Ideasta toteutukseen – opas ensimmäisen mobiilisovelluksen suunnitteluun

Mari Rautanen

Haaga-Helia ammattikorkeakoulu

Tradenomi

Opinnäytetyö

2024

Tiivistelmä

Tekijä
Mari Rautanen

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Ideasta toteutukseen – opas ensimmäisen mobiilisovelluksen suunnitteluun

Sivu- ja liitesivumäärä
23 + 28

Tämän opinnäytetyön aiheena oli oppaan tuottaminen ensimmäisen oman mobiilisovelluksen suunnittelua varten. Tavoitteena oli luoda niin selkeä ja helposti ymmärrettävä opas, että sen sisällön ymmärtäisivät myös sellaiset henkilöt, jotka eivät ole koskaan opiskelleet tietojenkäsittelyä. Aiheen valintaan vaikutti opinnäytetyöntekijän oma motivaatio, kiinnostus sekä muisto siitä, kuinka olisi itse aikanaan kaivannut vastaavaa opasta avuksi suunnitteluun.

Työtä rajattiin voimakkaasti, sillä ohjelmistosuunnittelu on aiheena hyvin laaja. Työ rajatiin koskemaan vain vaatimusmäärittelyn toiminnallisia vaatimuksia, ohjelmistojen mallintamista luokkakaavioiden kautta sekä käyttöliittymän suunnittelua. Myös saavutettavuus rajattiin opinnäytetyön ulkopuolelle. Rajauksia tehdessä kriteerinä käytettiin opinnäytetyöntekijän kokemusta siitä, mistä aiheista on ensimmäisen sovelluksen suunnittelussa ollut eniten hyötyä ja miten rakennetaan loogisesti etenevä ja ymmärrettävä kokonaisuus.

Opinnäytetyön tietoperusta koottiin ammattikirjallisuuden perusteella. Lähteitä valitessa pyrittiin hakemaan mahdollisimman tuoreita ja ajantasaisia lähteitä. Vanhempia lähteitä käytettäessä niiden tietoja verrattiin tuoreempien lähteiden sisältöön luotettavuuden varmistamiseksi.

Opinnäytetyön tuloksena syntyi 26-sivuinen opas, joka sisältää tekstin lisäksi useita selventäviä kuvia. Termejä ja aiheita avataan oppaassa käytännön esimerkkien kautta ja oppaan aikana valmistuukin suunnitelma yhtä mobiilisovellusta varten. Opas alkaa johdannolla, jossa lukijaa houkutellaan käyttämään aikaa suunnitteluun ja perustellaan sen tärkeyttä. Tämän jälkeen siirrytään vaatimusmäärittelyyn ja ohjelmiston mallintamiseen. Niiden jälkeen vuorossa on käyttöliittymän suunnittelu, jonka aikana valmistetaan prototyyppi suunnitelmien sovelluksesta. Lopuksi annetaan vielä hieman vinkkejä muun muassa teknologian valintaan ja työjärjestyksen pohdintaan sekä listataan tietopohjaan käytetyt lähteet.

Valmis tuotos luetutettiin kolmella testilukijalla, joilta saatiin sekä positiivista palautetta että rakentavaa kritiikkiä. Saatujen palautteiden perusteella oppaan kahteen päälukuun päädyttiin lisäämään käsitelistat selityksineen luvun loppuun. Positiivista palautetta saatiin erityisesti käytännön esimerkeistä ja oppaan visuaalisuudesta.

Opinnäytetyö jättää myös kehittämiskohteita tulevaisuuteen. Tuotettua opasta voisi jatkaa esimerkiksi teoksella tekoälyn hyödyntämisestä suunnittelussa ja saavutettavuus mobiilisovelluksissa ansaitsisi ehdottomasti oman oppaansa.

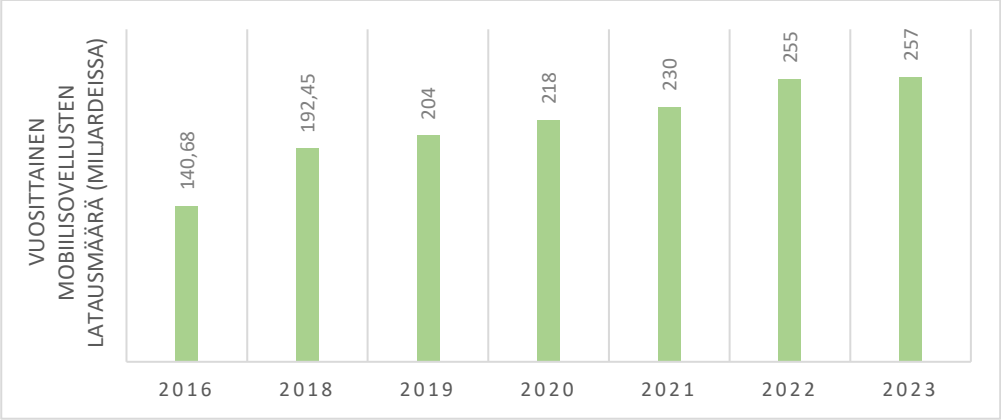
Asiasanat
Ohjelmistosuunnittelu, mobiilisovellus, vaatimusmäärittely, ohjelmiston mallintaminen

Sisällys

1 Johdanto

Historian ensimmäiset mobiilisovellukset ohjelmoitiin jo 1980-luvulla. Nuo sovellukset tosin olivat hyvin suppeita ja sisälsivät vain yksinkertaisia toimintoja. Mobiilisovellukset yleistyivät ja kehittyivät vasta 2000-luvulla älypuhelimien kehityksen myötä. Mobiilisovelluksista syntyi kokonaan uusi liiketoiminnan ala, kun puhelinvalmistajat sallivat käyttäjien ladata laitteilleen myös kolmansien osapuolten sovelluksia. (Volle 1.3.2024.)

