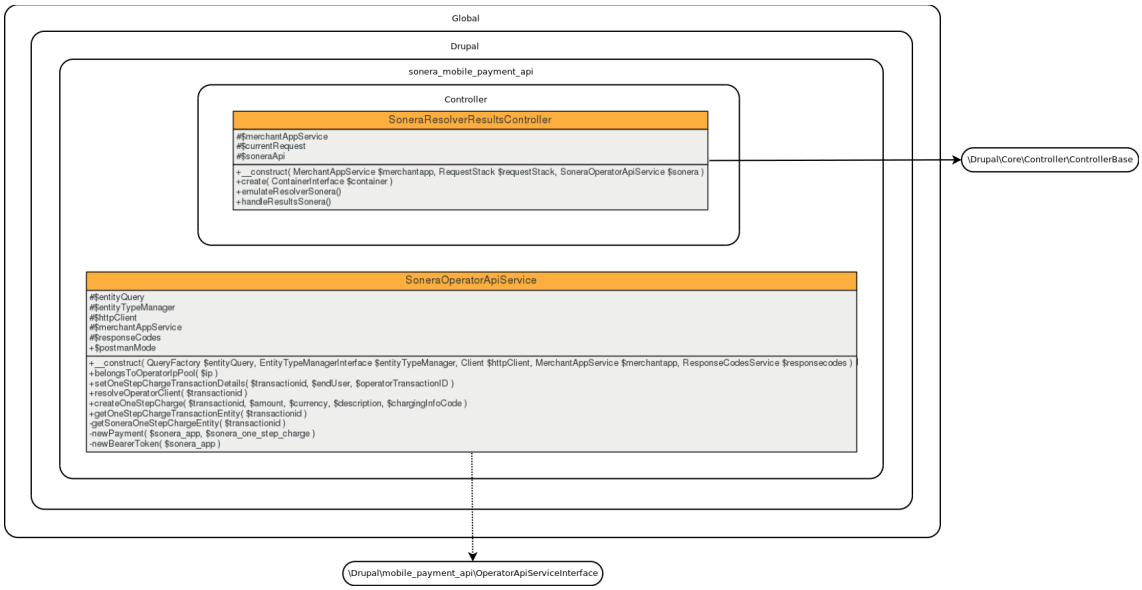
UML-luokkakaavioiden luonnissa käytettiin apuna phUML- ja phpDocumentor-generaattoreita, joiden tuottamat kaaviot yhdistettiin kuvankäsittelyohjelmalla ja Dia-CASE-työkalulla. Lähdekoodista poistettiin moduuleiden luokkakaavioiden generoinnin ajaksi edellä mainitut Drupal Consolen tuottamat entiteettikoodit, sillä niitä käytettiin vain entiteettityyppien luomiseen ja muokkaamiseen, eikä niillä ollut järjestelmän toiminnan kannalta muuta merkitystä. Sen sijaan transaktioentiteettien luominen, kyseleminen, lataaminen ja päivittäminen kuin myös app-entiteettien kyseleminen ja lataaminen olivat järjestelmässä keskeisiä toimintoja, joihin koodi kirjoitettiin alusta alkaen itse.



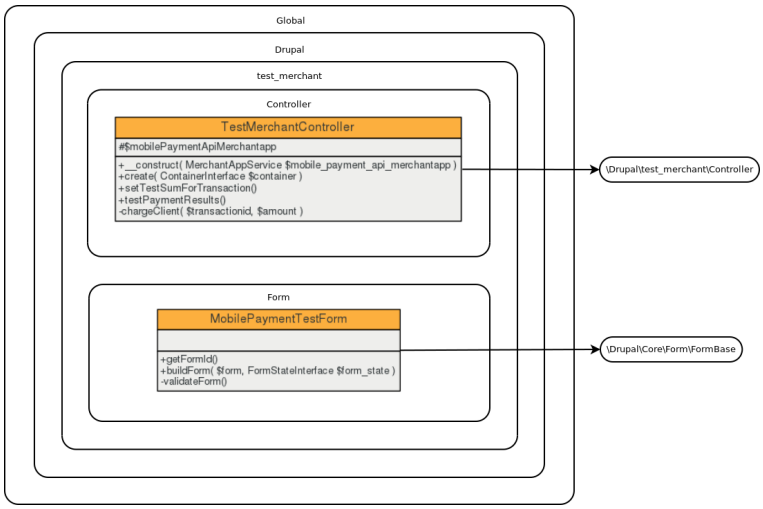
Kuvio 25. Maksuvälitysjärjestelmän yksinkertaistettu tietorakenne ER-kaaviona. Sininen väri indikoi Drupal-ytimessä olevia tietorakenteita. Vihreä väri indikoi prototyypissä toteutettuja tietorakenteita. Valkoinen väri indikoi ei-toteutettuja tietorakenteita, jotka kuitenkin pyrittiin ottamaan huomioon. Harmaa väri indikoi suhteen muodostamisen ehtoa eli operaattorin asiakasselvityksen tulosta, jota prototyypissä yritettiin simuloida, mutta jota ei vielä varsinaisesti toteutettu. Jos suhteen syntyminen ratkaistaan REST API:n kautta, merkittiin suhteen läheisyyteen tapaukseen liittyvä URI-polku. Suhteen yhteyteen merkittiin myös sen kardinaliteetti.



Kuvio 26. Mobile Payment API -moduulin luokkakaavio. Ympärillä olevat kehykset kuvaavat nimiavaruuksia. Aliluokkien periytyminen yliluokista merkittiin nuolilla.



Kuvio 27. Sonera Mobile Payment API -moduulin luokkakaavio. Ympärillä olevat kehykset kuvaavat nimiavaruuksia. Aliluokan periytyminen yliluokasta merkittiin nuolella, jossa on yhtenäinen viiva. Luokan toteuttama rajapinta merkittiin nuolella, jossa on katkoviiva.



Kuvio 28. Test Merchant -moduulin luokkakaavio. Ympärillä olevat kehykset kuvaavat nimiavaruuksia. Aliluokkien periytyminen yliluokista merkittiin nuolilla.

7 Pohdinta

Opinnäytetyön tulokset vastaavat mielestäni hyvin sille asetettuja tavoitteita. Tulosten vastaavuus tavoitteiden kanssa on kuitenkin suhteellista, sillä opinnäytetyön tavoitteita rajattiin ja supistettiin niin aloitusvaiheessa kuin projektin kuluessakin. Jos tavoitteita ei oltaisi karsittu, prototyyppi olisi varsin kaukana perimmäisestä tavoitteesta eli maksuvälitysjärjestelmän MVP-julkaisuversiosta ja opinnäytetyön tuloksia voitaisiin pitää jokseenkin epäonnistuneina. Opinnäytetyön laajuus on kuitenkin vain 15 opintopistettä, mihin toimeksiannon täytyi mahtua, joten aiheen rajaaminen oli perusteltua.