Mobiilisovellusten suosio jatkaa kasvuaan vuodesta toiseen, eikä kehitykselle näy loppua (Statista 2024). Kuva 1 havainnollistaa kansainvälisten mobiilisovellusten latauskertojen kasvun vuodesta 2016 vuoteen 2023. Mobiilisovellusten suosion kasvaessa on selvää, että niiden toteuttaminen tulee työllistämään myös entistä enemmän ohjelmoijia. Sen vuoksi ohjelmoijaksi haluavien kannattaakin harjoitella myös mobiilisovellusten tuottamista ja suunnittelua.



Kuva 1. Mobiilisovellusten latausmäärät vuosilta 2016–2023 miljardeissa ilmoitettuna (mukaillen Statista 2024)

Tämän opinnäytetyön aiheena on opas ensimmäisen mobiilisovelluksen suunnitteluun. Opinnäytetyön tavoitteena on luoda helposti ymmärrettävä ja selkokielinen opas ohjelmointia harjoitteleville opiskelijoille ja harrastajille, jotka ovat vasta kehittämässä ensimmäistä omaa sovellustaan. Oppaan tarkoitus on ohjeistaa lukijaa suunnitteluprosesseista sekä niiden hyödyllisyydestä ohjelmistokehityksessä.

Mobiilisovellusten suunnittelu, kuten sovellusten suunnittelu yleensäkin, on laaja aihe, joten sitä ei käsitellä kaikenkattavasti tässä työssä. Koska oppaan tarkoitus on auttaa ensimmäisen sovelluksen suunnittelussa, keskitytään ohjeissa suunnitteluprosesseihin, jotka auttavat ohjelmoijaa käsittämään projektinsa vaatimukset ja luomaan yksinkertaiset tavoitteet, joita kohti kulkea. Opinnäytetyössä keskitytään lähinnä vaatimusmäärittelyyn ja käyttöliittymän hahmotteluun ennen ohjelmointia. Varsinaisen oppaan viimeisessä luvussa annetaan lukijalle vinkkejä siitä, mitä kannattaa vielä

pohtia suunnitelman teon jälkeen ennen kuin aloittaa itse sovelluksen ohjelmoinnin. Nämä lyhyet maininnat, esimerkiksi erilaisista teknologioista tai sisällönsuunnittelusta, eivät kuitenkaan ole varsinainen opinnäytetyön aihe, joten niitä käsitellään hyvin suppeasti.

Tuotettavassa oppaassa lähdetään liikkeelle ajatuksesta, että lukijalla on jo mielessään aihe omaan sovellukseen. Opinnäytetyön ulkopuolelle rajataan siis aiheen innovointiin liittyvät prosessit. Työstä rajataan pois myös sovelluksen kehittämiseen liittyvät tekijät, kuten teknologioiden opastaminen ja erilaiset ohjelmistokehityksen viitekehykset, testaaminen ja sen suunnittelu sekä sovelluksen julkaisu ja julkaisun suunnittelu. Ohjelmistojen mallintaminen rajataan koskemaan vain UML-kaavioita ja niistä erityisesti luokkakaavioita. Käyttöliittymän osalta opinnäytetyössä käsitellään lähinnä ulkoisia tekijöitä, käyttäjäkokemuksen pohtiminen rajataan työn ulkopuolelle. Myös saavutettavuus rajataan pois, sillä aiheena se on niin kattava, etteivät opinnäytetyön resurssit ole riittäviä sen käsittelyyn. Opinnäytetyössä keskitytään natiivi- ja hybridimobiiliapplikaatioihin, mobiilissa toimivia webapplikaatioita ei käsitellä.

Opinnäytetyössä tuotettu opas toteutettiin Visme-palvelun avulla. Vismeä hyödyntämällä luotiin graafinen opas, joka on helposti painettavissa myös fyysiseksi teokseksi. Opinnäytetyötä tehdessä on hyödynnetty myös Microsoft Wordia oikoluvun helpottamiseen ja lisäksi Canvan avulla on tuotettu osa käytetyistä kuvista. Opas on kirjoitettu pääsääntöisesti me-muotoon, millä on pyritty luomaan kuva helposti lähestyttävästä tekstistä. Valitut aiheet on esitetty käytännön esimerkkien avulla, jotta lukijan on helpompi ymmärtää kulloinkin käsiteltävä aihe tai termi.

Opinnäytetyön keskeisiä käsitteitä ovat:

- Käyttöliittymä: Sovelluksen osa, jonka käyttäjä näkee. Käyttöliittymän kautta käyttäjä voi nähdä sovelluksen ja käyttää sen toiminnallisuuksia.
- Mobiilisovellus: Erityinen applikaatio, jota käytetään vain mobiililaitteella.
- MVP: Minimum Viable Product. Sisältää listan ominaisuuksista, jotka sovelluksen täytyy vähintään sisältää, jotta ohjelmistoprojekti on onnistunut.
- Prototyyppi: Nopeasti toteutettu osa sovellusta, jonka avulla voidaan testata erilaisia vaihtoehtoja. Tässä opinnäytetyössä prototyyppejä käsitellään vain sovelluksen käyttöliittymän osalta.
- Rautalankamalli: Sovelluksesta tuotettu malli, joka kuvaa rakenteiden sijoittelun eri näkymissä. Rautalankamalli ei ota kantaa tai selvennä sovelluksen sisältöä.
- Sovelluskartta: Hierarkkinen kuvaus sovelluksen eri näkymistä ja niiden toiminnallisuuksista.
- UML-kaaviot: Unified Modeling Language, erityinen tapa tuottaa sovelluksia kuvaavia kaavioita. Soveltuu erityisesti oliopohjaiseen ohjelmointiin.
- Vaatimusmäärittely: Sovelluksen suunnittelua edeltävä työvaihe, jossa määritellään kaikki vaatimukset, joita lopulliselta sovellukselta odotetaan.

2 Mobiilisovelluksen suunnittelu

Erityisesti aloittelevat ohjelmoijat aliarvioivat suunnittelun tärkeyttä ja voivat aloittaa projektinsa toteutuksen ilman suurempaa harkintaa. Vaikka pieniä, yhden henkilön tekemiä ohjelmia voi tuottaa lähes millaisilla menetelmillä vain, ohjelmistokehityksen systemaattiset menetelmät ovat niissäkin hyödyllisiä ja tuovat toimintaan suunnitelmallisuutta. Projekteissa, joiden parissa työskentelee useampi ohjelmoija, tarvitaan väkisin systemaattisempia menetelmiä. (Luukkainen & Laine 2010, 1.) Suunnitteluun kannattaa siis panostaa jo varhaisessa ohjelmointiuran vaiheessa, erityisesti mikäli aikoo tehdä projekteja yhteistyössä muiden kanssa.

Mobiilisovellukset eroavat tavallisista työpöytäsovelluksista monin tavoin, muun muassa sovellusalustan, käytettyjen teknologioiden ja käyttäjäkontekstin osalta. Kuitenkin keskeiset suunnittelu- ja kehitysprosessit ovat samankaltaisia keskenään. (Ballard 2007, 1.) Vaikka suunnitteluprosessit eivät luonteeltaan eroa mobiili- ja työpöytäsovellusten välillä, tulee suunnittelun lopputuloksessa huomioida, millaiselle laitteelle sovellusta ollaan kehittämässä. Esimerkiksi käyttöliittymää suunnitellessa tulee ottaa huomioon, ettei käyttäjä kyky koskea näyttöön ei ole yhtä tarkka kuin tietokoneen hiirellä klikatessa. Tämä voi vaikuttaa siihen, kuinka käyttäjä pystyy käyttämään sovelluksen ominaisuuksia, esimerkiksi navigaatiota. (Nielsen & Budiu 2013, 41.)

Luukkainen ja Laine (2010, 1–2) mainitsevat ohjelmistotuotannon ensimmäisiksi vaiheiksi järjestyksessä vaatimusmäärittelyn ja sen jälkeen suunnittelun. Ennen ohjelmiston tarkempaa suunnittelua tulee siis toteuttaa vaatimusmäärittely, jossa määritellään ohjelmistoon toivottavat toiminnallisuudet. Vasta tämän jälkeen voidaan siirtyä ohjelmiston mallintamiseen eli suunnitteluun. Myös Sinkkonen, Nuutila, ja Törmä (2009, 49) neuvovat aloittamaan uuden palvelun suunnittelun vaatimusmäärittelyllä, jotta tuotettavasta ohjelmasta saadaan heti mahdollisimman selkeä kuva. Tämä auttaa luomaan tarkat ja luotettavat suunnitelmat ohjelmistolle.

2.1 Vaatimusmäärittely

Idea uuteen sovellukseen voi syntyä hetkessä, mutta idea itsessään ei vielä anna keksijälleen tai sovelluksen toteuttajalle tarkkaa kuvaa siitä, mitä kaikkea sovelluksella voisi tehdä tai miten sen tulisi toimia. Tämän vuoksi ideoinnin jälkeen aloitetaan vaatimusmäärittely, jonka tarkoituksena on tuottaa selkeä ja ristiriidaton kuvaus uudesta sovelluksesta. (Sinkkonen ym. 2009, 49.) Mikäli sovelluksen tuotannossa on mukana eri taho keksimässä ideaa ja toteuttamassa sitä, molempien tahojen tulisi osallistua vaatimusmäärittelyyn. Osallistamalla molemmat tahot määrittelyyn pyritään varmistamaan, että sovelluksen toiminnallisuus vastaa haluttua. (Luukkainen & Laine 2010, 1.)

Vaatimusmäärittely on laaja prosessi, johon kuuluu nimensä mukaisesti erilaisten vaatimusten määrittely tulevalle ohjelmistolle. Nämä vaatimukset eivät rajoitu vain ohjelmiston toiminnallisuuksiin, vaan kattavat myös tarvittavat toimintaympäristön ja teknologian aiheuttamat rajoitukset, tieto- ja käyttäjävaatimukset, saavutettavuuden ja käytettävyyden määritykset sekä turvallisuusvaatimukset. Vaatimusmäärittelyn aikana käydään tarvittaessa läpi myös projektiin liittyviä vaatimuksia. (Luukkainen & Laine 2010, 1; Sinkkonen ym. 2009, 49.)