Opinnäytetyöprosessi itsessään muistuttaa hieman ohjelmistotuotannon perinteistä vesiputousmallia, koska se jakautuu kolmeen perättäiseen vaiheeseen – suunnitteluun, toteutukseen ja raportointiin – joista kunkin vaiheen ajankäytön tulisi vastata noin kolmasosaa (5 op) opinnäytetyön laajuudesta. Aiheen rajaamisen jälkeen toteutusosuuden ajankäyttö oli 170 h, mikä vastasi noin kuutta opintopistettä. Tuntimäärä ei kuitenkaan sisällä mm. raportin kirjoittamisen yhteydessä tehtyä mittavaa selvitystyötä, mikä tapahtui pitkälti iteraatioiden lomassa toteutusosuuden tekemistä tukien. Ketterän ohjelmistokehityksen periaatteiden (Beck ym. 2001) soveltaminen paljon epävarmuustekijöitä sisältävässä projektissa on opinnäytetyönä haastava tehtävä. Esimerkiksi maksuvälitysjärjestelmän tapauksessa työtä ei ollut mahdollista suunnitella kovinkaan tarkasti etukäteen, sillä opinnäytetyöntekijällä ei ollut liioin Drupal 8 -back-end-ohjelmoinnista kuin Mobiilimaksu-maksutavan API-toteutuksesta aiempaa kokemusta.

Iteratiivinen ketterä toteutustapa ja vyöryvän aallon suunnittelutapa osoittautuivat projektissa kuitenkin oikeiksi valinnoiksi. Suunnittelun tuli alkaa kevyesti ja teoreettista tietoa täytyi kasvattaa vähitellen kokemuksen kautta. Projektin suunnitteluaineistoa oli hyvä päivittää ja laajentaa vähitellen projektin edetessä. Vasta käytännön toteutustyön myötä ja teoreettisen tiedon syvennyttyä huomattiin mille tietorakenteille ja malleille oli tarvetta. Esimerkkinä tästä on iteraatiossa 1 tehty maksutapahtuman tilojen mallintaminen (kuvio 8) tilakonekaavioksi ja Payment-sisältötyypin toteutus (taulukko 9), jotka tulevissa iteraatioissa osoittautuivat hiukan naiiveiksi esikäsityksiksi siitä, miten sovellus toimisi ja mitä tietorakenteita se tarvitsisi. Tilakonekaaviosta ei käytännön tasolla ollut juuri hyö-

työ. Dokumentaatiota kannattaa tehdä enemmän ohjelmistoprojektin loppupuolella, jolloin siitä tulee relevanttia. Eräs hyvä tapa tuottaa esimerkiksi UML-malleja tai API-dokumentaatioita onkin niiden generoiminen vasta, kun ohjelmiston tietty osavaihe on jo kehitetty kuten esimerkiksi opinnäytetyön tulosten esittelyä varten tehtiin luvussa 6.

Toisaalta hyvin harkiten ennen toteutukseen ryhtymistä tehty minimitarpeet täyttävä suunnittelu ja mallintaminen voi toimia arvokkaana apuvälineenä myös ketterässä kehityksessä, mihin Agile Modeling -menetelmä opastaa (Ambler 2002, 8-10). Esimerkiksi iteraatiossa 2 tehty UML-sekvenssikaavio (kuvio 13) ”yhden askeleen veloitus”-maksutapahtuman kulusta kuvauksineen (taulukko 12) vei toteutusprosessia eteenpäin ratkaisevalla tavalla, sillä se auttoi hahmottamaan kokonaisuuden, mitä maksuvälitysjärjestelmään tulisi ohjelmoida ja mistä työskentely kannatti aloittaa. Kaavio ja sen kuvaus sekä samassa iteraatiossa tehty API-kuvaus (taulukot 13-17) vaativat kuitenkin päivittämistä käytännössä joka iteraatiossa vastaamaan toteutuksessa kohdattuja realiteetteja. Tehty mallinnus oli käyttökelpoinen, joten se oli hyvä pitää aina ajan tasalla.

Raportin kirjoittaminen toteutuksen ohessa ei ollut tärkeää vain siksi, että se kuului raportin työskentelyä kuvaavaan esitystapaan, vaan myös siksi, että toteutuksen jatkaminen ei ollut useassa vaiheessa mahdollista ilman teoreettisen pohjan täydentämistä. Uutta tietoa omaksuessa se pyrittiin samassa yhteydessä kiteyttämään myös raporttiin tai muistiinpanoihin. Koska kirjallisuutta Drupal 8 -back-end-kehityksestä ei ollut julkaistu vielä opinnäytetyön tekemisen aikaan, parhaaksi tiedon lähteeksi osoittautui Google ja sivustot Drupal.org ja Symfony.com. Drupal.org-sivuston dokumentaatio oli vielä monin paikon puutteellista, joten tietoa oli etsittävä myös kokeneiden Drupal-kehittäjien blogeista. Drupal kehittyy kaiken aikaa, joten esimerkiksi Drupal 8.0.x:aa ja 8.1.x:aa varten laaditut dokumentaatiot omien REST-pluginien teosta olivat vanhentuneet jo muutamassa kuukaudessa Drupal 8.2.x:n julkaisuun mennessä (Drupal.org 2017o). Sama havainto koskee Borchert & Schirwinskin vuonna 2015 ilmestynyttä Drupal 8 Configuration Management -kirjaa, jonka tiedot eivät havaintojeni perusteella pitäneet täysin paikkaansa enää syksyllä 2016. Odotettavissa siis on, että tämänkin opinnäytetyön sisältö ja lähteet tulevat vanhenemaan nopeasti.

Erityisesti Drupal.org-sivuston dokumentaatio muuttuu koko ajan, mikä on huomattava opinnäytetyön lähteitä tutkittaessa.

Toteutusprosessia iteraatioittain kuvaileva ja selittävä raportointitapa tuntui hyvältä idealta, mutta osoittautui käytännössä hyvin työlääksi. Opinnäytetyöraportin pituus voi herättää lukijassa kysymyksen: lieneekö projektissa käynyt niin, että kattavaa dokumentointia olisi arvostettu sittenkin enemmän kuin toimivaa ohjelmistoa? Kysymys on paikallaan, sillä prototyypin viimeisteleminen olisi vaatinut vielä yhden iteraation, johon ei riittänyt aikaa massiivisen raportin viimeistelemisen vuoksi. Tästä syystä prototyypistä ja koko projektista paistaa läpi keskeneräisyyden maku. Iteraatiossa 6 tehtäviä viimeistelytoimia olisivat olleet mm. yksikkötestien suunnittelu ja toteutus sekä koodipohjan refaktorointi eli rakenteen parantelu. Kattava testaus olisi ollut välttämätöntä, jotta olisi voitu varmistua siitä, että koodi on toimivaa ja että tulevat muutokset koodiin eivät hajota järjestelmää. Refaktoroinnin tarve perustuu useisiin opinnäytetyöntekijän havaintoihin. Tärkeimmät havainnot olikin jo raportoitu heikkouksina iteraatioiden itsearviointivaiheissa (taulukot 10, 18, 23, 27 ja 29); esimerkiksi `ResponseCodesService`-palvelu tulisi kirjoittaa uudestaan huolehtimaan järjestelmän vastausten muotoilusta kokonaisvaltaisesti, operaattori-integraatiomoduulin entiteettejä ei pitäisi koskaan käsitellä moduulin ulkopuolella ja entiteettikyselytoiminnoissa pitäisi käyttää mm. liitoksia. Lisäksi PHP Mess Detector eli PHPMD-työkalulla ohjelmakoodista havaittiin taulukossa 30 esitetyt ongelmat, joihin refaktoroinnissa olisi hyvä kiinnittää huomiota.

Taulukko 30. Projektin koodipohjasta PHP Mess Detector -työkalulla löydetyt ongelmat.