Taulukossa 1 kuvaillaan erilaisia vaatimuksia ja annetaan niistä käytännön esimerkkejä. Taulukossa jokaiseen vaatimusluokkaan annetaan vain yksi esimerkki. On tärkeää huomioida, että jokainen vaatimusluokka on laaja kokonaisuus ja ne voivat sisältää esimerkistä suurestikin poikkeavia vaatimuksia. Tässä opinnäytetyössä keskitytään taulukon jälkeen vain toiminnallisiin vaatimuksiin.

Taulukko 1. Vaatimusmäärittelyn erilaiset vaatimusluokat ja niiden kuvaukset sekä käytännön esimerkit (mukaillen Sinkkonen ym. 2009, 49)

Vaatimusluokka	Kuvaus	Esimerkki
Käytettävyysvaatimus	Minkälaisia sovelluksen ominaisuuksien tulee olla, jotta käyttökokemus yltää halutulle tasolle?	Sovelluksen navigaation tulee olla riittävän suuri, jotta se on helposti kaikkien käytettävissä.
Käyttäjävaatimus	Millaisia käyttäjiä sovelluksella on ja mitä tietoja tai kokemusta heiltä vaaditaan?	Sovelluksen käyttäjien tulee osata suomen kieltä.
Projektiin liittyvät vaatimukset	Millaisella aikataululla projekti toteutetaan?	Sovelluksen viimeinen mahdollinen valmistumispäivä on 25.5.2024.
Saavutettavuusvaatimus	Millaisia ikä- tai vammaisryhmiä otetaan huomioon projektin toteutuksessa?	Sovelluksen käyttöliittymässä otetaan huomioon yleisimmät värisokeudet ja huolehditaan, etteivät nämä johda ongelmiin sovellusta käytettäessä.
Tietovaatimus	Mitä tietoja tarvitaan, kuinka tarkkoja tai ajantasaisia tietojen pitää olla, voiko tieto muuttua?	Sovellukseen tarvitaan ajantasaista tietoa aiheesta X.
Toiminnallinen vaatimus	Mitä ohjelmistolla voi tarkalleen tehdä?	Käyttäjä voi vaihtaa sovelluksessa profiilikuvansa.
Toimintaympäristövaatimus	Millaisessa sosiaalisessa ja fyysisessä ympäristössä sovelluksen tulee toimia, mitä käytettävältä laitteelta vaaditaan?	Sovelluksen tulee toimia Android-laitteilla, joiden käyttöliittymäversio on vähintään Android 12.

Vaatimusluokka	Kuvaus	Esimerkki
Turvallisuusvaatimukset	Mikä on vähimmäistaso, jolle sovelluksen turvallisuudessa halutaan yltää?	Käyttäjät kirjautuvat sovellukseen henkilökohtaisilla tunnuksillaan.

Vaatimusmäärittelyssä keskitytään miettimään mitä sovellukselta halutaan. Tarkoitus ei ole tässä vaiheessa ottaa kantaa siihen, miten nämä toiminnallisuudet toteutetaan. Sovellukselle voidaan kuitenkin määrittää joitain reunaehtoja, kuten käytettävän laitteen käyttöliittymäversio, jolla sovelluksen tulee vähintään toimia. (Luukkainen & Laine 2010, 1.)

Toiminnalliset vaatimukset kuvaavat kaikki ne toiminnot, joita ohjelmistoon toivotaan. Kaikki nämä vaatimukset eivät kuitenkaan ole samanarvoisia keskenään, vaan niitä voidaan jaotella esimerkiksi pakollisiin ja hyödyllisiin ominaisuuksiin. Pakolliset vaatimukset muodostavat yhdessä MVP:n (minimum viable product), eli minimivaatimukset täyttävän sovelluksen. Vaikka sovellus on täysin toimiva vain näillä minimivaatimusten mukaisilla toiminnallisuuksilla, on todennäköistä, että kehittäjälle tulee mieleen myös toiminnallisuuksia, jotka tekisivät sovellukset paremman. Nämä ovat niin sanottuja ’nice-to-have’ (mukava-olla) -ominaisuuksia, jotka eivät kuitenkaan ole pakollisia projektin onnistumisen kannalta. Ne voidaan luokitella siis hyödyllisiin ominaisuuksiin. (Filipova & Vilão 2018, 53–54.)

Vaatimusmäärittelyn tuotoksena syntyy määrittelydokumentti, johon on listattu sovelluksen toiminnallisuuksille esitetyt vaatimukset. Ohjelmistoa testataan useimmiten määrittelydokumentin vaatimuksia vastaan. (Luukkainen & Laine 2010, 2.) Ohjelmistoprojektin onnistuminen vertautuu siis siihen, kuinka hyvin esitettyihin vaatimuksiin on päästy.

2.2 Ohjelmiston mallintaminen

Ohjelmiston suunnitteleminen aloitetaan yleensä arkkitehtuurin suunnittelulla. Arkkitehtuurilla tarkoitetaan ohjelmiston erilaisten rakenneosien tunnistamista ja nimeämistä, kuten tietokanta, sovelluslogiikka ja käyttöliittymä. Nämä osat jaetaan edelleen pienempiin osiin, alikomponentteihin, joiden suunniteluun ja tuotantoon voidaan valita eri tekijöitä. Komponenttien välille on tärkeää suunnitella rajapinnat, joiden avulla mahdollistetaan erillään toteutettavien osien yhteistoimivuus. Rajapinnat kertovat eri komponenttien tarjoamat tiedot ja toiminnot. (Luukkainen & Laine 2010, 2.) Tässä opinnäytetyössä ei perehdytä sovellusarkkitehtuurin tämän syvemmin, vaan keskitytään käsittelemään ohjelmistojen mallintamista.