Kohde	**Havaittu ongelma**
Kaikki luokat	Useiden metodien NPath ja syklomaattinen kompleksisuus oli korkea (Pichler 2017). Useissa metodeissa käytettiin else-lauseketta, vaikka se ei olisi tarpeellista. Osassa muuttujia ja parametrejä ei käytetty ns. camelCase-kirjoitusasua.
Luokka MerchantAppService	Luokan kokonaiskompleksisuus oli hyvin korkea.
Luokka SoneraOperatorApiService	Osa metodeista oli hyvin pitkiä. Osa muuttujien nimistä oli hyvin pitkiä.

Pohdinnan perusteella tärkeimmäksi opinnäytetyön tulokseksi voitaisiinkin nostaa opinnäytetyöntekijän oma oppimiskokemus. Ohjelmoimaan voi oppia vain ohjelmoimalla. Kehityskulku vastasi melko hyvin Viking Code School -koulutusyrityksen blogissa (Trautman 2015) esitettyä neljää vaihetta, jotka esiteltiin opinnäytetyön johdantoluvussa. Alkuinnostuksen jälkeen ohjelmointityö tuntui valtaosan ajasta ”autiomaassa harhailulta”, kunnes projektin loppuvaiheessa – tarkkaan ottaen iteraation 5 aikana – koettiin loppua kohti nouseva huipentuma, jossa osaaminen ja produktiivisuus nousi tasolle, jolla varmistui, että toimiva prototyyppi saatiin ajoissa tehtyä. Projektia voisi luonnehtia tutkivan oppimisen hengessä tehdyksi matkaksi ennalta tuntemattomalle alueelle, jossa pyrittiin asiantuntijalle ominaiseen tapaan luoda uutta tietoa ja tehdä luovia kokeiluja (Suomen virtuaaliyliopisto 2017).

Vaikka ohjelmoinnissa olikin haasteita, käytetyistä menetelmistä ja työkaluista oli paljon hyötyä. Kados-projektinhallinnan käyttö piti työskentelyn kaiken aikaa tavoitteellisena ja tavoitteet kohtuullisen kokoisina. Käyttäjätarinat auttoivat pitämään fokuksen olennaisessa ja pysymään hyvin kartalla siitä, mitä oli tehtävä ja minne suuntaan oli mentävä. Drupal Console -komentorivityökalu helpotti toteutuksen tekemistä merkittävästi, kun useissa ohjelmointitehtävissä päästiin nopeammin alkuun. PHPStorm IDE Drupal-tukineen osoittautui hyväksi työkaluksi, jolla koodin kirjoittaminen oli nopeaa ja valtaosa virheellisestä koodista huomattiin jo heti kirjoitusvaiheessa. Xdebug-työkalu oli viritetty käyttövalmiuteen PHPStormissa, mutta sitä ei käytetty, koska koodin määrä oli sen verran vähäinen, että bugit tuntuivat löytyvän koodista helposti muutenkin. Version- ja konfiguraationhallinnan käyttö ei yksin työskennellessä ja vain paikallista kehitysympäristöä käyttäen ehkä olisi ollut täysin välttämätöntä, mutta kyseessä on hyvä käytäntö, jota kannattaa noudattaa ja joka tekee jatkossa muiden kehittäjien osallistumisen projektiin ja järjestelmän tuotantokäyttöön siirtymisen huomattavasti helpommaksi. Jos koodia olisi jostakin syystä tuhoutunut, sitä olisi saatu nopeasti palautettua versionhallinnasta.

Mikäli projektia jatketaan opinnäytetyön jälkeen, kannattaisi seuraavaksi lähteä ideoimaan pidemmälle maksuvälitysjärjestelmän palvelukonseptia ja konkretisoimaan ideat käyttäjätarinoiksi. Johdantoluvussa olikin jo lueteltu pitkä lista ideoita, jotka rajattiin opinnäytetyön ulkopuolelle. Opinnäytetyötä tehdessä ha-

vaittiin, että maksuvälitysjärjestelmän API ei välttämättä tulisi olemaan kovin triviaali integrointitehtävä joillekin kehittäjille. Tästä syystä maksuvälitysjärjestelmälle kannattaisi kehittää SDK-ohjelmistokehityspaketteja usealle eri alustalle – tärkeimpinä mahdollisesti Android- ja Apple iOS -mobiilialustat. Moduuleita ja plugineita Mobiilimaksun integroimiseksi voitaisiin kehittää muutamalle mobiilikäyttöön hyvin mukautuvalle verkkokauppa-alustalle. Myös Drupal 8:n soveltuvuus alustaksi kannattaisi mahdollisesti arvioida vielä uudestaan, sillä arkkitehtuurin kannalta hyvä ratkaisu olisi, että API:a pyörittävä back-end ja API:n taloushallinnollisen puolen käyttöliittymä voisivat toimia vähintäänkin erillisissä Drupal-instansseissa.

Uskon, että web-kehittäjien on hedelmällistä pyrkiä tarkastelemaan Drupalia erityisesti sisällönhallintakehyksenä (CMF), koska tämä näkökulma auttaa parhaiten ymmärtämään Drupalin mahdollisuudet web-sovellusten kehitysalustana. Kun sisällönhallintajärjestelmä (CMS) tarjoaa valmiita ratkaisuja websivujen ja -palveluiden toteuttamiseen, kehys tarjoaa pohjan omien komponenttien kehittämiseen ja oman tietojärjestelmän rakentamiseen. Tässä yhteydessä sisältö-käsitettä kannattaa tarkastella laajassa merkityksessä: sen ei tarvitse tarkoittaa vain web-sivujen sisältöä, vaan ”sisältö” voi olla lähes mitä tahansa informaatiota, jolla on rakenne ja jota on tarpeen hallita, varastoida ja esittää. Ainakin itselleni tämä opinnäytetyö vahvisti sen, että Drupal 8 on erinomainen tietojärjestelmä, joka taipuu moneen käyttöön – myös vähän erikoisempienkin web-sovellusten alustaksi.

Lähteet

Ambler, S. 2002. Agile modeling. Effective Practices for eXtreme Programming and the Unified Process. New York, USA: John Wiley & Sons.

Ambler, S. 2016a. Effective Practices for Modeling and Documentation. Agile Modeling (AM) homepage. http://www.agilemodeling.com/ 15.11.2016.

Ambler, S. 2016b. Document Late: An Agile Best Practice. Agile Modeling (AM) Home Page. http://agilemodeling.com/essays/documentLate.htm 28.9.2016.

Ambler, S. 2016c. Document Continuously: An Agile Best Practice. Agile Modeling (AM) Home Page. http://agilemodeling.com/essays/documentContinuously.htm 28.9.2016.

Ambler, S. 2017. An Introduction to Agile Modeling. Agile Modeling (AM) homepage. http://www.agilemodeling.com/essays/introductionToAM.htm 22.2.2017.

Anderson, G. 2015. Configuration Workflow for Drupal 8 Sites. Patheon blog. 4.6.2015. https://pantheon.io/blog/configuration-workflow-drupal-8-sites 23.11.2016.

API University. 2016. What Are APIs and How Do They Work? API U Series. http://www.programmableweb.com/api-university/what-are-apis-and-how-do-they-work 15.11.2016.