Ohjelmistoja voidaan mallintaa usein eri tavoin. Voidaan esimerkiksi kirjoittaa käyttäjätarinoita, joissa kuvataan, kuinka yksittäiset käyttäjät käyttäisivät sovellusta ja mitä erityistoiveita heillä sovellukselle olisi (Sinkkonen ym. 2009, 175). Voidaan myös luoda erilaisia kaavioita kuvaamaan

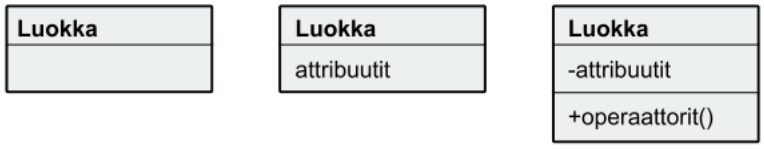
ohjelmiston eri ominaisuuksia. Näitä kaavioita voidaan toteuttaa lukuisilla tekniikoilla, joista yksi suosituimmista on UML eli Unified Modeling Language (Luukkainen & Laine 2010, 2).

UML on 90-luvulla kehitetty kuvaustekniikka, joka on kehitetty oliopohjaisia projekteja varten yhdistämällä sitä edeltäneitä kuvaustekniikoita. UML:ssä on määritelty erilaisia kaaviotyyppejä ohjelmiston kuvaamista varten ja osaa niistä voi käyttää monessa eri tarkoituksessa. UML:n kaaviotyyppejä on yhteensä 13 erilaista, mm. luokka-, käyttötapaus-, olio-, sekvenssi- ja tilakaavio. (Luukkainen & Laine 2010, 7–8.) Tässä opinnäytetyössä keskitytään luokkakaavioihin.

Luokkakaavion avulla kuvataan ohjelmiston rakennetta ja sen osien välisiä yhteyksiä, tietosisältöä sekä ohjelmiston tuottamia palveluita. Luokkakaavioissa yksi luokka kuvaa olioita, jotka ovat samankaltaisia ja niin sanotusti kuuluvat samaan oliolajiin. Esimerkiksi opintojärjestelmä voi sisältää luokan *opiskelija*, jolloin jokainen tämän luokan ilmentymä, eli olio, kuvaa yhtä opiskelijaa. (Luukkainen & Laine 2010, 21–22.) Bennett (2015c, 00:15–01:05 min.) kuvailee luokan ja olion eroa siten, että luokat kuvastavat erilaisten tyyppien määritelmiä järjestelmässä oliot puolestaan ovat yksittäisiä tietojoukkojen ilmentymiä tietyllä toimialueella. Luokat määritellään koodissa ja niiden avulla rakennetaan ohjelmisto. Ohjelmiston ollessa käynnissä oliot ovat olemassa sen muistissa ja vuorovaikuttavat muiden ohjelmiston osien kanssa luoden sen toiminnallisuuden. (Bennett 2015c, 00:15–01:05 min.)

Luokat sisältävät erilaisia attribuutteja, jotka yksilöivät erilaiset luokat toisistaan. Attribuuteista voidaan puhua myös ominaisuuksina, muuttujina tai kenttinä ja ne sisältävät tietoja oliosta tai kertovat, millaisia palveluita oliolta voi pyytää. Attribuutteja voi olla eri tyylisiä, esim. yksittäisiä tai rakeisia tietoja, tietokokoelmia tai viitteitä toisiin olioihin. (Luukkainen & Laine 2010, 22–23; Shelton 11.8.2023, 0:40–1:30 min.)

UML:ssa on määritelty luokan visuaaliseksi ilmeeksi suorakaide, joka sisältää aina vähintään luokan nimen, mutta voi sisältää myös attribuuttien ja/tai operaattorien nimet. Luokan nimi, attribuutit ja operaattorit erotellaan toisistaan poikkiviivojen avulla. Luokan nimi kirjoitetaan isolla alkukirjaimella ja lihavoidulla fontilla. Attribuutit ja operaattorit kirjoitetaan pienellä alkukirjaimella. Luokkakaavioon voidaan kirjoittaa luokan, attribuuttien ja operaattoreiden nimet joko selkokielisesti tai käytettävän koodikielen syntaksin mukaan. Operaattoreiden kohdalla koodikielen syntaksin käyttäminen on jopa suositeltavaa. (Bennett 2015b, 00:15–00:50 min.; Luukkainen & Laine 2010, 21–27; Shelton 11.8.2023, 0:40–1:30 min.) Kuva 2 havainnollistaa, kuinka sama luokka voidaan kuvata eri laajuisena UML:n avulla.



Kuva 2. Luokka kuvattuna eri laajuisena (mukaillen Luukkainen & Laine 2010, 23)

Luokkakaaviossa attribuutteja voidaan esittää pelkällä attribuutin nimellä. Nimi yksinään ei kuitenkaan kerro attribuuteista kaikkea tarpeellista, joten jos luokkakaavion on merkitty attribuuteista vain nimet, on tärkeää kirjoittaa tekstimuotoinen määrittely täydentämään kaaviota. On kuitenkin mahdollista määritellä luokkakaavioon näkyville muitakin arvoja attribuuteille, kuten näkyvyys, tietotyyppi, oletusarvo tai, kokoelman ollessa kyseessä, moniarvoisuus. Attribuuttien eri arvot merkitään luokkakaavioon järjestyksessä vasemmalta oikealle: näkyvyys, nimi, moniarvoisuus, tietotyyppi, oletusarvo ja lopuksi mahdolliset muut määreet. Attribuuttien arvoja kirjatessa kaavioon merkitään myös ennen tietotyyppiä kaksoispiste, ennen oletusarvoa yhtäsuuruusmerkki ja muut määreet kirjataan aaltosulkeiden sisään. (Bennett 2015b, 00:15–00:50 min.; Luukkainen & Laine 2010, 24–25.) Taulukossa 2 kerrotaan lisää näistä attribuuttien eri arvoista ja niiden esitystavoista.

Taulukko 2. Luokkien attribuuttien eri arvot selitettyinä (mukaillen Luukkainen & Laine 2010, 24)

Attribuutin arvo	Kuvaus
Nimi	Luokan nimi, yksilöi luokan. Ainut välttämätön tieto attribuuteille.
Tietotyyppi	Voidaan määritellä tarpeen mukaan joko ohjelmointikielen mukaan (esim. string, boolean) tai tapauskohtaisesti (prosenttiosuus, png-kuva).
Oletusarvo	Arvo, jonka attribuutti saa, mikäli sille ei syötetä ohjelmassa muuta arvoa.
Moniarvoisuus	Kokoelmien vähimmäis- ja enimmäismäärät voidaan määritellä hakasulkeisiin. *-merkki tarkoittaa lukuisia vaihtoehtoja, kun absoluuttista ylärajaa ei voida antaa. Esim. [1..*] tarkoittaa, että kokoelma voi sisältää yhdestä moneen arvoa.
Näkyvyys	Merkittävä tieto vain ohjelmointikielitason kaavioissa. Sisältää neljä eri tasoa ja ne merkitään luokkakaavioon erilaisilla merkeillä ennen attribuutin nimeä; julkinen (merkkinä +), suojattu (merkkinä #), pakkauksen sisäinen (merkkinä ~) ja yksityinen (merkkinä -).
Muut määreet	Mikä tahansa attribuutti voidaan määritellä muuttumattomaksi antamalla sille määre *readonly*. Kokoelmiin voidaan liittää määre *ordered* kertomaan kokoelman olevan järjestyksessä tai määre *unique* kertomaan, että sen arvot ovat toisistaan eroavia.

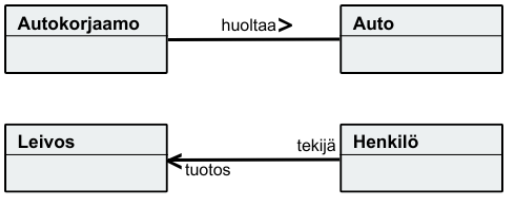
Luokan operaattoreille voidaan myös määrittää erilaisia arvoja, kuten nimi, näkyvyys, paluuarvon tyyppi tai parametrit. Operaattorien arvot ilmaistaan luokkakaaviossa järjestyksessä vasemmalta

oikealle: näkyvyys, nimi, parametrit, paluuarvon tyyppi ja mahdolliset muut määreet. Parametrit merkitään sulkumerkkien sisään ja paluuarvon tyypin edelle merkitään kaksoispiste. (Bennett 2015b, 03:15–04:00 min.; Luukkainen & Laine 2010, 25–27.) Taulukossa 3 esitellään operaattoreiden erilaiset arvot.

Taulukko 3. Luokkien operaattorien erilaiset arvot ja niiden merkitykset (mukaillen Luukkainen & Laine 2010, 26–27)

Operaattorin arvo	Kuvaus
Nimi	Ainut välttämätön arvo operaattorien kuvauksessa, yksilöi operaattorit.
Parametrit	Kuvaavat operaattorin rajapintaa, eli mitä syöttö- tai tulostietoja operaattorille voi välittää. Parametreille voidaan ilmoittaa nimen lisäksi muita arvoja. Merkintäjärjestys kaaviossa on tällöin; suunta (*in*, *out* tai näiden yhdistelmä *inout*), parametrin nimi, tietotyyppi ja mahdollinen oletusarvo.
Näkyvyys	Merkittävä tieto vain ohjelmointikielitason kaavioissa, ylemmän tason abstrakteissa kuvauksissa ollaan kiinnostuneita vain julkisista operaattoreista. Näkyvyys sisältää neljä eri tasoa ja ne merkitään luokkakaavioon erilaisilla merkeillä ennen attribuutin nimeä; julkinen (merkkinä +), suojattu (merkkinä #), pakkauksen sisäinen (merkkinä ~) ja yksityinen (merkkinä -).
Paluuarvon tyyppi	Kertoo, minkä tyyppisen tiedon suoritettava operaatio palauttaa. Voi olla jokin perustietotyyppi tai ohjelmassa käytettävä olioluokka. UML-standardien mukaan operaattorilla voi olla enintään yksi paluuarvo.
Muut määreet	UML:ssä on käytössä joitain samanaikaisuuden hallintaan liittyviä määreitä. Näitä määreitä ei kuitenkaan käsitellä tässä opinnäytetyössä tämän laajemmin.