Beal, V. 2016. Content management system. Webopedia. http://www.webopedia.com/TERM/C/content_management_system.html 15.11.2016.

Beal, V. 2017. MSISDN - Mobile Station International Subscriber Directory Number. http://www.webopedia.com/TERM/D/debug.html 21.2.2017.

Beams, C. 2014. How to Write a Git Commit Message. https://chris.beams.io/posts/git-commit/ 1.3.2017.

Beck, K., Beedle, M., Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A.,Jeffries R., Kern J., Marick B., Martin R., Mellor S., Schwaber K., Sutherland J. & Thomas D. Manifesto for Agile Software Development. 2001. http://agilemanifesto.org/ 28.9.2016.

Benet, J. 2014. A simple git branching model. GitHub Gist. 18.9.2014. https://gist.github.com/jbenet/ee6c9ac48068889b0912 1.3.2017.

Black Duck Open Hub. 2016. Drupal (core). https://www.openhub.net/p/drupal/analyses/latest/languages_summary 23.5.2016.

Bloch, J. 2007. How To Design A Good API and Why it Matters. GoogleTechTalks. 24.1.2007. https://www.youtube.com/watch?v=aAb7hSCtvGw 30.1.2017.

Borchert, S. & Schirwinski A. 2015. Drupal 8 Configuration Management. Birmingham, Iso-Britannia: Packt Publishing.

Bosch, M. A. 2016. Injecting services in your D8 plugins. Lullabot. 30.3.2016. https://www.lullabot.com/articles/injecting-services-in-your-d8-plugins 2.2.2017.

Brail G., Jacobson D. & Woods D. 2012. APIs: A Strategy Guide. Sebastopol, USA: O'Reilly Media.

BuildAModule. 2016a. How the dependency injection container acts like a backbone to Drupal 8. Drupal 8 Developer Prep. https://buildamodule.com/video/drupal-8-developer-prep-how-the-dependency-injection-container-works-in-drupal-8-how-the-dependency-injection-container-acts-like-a-backbone-to-drupal-8 18.11.2016.

Bujisic, B. 2016. Drupal 8: Pursue for Best Practices. DrupalCon Dublin 2016. Seminaaritaltiointi 27.9.2016. https://www.youtube.com/watch?v=LE7b-N12Qs4 25.2.2017.

Burge, S. 2014. What is Headless Drupal? The OSTraining Blog. 26.8.2014. https://www.ostraining.com/blog/drupal/what-is-headless-drupal/ 21.9.2016.

Buytaert, D. 2016a. How should you decouple Drupal? Dries Buytaert blog. 22.3.2016. http://buytaert.net/how-should-you-decouple-drupal 22.9.2016.

Buytaert, D. 2016b. A roadmap for making Drupal more API-first. Drupal.org blog. 7.7.2016. https://www.drupal.org/blog/roadmap-making-drupal-more-api-first 14.1.2017.

Byron, A., McGuire, J. 2016. The Ultimate Guide To Drupal 8. Revised and updated to Drupal 8.1. Acquia. https://www.acquia.com/resources/ebooks/ultimate-guide-drupal-8 17.9.2016.

Career Centre. 2016. Developer programmer. Occupations. Government of West Australia. Department of Training and Workforce Development. http://www.careercentre.dtwd.wa.gov.au/Occupations/Pages/developer-programmer.aspx 15.11.2016.

Cgit.drupalcode.org. 2016a. Drupal 8.2.x. Path: root/core/lib/Drupal/Component. Drupal code repository. http://cgit.drupalcode.org/drupal/tree/core/lib/Drupal/Component 19.11.2016.

Cgit.drupalcode.org. 2016b. Drupal 8.2.x. Path: root/core/lib/Drupal/Core. Drupal code repository. http://cgit.drupalcode.org/drupal/tree/core/lib/Drupal/Core 19.11.2016.

Cgit.drupalcode.org. 2016c. Drupal 8.2.x. Path: root/core/includes. Drupal code repository. http://cgit.drupalcode.org/drupal/tree/core/includes 19.11.2016.

Cgit.drupalcode.org. 2017. Drupal 8.2.x. Path: root/core/modules. Drupal code repository. http://cgit.drupalcode.org/drupal/tree/core/modules 20.2.2017.

Chromium Projects. 2017. Chhromium. https://www.chromium.org/Home 25.2.2017.

Cipix. 2013. Understanding Drupal 8, part 1: The general structure of the framework. Blog. 14.11.2013. https://cipix.nl/understanding-drupal-8-part-1-general-structure-framework 18.11.2016.

Cobb, C. 2015. The Project Manager's Guide to Mastering Agile. Principles and Practices for Adaptive Approach. New Jersey, USA: John Wiley & Sons.

Cohn, A. 2014. What is decoupling and what development areas can it apply to? Software Engineering. 9.6.2014. http://softwareengineering.stackexchange.com/a/244478 19.11.2016.

ComputerHope. 2017. SIM Card. http://www.computerhope.com/jargon/s/simcard.htm 21.2.2017.

Dia. 2017. Dia Diagram Editor. http://dia-installer.de/ 25.2.2017.

DNA. 2017a. Mikä on Mobiilimaksu? DNA:n tukisivusto. https://tuki.dna.fi/org/dna-fi/d/mika-on-mobiilimaksu/ 7.1.2017.

DNA. 2017b. Mobiilimaksu (IP-palveluviestit). DNA:n tukisivusto. https://tuki.dna.fi/org/dna-fi/d/ip-palveluviestit-1/ 7.1.2017.

Drupal.com. 2017. Content as a Service. http://drupal.com/feature/content-as-a-service 27.1.2017.

Drupal.org. 2015aa. Consider using Doctrine ORM for Entities. Drupal core issues. https://www.drupal.org/node/1817778 8.2.2017.

Drupal.org. 2016a. About. https://www.drupal.org/about 23.5.2016.

Drupal.org. 2016b. Understanding Drupal. Overview. https://www.drupal.org/node/265726 21.9.2016.

Drupal.org. 2016c. RESTful Web Services module overview. 9.11.2016 https://www.drupal.org/docs/8/core/modules/rest/overview 27.1.2017.

Drupal.org. 2016d. Our history. https://www.drupal.org/about/history 23.5.2016.

Drupal.org. 2016e. CHANGELOG.txt. 8.0.x. https://api.drupal.org/api/drupal/core!CHANGELOG.txt/8.0.x 23.5.2016.

Drupal.org. 2016f. Usage statistics for Drupal core. https://www.drupal.org/project/usage/drupal 23.5.2016.

Drupal.org. 2016g. Mission and principles. https://www.drupal.org/about/mission-and-principles 17.9.2016.

Drupal.org. 2016h. Technology stack. https://www.drupal.org/node/176052 23.5.2016.

Drupal.org. 2016i. System requirements. https://www.drupal.org/requirements 23.5.2016.

Drupal.org. 2016j. Basic structure of Drupal 8. Drupal 8 APIs. https://www.drupal.org/docs/8/api/basic-structure-of-drupal-8 18.11.2016.

Drupal.org. 2016k. Class DrupalKernel. Drupal API reference. https://api.drupal.org/api/drupal/core!lib!Drupal!Core!DrupalKernel.php/class/DrupalKernel/8.2.x 18.11.2016.

Drupal.org. 2016l. Services and dependency injection in Drupal 8. Drupal 8 APIs. https://www.drupal.org/docs/8/api/services-and-dependency-injection/services-and-dependency-injection-in-drupal-8 18.11.2016.