Luokkakaavioissa voidaan esittää eri luokkien välisiä yhteyksiä erilaisin tavoin riippuen tuon yhteyden luonteesta ja kaavion tarkkuudesta. Yksinkertaisimmillaan kahden luokan välistä yhteyttä voidaan kuvata niitä yhdistävällä viivalla. Yhteys voidaan myös nimetä suunnatulla tai suuntaamattomalla nimellä. Suunnattu nimi tarkoittaa verbimuotoista nimeä ja sen yhteydessä merkittään nuolella myös mihin suuntaan se kohdistuu. Vaihtoehtoisesti on mahdollista nimetä yhteyden osapuolille yhteyttä kuvaavat roolit. Tässä tapauksessa roolit kirjoitetaan siihen päähän viivaa, missä nimetty luokka sijaitsee. Myös rooleja nimetessä voi merkitä yhteyden suunnan kaavioon. Yhteyden suunta muodostuu aina sen mukaan, mistä luokasta on suora pääsy toiseen. Suoralla pääsyllä tarkoitetaan sitä, että jokin luokan attribuuteista viittaa tuohon toiseen luokkaan. Suunta voi olla yksi- tai kaksisuuntainen. (Bennett 2015a, 00:20–01:15 min., 02:35–02:55 min.; Luukkainen & Laine 2010, 28–33.) Kuva 3 auttaa hahmottamaan näitä yhteyksien nimeämis- ja suuntausmerkintöjä.

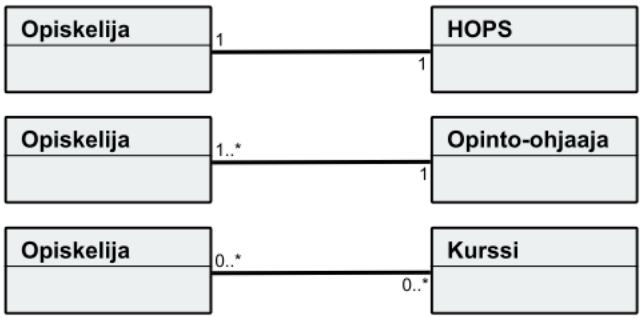


Kuva 3. Luokkien välisen yhteyden nimeäminen ja suunnan osoittaminen eri tavoin

Luokkakaavion yhteyksille voidaan merkitä kytkentärajoitteita. Kytkentärajoitteella tarkoittaan sitä, kuinka monta yhteyttä luokkien mukaisilla olioilla voi olla. Rajoitteet merkitään alaraja..yläraja ja ne merkitään aina sen luokan viereen, jota kohti yhteyttä ollaan rajoittamassa. Mikäli luokkien välinen yhteys ei ole pakollinen, voidaan alarajaksi asettaa 0. Ja jos taas ylärajaa ei haluta asettaa, numeraalisen arvon sijasta voidaan merkitä *. (Bennett 2015a, 01:15–02:35 min.; Luukkainen & Laine 2010, 31–31; Shelton 11.8.2023, 11:25–12:05 min.)

Kuvassa 4 esitettään kaikki kolme erilaista kytkentärajoiteyhdistelmää: yhden suhde yhteen, yhden suhde moneen ja monen suhde moneen. Kuvassa merkityt rajoitteet tulkitaan seuraavasti:

- Yhteen opiskelijaolioon voi liittyä vain yksi HOPS-olio ja yhteen HOPS-olioon voi liittyä vain yksi opiskelijaolio.
- Yhteen opiskelijaolioon voi liittyä vain yksi opinto-ohjaajaolio, mutta yhteen opinto-ohjaajaolioon voi liittyä useita opiskelijaoliota.
- Yhteen opiskelijaolioon voi liittyä ei yhtäkään tai monta kurssioliota ja yhteen kurssiolioon voi liittyä ei yhtäkään tai monta opiskelijaoliota.



Kuva 4. Luokkien välisten yhteyksien erilaiset kytkentärajoitteet

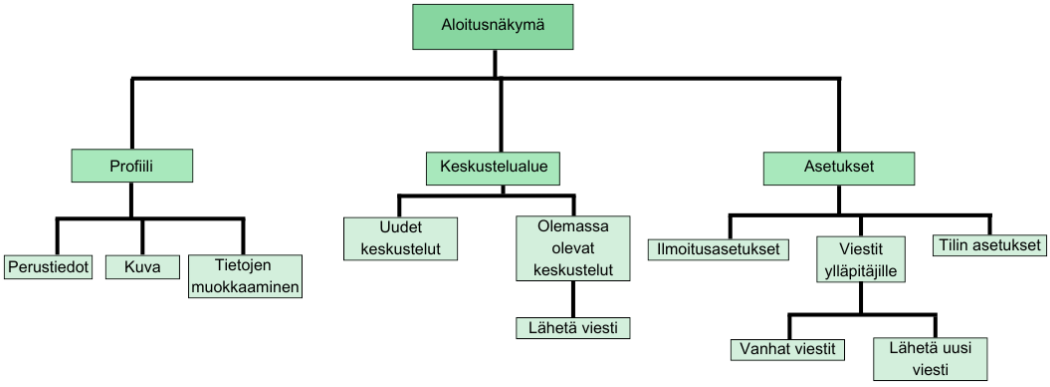
2.3 Käyttöliittymän suunnittelu

Ohjelmiston käyttöliittymän suunnittelu on monivaiheinen prosessi, joka alkaa sovelluskartan tekemisellä. Sovelluskarttaan sijoitetaan sovellukseen toivotut elementit hierarkkiseen järjestykseen.

Sovelluskartta auttaa hahmottamaan, kuinka käyttäjät liikkuvat sovelluksessa. (Wells 2023, 137.) Sinkkonen ja muut (2009, 196–198) toteavat tällaisen rakennemallin kuvaavaan palvelun informaatiorakennetta ja huomauttavat, että saattaa tulla lisäyksiä vielä myöhemmissä vaiheissa. Tämän vuoksi rakennemallia on hyvä testata esimerkiksi käyttäjätarinoiden avulla ennen seuraavaan vaiheeseen siirtymistä.