Drupal.org. 2016m. Altering existing services, providing dynamic services. Drupal 8 API. https://www.drupal.org/docs/8/api/services-and-dependency-injection/altering-existing-services-providing-dynamic-services 20.11.2016.

Drupal.org. 2016n. Use Symfony EventDispatcher for hook system. Download & Extend. https://www.drupal.org/node/1509164 19.11.2016.

Drupal.org. 2016o. Class Drupal. Drupal API. https://api.drupal.org/api/drupal/core%21lib%21Drupal.php/class/Drupal/8.2.x 22.11.2016.

Drupal.org. 2016p. Events. Drupal API. https://api.drupal.org/api/drupal/core%21core.api.php/group/events/8.2.x 23.11.2016.

Drupal.org. 2016q. Hook Event Dispatcher. Download & Extend. https://www.drupal.org/project/hook_event_dispatcher 19.11.2016.

Drupal.org. 2016r. Overview of Configuration (vs. other types of information). Drupal 8 API. https://www.drupal.org/docs/8/api/configuration-api/overview-of-configuration-vs-other-types-of-information 20.11.2016.

Drupal.org. 2016s. Configuration API overview. Drupal 8 API. https://www.drupal.org/docs/8/api/configuration-api/configuration-api-overview 21.11.2016.

Drupal.org. 2016t. Simple Configuration API. Drupal 8 API. https://www.drupal.org/docs/8/api/configuration-api/simple-configuration-api 23.11.2016.

Drupal.org. 2016u. Configuration Storage in Drupal 8. Drupal 8 API. https://www.drupal.org/docs/8/api/configuration-api/configuration-storage-in-drupal-8 23.11.2016.

Drupal.org. 2016v. Entity API. https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/group/entity_api/8.2.x 24.11.2016.

Drupal.org. 2016w. Introduction to Entity API in Drupal 8. Drupal 8 API. https://www.drupal.org/docs/8/api/entity-api/introduction-to-entity-api-in-drupal-8 24.11.2016.

Drupal.org. 2016x. Configuration schema/metadata. https://www.drupal.org/node/1905070 24.11.2016 .

Drupal.org. 2016y. Configuration Entity. Drupal 8 API. https://www.drupal.org/docs/8/api/entity-api/configuration-entity 24.11.2016.

Drupal.org. 2016z. Entity CRUD, editing, and view hooks. https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/group/entity_crud/8.2.x 24.11.2016.

Drupal.org. 2016å. Typed Data API overview. https://www.drupal.org/node/1794140 24.11.2016.

Drupal.org. 2016ä. Entity Validation API overview. https://www.drupal.org/node/2015613 24.11.2016.

Drupal.org. 2016ö. Bundles. Drupal 8 API. https://www.drupal.org/docs/8/api/entity-api/bundles 24.11.2016.

Drupal.org. 2016aa. Content Entity. Drupal 8 API. https://www.drupal.org/docs/8/api/entity-api/content-entity 24.11.2016.

Drupal.org. 2016ab. Plugin API. Drupal API. https://api.drupal.org/api/drupal/core!core.api.php/group/plugin_api/8.2.x 20.11.2016.

Drupal.org. 2016ac. Plugin API overview. Drupal 8 APIs. https://www.drupal.org/docs/8/api/plugin-api/plugin-api-overview 20.11.2016.

Drupal.org. 2016ad. Render arrays. https://www.drupal.org/docs/8/api/render-api/render-arrays 16.11.2016.

Drupal.org. 2016ae. Render API overview. https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/theme_render/8.2.x 16.11.2016.

Drupal.org. 2016af. Render API. Drupal 8 API. https://www.drupal.org/docs/8/api/render-api 16.11.2016.

Drupal.org. 2016ag. The Drupal 8 render pipeline. Drupal 8 API. https://www.drupal.org/docs/8/api/render-api/the-drupal-8-render-pipeline 16.11.2016.

Drupal.org. 2016ah. Namespace Drupal\Core\Render\Element. Drupal API. https://api.drupal.org/api/drupal/namespace/Drupal%21Core%21Render%21Element/8.2.x 17.11.2016.

Drupal.org. 2016ai. Routing API. Drupal API. https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Routing!routing.api.php/group/routing/8.2.x 23.11.2016.

Drupal.org. 2016aj. Structure of routes. Drupal 8 APIs. https://www.drupal.org/node/2092643 23.11.2016.

Drupal.org. 2016ak. Ajax API. Drupal API reference. https://api.drupal.org/api/drupal/core%21core.api.php/group/ajax/8.2.x 18.11.2016.

Drupal.org. 2016al. Form API. Drupal 8 API. https://www.drupal.org/docs/8/api/form-api 20.11.2016.

Drupal.org. 2016am. Form and render elements. Drupal API. https://api.drupal.org/api/drupal/elements/8.2.x 20.11.2016.

Drupal.org. 2016an. Marketplace. https://www.drupal.org/drupal-services 27.9.2016.

Drupal.org. 2016ao. Block API. Drupal 8 API. https://www.drupal.org/docs/8/api/block-api 19.11.2016.

Drupal.org. 2016ap. Block API. https://api.drupal.org/api/drupal/core%21modules%21block%21block.api.php/group/block_api/8.2.x 19.11.2016.

Drupal.org. 2016aq. Drupal core maintainers. Getting Involved Guide. https://www.drupal.org/contribute/core/maintainers 27.9.2016.

Drupal.org. 2017a. Glossary. Drupal 7 documentation. https://www.drupal.org/docs/7/understanding-drupal/glossary 21.2.2017.

Drupal.org. 2017b. Coding standards. Drupal 8 documentation. https://www.drupal.org/docs/develop/standards/coding-standards 20.2.2017.

Drupal.org. 2017c. Service Tags. Drupal API. https://api.drupal.org/api/drupal/core%21core.api.php/group/service_tag/8.2.x 1.3.2017.

Drupal.org. 2017c. FieldTypes, FieldWidgets and FieldFormatters. Drupal 8 Apis. https://www.drupal.org/docs/8/api/entity-api/fieldtypes-fieldwidgets-and-fieldformatters 20.2.2017.

Drupal.org. 2017e. Field API. https://api.drupal.org/api/drupal/core%21modules%21field%21field.module/group/field/8.2.x 20.2.2017.

Drupal.org. 2017f. Annotations. Drupal API. https://api.drupal.org/api/drupal/core!core.api.php/group/annotation/8.2.x 20.2.2017.

Drupal.org. 2017g. Cache API. Drupal 8 APIs. https://www.drupal.org/node/1884796 20.2.2017.

Drupal.org. 2017h. Logging API. Drupal 8 APIs. https://www.drupal.org/docs/8/api/logging-api/overview 20.2.2017.

Drupal.org. 2017i. State API. Drupal 8 APIs. https://www.drupal.org/docs/8/api/state-api/overview 20.2.2017.

Drupal.org. 2017j. Translation API. Drupal 8 APIs. https://www.drupal.org/docs/8/api/translation-api/overview 20.2.2017.

Drupal.org. 2017k. Modules. Download & Extend. https://www.drupal.org/project/project_module 20.2.2017.

Drupal.org. 2017l. About nodes. Drupal 7 documentation. https://www.drupal.org/docs/7/nodes-content-types-and-fields/about-nodes 20.2.2017.

Drupal.org. 2017m. Core modules. Drupal 8 documentation. https://www.drupal.org/docs/8/core/modules 20.2.2017.

Drupal.org. 2017n. HTTP Basic Authentication overview. Drupal 8 documentation. https://www.drupal.org/docs/8/core/modules/basic_auth/overview 21.2.2017.

Drupal.org. 2017o. RESTful Web Services API overview. Drupal 8 APIs. https://www.drupal.org/docs/8/api/restful-web-services-api/restful-web-services-api-overview 21.2.2017.

Drupal.org. 2017p. Serialization API overview. Drupal 8 APIs. https://www.drupal.org/docs/8/api/serialization-api/serialization-api-overview 21.2.2017.

Drupal.org. 2017q. Views overview. Drupal 8 documentation. https://www.drupal.org/docs/8/core/modules/views/overview 21.2.2017.

Drupal.org. 2017r. User module overview. Drupal 8 documentation. https://www.drupal.org/docs/8/core/modules/user/overview 21.2.2017.

Drupal.org. 2017s. Features. Download & Extend. https://www.drupal.org/project/features 21.2.2017.

Drupal.org. 2017t. What's new in Features for Drupal 8? Drupal 8 documentation. https://www.drupal.org/docs/8/modules/features/whats-new-in-features-for-drupal-8 21.2.2017.

Drupal.org. 2017u. Themes. Download & Extend. https://www.drupal.org/project/project_theme 20.2.2017.

Drupal.org. 2017v. Core themes. Drupal 8 documentation. https://www.drupal.org/docs/8/core/themes 20.2.2017.

Drupal.org. 2017w. Stable theme. Drupal 8 documentation. https://www.drupal.org/docs/8/core/themes/stable-theme 20.2.2017.

Drupal.org. 2017x. Comparison of PHPTemplate and Twig theming paradigms. Drupal 8 documentation. https://www.drupal.org/docs/8/theming/twig/comparison-of-phptemplate-and-twig-theming-paradigms 20.2.2017.

Drupal.org. 2017y. Theming Drupal 8. Drupal 8 documentation. https://www.drupal.org/docs/8/theming 20.2.2017.

Drupal.org. 2017z. Adding stylesheets (CSS) and JavaScript (JS) to a Drupal 8 theme. Drupal 8 documentation. https://www.drupal.org/docs/8/theming-drupal-8/adding-stylesheets-css-and-javascript-js-to-a-drupal-8-theme 20.2.2017.

Drupal.org. 2017å. Managing your site's configuration. Drupal 8 documentation. www.drupal.org/docs/8/configuration-management/managing-your-sites-configuration 1.3.2017.

Drupal.org. 2017ä. HAL module overview. Drupal 8 documentation. https://www.drupal.org/docs/8/core/modules/hal/overview 24.2.2017.

Drupal.org. 2017ö. Disable Drupal 8 caching during development. https://www.drupal.org/node/2598914 25.2.2017.

Drupal.org. 2017aa. Installing Modules. Extending Drupal 8. https://www.drupal.org/docs/8/extending-drupal-8/installing-modules 25.1.2017.

Drupal.org. 2017ab. Annotations-based plugins. Plugin API. https://www.drupal.org/docs/8/api/plugin-api/annotations-based-plugins 25.1.2017.

Drupal.org. 2017ac. Authentication API. Drupal 8 APIs. https://www.drupal.org/docs/8/api/authentication-api/overview 26.2.2017.

Drupal.org. 2017ad. Introduction to Form API. Drupal 8 APIs. https://www.drupal.org/docs/8/api/form-api/introduction-to-form-api 19.2.2017.

Drupal.org. 2017ae. Adding stylesheets (CSS) and JavaScript (JS) to a Drupal 8 module. Creating custom modules. https://www.drupal.org/docs/8/creating-custom-modules/adding-stylesheets-css-and-javascript-js-to-a-drupal-8-module 19.2.2017.

Drupal Association. 2015. State of the Drupal Job Market. https://assoc.drupal.org/drupal-job-market-2015-still-red-hot 14.1.2017.

Drupal Console. 2016. Container:debug. Drupal Console Documentation. https://hechoendrupal.gitbooks.io/drupal-console/content/en/commands/container-debug.html 22.11.2016.

Drupal Console. 2017. Drupal Console. The Drupal CLI. A tool to generate boilerplate code, interact with and debug Drupal. https://drupalconsole.com/ 25.2.2017.

Drush. 2017. Drush docs. http://www.drush.org 25.2.2017.

Eclipse Foundation. 2017. Papyrus. https://eclipse.org/papyrus/ 25.2.2017.

ECMA. 2013. The JSON Data Interchange Format. ECMA-404 standard. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf 21.2.2017.

Elisa yritysasiakaspalvelu. 2015. Elisa Mobiilimaksu Direct Carrier Billing Service. Developers Guide, version 2.0. PDF-dokumentti, luottamuksellinen. Elisa Oyj 1.7.2015.

Fielding, R. 2000. Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm 21.2.2017.

Finanssivalvonta. 2014. Maksupalvelun tarjoajat. http://www.finanssivalvonta.fi/fi/Finanssiasiakas/Palveluntarjoajat/Maksupalvelu/Pages/Default.aspx 23.2.2017.

Finanssivalvonta. 2017. Määräykset ja ohjeet 8/2016. Maksulaitokset ja maksupalvelua ilman toimilupaa tarjoavat henkilöt. http://www.finanssivalvonta.fi/fi/Saantely/Maarayskokoelma/Uusi/Documents/2016_08.M1.pdf 23.2.2017.

Finlex. 2017. Maksulaitoslaki 30.4.2010/297. http://www.finlex.fi/fi/laki/ajantasa/2010/20100297 23.2.2017.

Garcia, E. 2014. How to create a Rest Resource in Drupal 8. Blog 16.12.2014. http://enzolutions.com/articles/2014/12/16/how-to-create-a-rest-resource-in-drupal-8/ 26.2.2017.

Garfield, L. 2009. ORMs vs. Query Builders: Database portability. 1.7.2009. http://www.garfieldtech.com/blog/orm-vs-query-builders 8.2.2017.

Garfield, L. 2015. D8FTW: Storing Data in Drupal 8, The Palantir Blog. 11.8.2015. https://www.palantir.net/blog/d8ftw-storing-data-drupal-8 28.1.2017.

Google. 2017. Google Docs. https://www.google.com/intl/en_us/docs/about/ 25.2.2017.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum.

Hakimzadeh, D., Melançon B. ja Nordin D. 2011. There's a Module for That. Teoksessa Melançon B. (toim.). The Definitive Guide to Drupal 7. New York, USA: Apress, 87-108.

IETF. 1997. HTTP Authentication: Basic and Digest Access Authentication. Request for Comments: 2069. https://tools.ietf.org/html/rfc2069 21.1.2017.

Indiana University. 2014. What is a URL? Knowledge Base. https://kb.iu.edu/d/adnz 21.2.2017.

IETF. 1999. Hypertext Transfer Protocol HTTP/1.1. RFC 2616. https://www.ietf.org/rfc/rfc2616.txt 21.2.2017.

IETF. 2012. The OAuth 2.0 Authorization Framework. Request for Comments: 6749. https://tools.ietf.org/html/rfc6749 21.1.2017.

Juutinen, J. 2016. Mobiilimaksaminen – kuluttajanoikeuksien turvaamisen näkökulma. Viestintäviraston yhteenveto-puheenvuoro Viestintäviraston numerointityöryhmän teematilaisuudessa "Mobiilimaksaminen - kuluttajan oikeuksien turvaamisen näkökulma" 10.2.2016. Esitysmateriaali. https://www.viestintavirasto.fi/attachments/esitykset/Mobiilimaksu_Mobiilimaksunkuluttajansuoja100216.pdf 10.1.2017.

Järvinen, T. 2016a. Skype-keskustelu. Juhani Pirinen. Skype. 20.10.2016.

Järvinen, T. 2016b. Mobiilimaksut / DNA. Juhani.Pirinen@edu.karelia.fi. Sähköpostiviesti. 21.10.2016.

KADOS. 2017. Kanban Dashboard for Online Scrum. http://www.kados.info/ 25.2.2017.

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Koskelainen, M. 2016. Opinnäytetyö: Soneran Mobile Charger -palvelu hyödyntävä asiakas-palvelin-sovellus Mediayhtiö B105 Oy:lle. Juhani.Pirinen@edu.karelia.fi. Sähköpostiviesti. 1.11.2016.

Letharion. 2015. What are all the directories for in the new Drupal 8 structure? Drupal Answers. 11.11.2015. http://drupal.stackexchange.com/a/84851 18.11.2016.

Lorétan, F. 2011. Writing Project-Specific Code. Teoksessa Melançon B. (toim.). The Definitive Guide to Drupal 7. New York, USA: Apress, 501-516.

Lullabot. 2016. "One Weird Trick" for Drupal Security... or Something. Lullabot Podcast, Episode 185. 21.4.2016. https://www.lullabot.com/podcasts/drupalizeme-podcast/one-weird-trick-for-drupal-security-or-something 14.1.2017.

Mantere, T. 2014. Wikikirja kurssin aiheista. Organisaation tietojärjestelmät TITE2060. Vaasan Yliopisto. http://www.uva.fi/~timan/TITE2060/OT2014_wiki.pdf 21.9.2016.

MAPEL. 2017. 25§ Mobiilimaksu. Erityiset säännöt. Eettiset säännöt. Teleforum ry / Maksullisten puhelinpalveluiden eettinen lautakunta. https://www.mapel.fi/eettiset_saannot/erityiset_saannot/ 7.1.2017.

Nazzaro, W. & Suscheck, C. 2010. New to User Stories? Member Articles. Scrum Alliance. 19.4.2010. https://www.scrumalliance.org/community/articles/2010/april/new-to-user-stories 10.1.2017.

Norton, P. 2016. Becoming A Drupal Master Builder. Drupal Camp London 2016. Seminaaritaltiointi 20.6.2016. https://www.youtube.com/watch?v=O1UY2PyvXRA 1.3.2017.

Object Management Group. 2005. Introduction to OMG's Unified Modeling Language™ (UML®). http://www.uml.org/what-is-uml.htm 22.2.2017.

Octal Info Solution. 2016. How Front-end and Back-end Web Development make a difference? Blog. 18.5.2016. http://www.octalsoftware.co.uk/blog/how-front-end-and-back-end-web-development-make-a-difference/ 15.11.2016.

Ojajärvi, M. 2016. Mobiilimaksu ja kuluttajansuoja. Kilpailu- ja kuluttajaviraston puheenvuoro Viestintäviraston numerointityöryhmän teematilaisuudessa "Mobiilimaksaminen - kuluttajan oikeuksien turvaamisen näkökulma" 10.2.2016. Esitysmateriaali. https://www.viestintavirasto.fi/attachments/esitykset/Mobiilimaksu_Mobiilimaksunkuluttajansuoja100216.pdf 10.1.2017.

OWASP. 2016. Cross-Site Request Forgery (CSRF). 2.11.2016. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) 21.1.2017.

Pantheon. 2017. Decoupled CMS: Why “Going Headless” Is Becoming So Popular. https://pantheon.io/decoupled-cms 27.1.2017.

Pathirage, G. 2016. What are the pros and cons of git-flow vs github-flow? Stack Overflow. 10.3.2016. http://stackoverflow.com/a/35915110 1.3.2017

PHP-DI. 2016. Understanding Dependency Injection. http://php-di.org/doc/understanding-di.html 19.11.2016.

PHP.net Wiki. 2014. Request for Comments: Class Metadata. 17.2.2014. https://wiki.php.net/rfc/annotations 8.2.2017.

PhpDocumentor. 2017. Basic Syntax. https://www.phpdoc.org/docs/latest/references/phpdoc/basic-syntax.html 8.2.2017.

Pichler, M. 2017. Code Size Rules. PHP Mess Detector documentation. 8.1.2017. https://phpmd.org/rules/codesize.html 28.2.2017.

Postdot Technologies. 2017. Postman. https://www.getpostman.com/ 25.2.2017.

Pressman, R. 2010. Software engineering : a practitioner's approach. New York, USA: McGraw-Hill.

Rouse, M. 2007. Waterfall model. Definition. http://searchsoftwarequality.techtarget.com/definition/waterfall-model 22.2.2017.

Scavarda, A. 2011. Planning and Managing a Drupal Project. Teoksessa Melançon B. (toim.). The Definitive Guide to Drupal 7. New York, USA: Apress, 203-220.

Scott. 2012. PHP: How to generate a random, unique, alphanumeric string? Stack Overflow. 5.12.2012. http://stackoverflow.com/questions/1846202/php-how-to-generate-a-random-unique-alphanumeric-string/13733588#13733588 1.2.2017.

Scrum.org. 2017. What is Scrum? https://www.scrum.org/resources/what-is-scrum 1.3.2017.

Segue Technologies. 2013. What is Ajax and Where is it Used in Technology? Blog. 12.3.2013. http://www.seguetech.com/ajax-technology/ 19.11.2016.

Shindelar, J. 2014. Unraveling the Drupal 8 Plugin System. Drupalize.me blog 9.9.2014. https://drupalize.me/blog/201409/unravelling-drupal-8-plugin-system 20.11.2016.

Shindelar, J. 2016. Why Is Learning Drupal Hard? Drupalize.org blog. 21.7.2016. https://drupalize.me/blog/201607/why-learning-drupal-hard 14.1.2017.

Siriwardena, P. 2014. Advanced API Security. Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE. New York, USA: Apress.

Software Freedom Conservancy. 2017. Git. https://git-scm.com/ 21.2.2017.

Sonera. 2016a. MSISDN Enrichment. Sonera Service Provision Solution, Appendix 3.2.1. PDF-dokumentti. Sonera Oyj 27.10.2016.

Sonera. 2016b. OMA Payment REST API Guide, revision 5.1. 23.11.2016. https://developer.sonera.fi/f/files/resources/API_Sonera_OMA_Payment_REST_5.1 3.1.2017.

Sonera. 2016c. OMA Authorization REST API Guide, revision 5.0. 16.6.2016. https://developer.sonera.fi/f/files/resources/API_Sonera_OMA_OAuth_REST_5.0 12.2.2017.

Sonera. 2017. API_Sonera_OMA_Sandbox_REST_8.0. https://developer.sonera.fi/f/files/resources/API_Sonera_OMA_Sandbox_REST_8.0 25.2.2017.

Suomen virtuaaliyliopisto. 2017. Tutkiva oppiminen. Oppimisen teoriasta tukea tieto- ja viestintätekniikan pedagogiseen käyttöön. http://tievie.oulu.fi/verkkopedagogiikka/luku_6/tutkiva_oppiminen.htm 2.3.2017.

Symfony.com. 2016a. Why should I use a framework? Symfony in 5 minutes. http://symfony.com/why-use-a-framework 21.9.2016.

Symfony.com. 2016b. The DependencyInjection Component. http://symfony.com/doc/current/components/dependency_injection.html 21.9.2016.

Symfony.com. 2016c. Drupal. Projects using Symfony. http://symfony.com/projects/drupal 18.9.2016.

Symfony.com. 2016d. Service Container. http://symfony.com/doc/current/service_container.html 20.11.2016.

Symfony.com. 2016e. Events and Event Listeners. http://symfony.com/doc/current/event_dispatcher.html 22.11.2016.

Symfony.com. 2016f. The HttpFoundation Component. https://symfony.com/doc/current/components/http_foundation/index.html 23.11.2016.

Symfony.com. 2016g. The HttpKernel Component. http://symfony.com/doc/current/components/http_kernel.html 23.11.2016.

Symfony.com. 2016h. The Routing Component. http://symfony.com/doc/current/components/routing.html 24.11.2016.

Symfony.com. 2017. Types of Injection. http://symfony.com/doc/current/service_container/injection_types.html 2.2.2017.

Symfony CMF. 2016. The Symfony CMF Project. http://cmf.symfony.com/ 15.11.2016.

Tamminen, T. 2016. It Is Not Just About Having The Right Tool for The Job. Blog 25.6.2016. https://www.triplet.fi/blog/it-is-not-just-about-having-the-right-tool-for-the-job/ 14.1.2017.

Techopedia. 2016. Application Framework. https://www.techopedia.com/definition/6005/application-framework 21.9.2016.

Techopedia. 2017a. Legacy Code. https://www.techopedia.com/definition/25326/legacy-code 21.2.2017.

Techopedia. 2017b. Boilerplate. https://www.techopedia.com/definition/1259/boilerplate 21.2.2017.

TechTerms. 2017a. IP Address. https://techterms.com/definition/ip_address 21.2.2017.

TechTerms. 2017b. Framework. https://techterms.com/definition/framework 21.2.2017.

TechTerms. 2017c. URI. https://techterms.com/definition/framework 1.3.2017.

Teleforum. 2014. IP-laskutus tunnetaan jatkossa nimellä Mobiilimaksu. Blogi. 12.12.2014. https://www.teleforum-ry.fi/uncategorized/ip-laskutus-tunnetaan-jatkossa-nimella-mobiilimaksu/ 31.12.2016.

Teleforum. 2016a. Mobiilimaksu. http://mobiilimaksuinfo.fi/ 31.12.2016.

Teleforum. 2016b. Mobiilimaksun logo, maksupainike ja graafinen ohjeisto. http://mobiilimaksuinfo.fi/mobiilimaksu-visual-assets.zip 31.12.2016.

Teleforum. 2016c. Uudet pelisäännöt Mobiilimaksulle. 14.1.2016. https://www.teleforum-ry.fi/itsesaantely/uudet-pelisaannot-mobiilimaksulle/ 10.2.2017.

Tieto Connection / VAS Center. 2014. Tieto Connection IP Connectivity Service description. Charging API 1.2.2, HTTP Interface. PDF-dokumentti. Helsinki: Tieto Finland Oy.

Tift, M. 2016. Why Paid Drupal Modules Fail: Drupal as Art. Lullabot articles. 7.4.2016. https://www.lullabot.com/articles/why-paid-drupal-modules-fail-drupal-as-art 27.9.2016.

Tomlinson, T. 2015. Beginning Drupal 8. New York, USA: Apress.

Trautman, E. 2015. Why Learning to Code is So Damn Hard. Viking Code School Blog. 4.2.2015. https://www.vikingcodeschool.com/posts/why-learning-to-code-is-so-damn-hard 27.1.2017.

Tutorialspoint. 2017a. Object Oriented Paradigm. https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm 21.2.2017.

Tutorialspoint. 2017b. What is SOAP? https://www.tutorialspoint.com/soap/what_is_soap.htm 21.2.2017.

Tutorialspoint. 2017c. Design Pattern - Overview. https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm 21.2.2017.

Tutorialspoint. 2017d. RESTful Web Services – statelessness. https://www.tutorialspoint.com/restful/restful_statelessness.htm 25.2.2017.

Vaittinen, J. 2016. Mobiilimaksu. Matkaviestinoperaattorin puheenvuoro Viestintäviraston numerointityöryhmän teematilaisuudessa "Mobiilimaksaminen - kuluttajan oikeuksien turvaamisen näkökulma" 10.2.2016. Esitysmateriaali.

https://www.viestintavirasto.fi/attachments/esitykset/Mobiilimaksu_Vivimobiilmaksu.pdf 4.1.2017.

Veracode. 2017. What Is an Integrated Development Environment (IDE)? https://www.veracode.com/security/integrated-development-environments 21.2.2017.

Vierityspalkki. 2017. Työpaikat. http://vierityspalkki.fi/tyopaikat/ 14.1.2017.

Viestintävirasto. 2016a. Määräys liittymän estopalveluista 35 R/2016 M. 22.12.2016. https://www.viestintavirasto.fi/attachments/maaraykset/M_35_R_2016.pdf 7.1.2017.

Viestintävirasto. 2016b. Määräyksen 35 perustelut ja soveltaminen. Liittymän estopalveluista MPS 35 R/2016 M. 22.12.2016. https://www.viestintavirasto.fi/attachments/maaraykset/M_35_R_MPS.pdf 7.1.2017.

Vink, M. 2016. Drupal Development using PhpStorm. 30.9.2016. https://confluence.jetbrains.com/display/PhpStorm/Drupal+Development+using+PhpStorm 25.2.2017.

Waqa Studios. 2011. What is a CMS and do I need one? Blog. 14.10.2011. https://waqastudios.com/blog/what-is-a-cms-and-do-i-need-one/ 20.2.2017.

Webopedia. 2017a. Debug. http://www.webopedia.com/TERM/D/debug.html 21.2.2017.

Webopedia. 2017b. Web services. http://www.webopedia.com/TERM/W/Web_Services.html 21.2.2017.

Wells, D. 2013. Extreme Programming: A gentle introduction. http://www.extremeprogramming.org/ 15.11.2016.

Wiktionary. 2016. Pre-alpha version. 30.10.2016. https://en.wiktionary.org/wiki/pre-alpha_version 14.1.2017.

Wolanin, P. 2016. Drupal 8, Where Did the Code Go? From Info Hook to Plugin. Acquia webinar. 14.7.2016. https://dev.acquia.com/events/webinars/drupal-8-where-did-code-go-info-hook-plugin 20.11.2016.

Wrike. 2017. What is Agile Methodology in Project Management? https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/ 21.2.2017.

Yaml.org. 2016. YAML Ain't Markup Language. http://yaml.org/ 18.9.2016.

Zamor, R. 2017. When To Use Content Types, Taxonomies, And Custom Entities In Drupal. 4site blog. http://www.4sitestudios.com/blog/when-to-use-content-types-taxonomies-and-custom-entities-in-drupal 3.2.2017.