Käyttäjätarinoita voidaan kirjoittaa pohtimalla, kuinka käyttäjät haluaisivat sovellusta käyttää; mitä tietoa tai toimintoja tämä tarvitsee. Tämän jälkeen sovelluskarttaa verrataan näihin tarinoihin yksi kerrallaan. Jokaisen tarinan kohdalla tarkkaillaan, kuinka käyttäjä siirtyy näkymästä toiseen tarvitsemiensa tietojen tai toiminnallisuuksien perässä. Jos käyttäjä joutuu usein siirtymään ylös-alas hierarkiassa, kannattaa näihin kohtiin lisätä linkit toiminnallisuuksien yhdistämiseksi. (Sinkkonen ym. 2009, 197.)

Kuvassa 5 esitetään esimerkki mobiilisovelluksen sovelluskartasta. Hierarkiassa ylimpänä on sovelluksen aloitusnäkymä, sillä se eroaa käyttötarkoitukseltaan muista näkymistä. Aloitusnäkymä aukeaa aina käyttäjän avatessa sovelluksen, kun taas muihin näkymiin siirrytään valikon kautta. Toisella tasolla on sovelluksen muut näkymät ja niiden alla jokaisen näkymän tärkeimmät toiminnallisuudet.

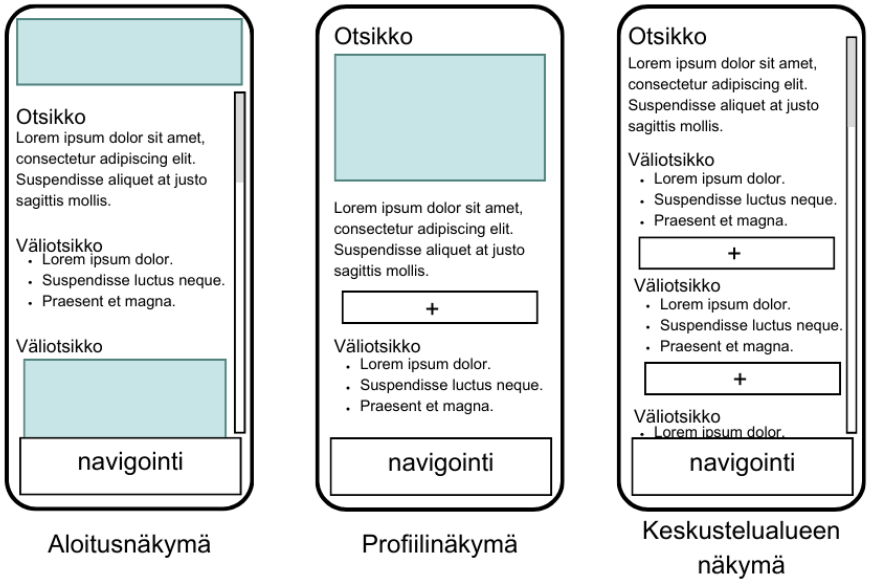


Kuva 5. Esimerkki sovelluskartasta (mukaillen Wells 2023, 138)

Valmiin sovelluskartan pohjalta voidaan jalostaa käyttöliittymän rautalankamalli. Rautalankamalli on hieman samankaltainen kuin sovelluskartta. Sen tarkoitus ei ole määrittää sovelluksessa käytettäviä värejä, typografiaa tai muutakaan tarkkaa tietoa käyttöliittymän ulkoasuun liittyen. Sen sijaan tavoite on luoda mahdollisimman selkeä kuva sovelluksen rakenteesta. (Wells 2023, 141.) Sinkkonen ja muut (2009, 203) huomauttavat rautalankamallin sisältävän vain ohuen kerroksen visuaalista suunnittelua, kyseessä on edelleen vain rakenteen suunnittelu.

Rautalankamalliin asetetaan paikoilleen erilaiset sovellukseen tulevat komponentit, kuten navigointi, kuvat, otsikot ja leipäteksti. Näiden visualisointiin käytetään erilaisia ruudukko- ja kolumnirakenteita, jotka toimivat paikanvaraajina. Valmiista rautalankamallista näkee nopealla vilkaisulla esimerkiksi, mihin sovelluksessa sijoitetaan navigaatio tai kuinka eri tekstiosuudet erottuvat toisistaan. Mikäli sovelluksen eri näkymiin halutaan erilaisia rakenteita, tulee niistä valmistella omat rautalankamallinsa. (Sinkkonen ym. 2009, 203; Wells 2023, 141.)

Kuva 6 havainnollistaa erilaisista näkymistä tehtyjä rautalankamalleja. Kuvasta on havaittavissa, että rautalankamallissa ei oteta lainkaan kantaa sovelluksen sisältöön, vaan rakenne näytetään mahdollisimman pelkistetyssä muodossa. Esimerkissä navigointi pysyy samassa kohdassa jokaisessa näkymässä, mutta muiden elementtien paikka vaihtelee.



Kuva 6. Rautalankamalleja mobiilisovelluksen käyttöliittymästä

Rautalankamallien pohjalta luodaan sovelluksen prototyypit. Prototyyppi on tarkka kuvaus tuotettavasta palvelusta, jonka avulla voidaan testata erilaisia vaihtoehtoisia ratkaisuja ilman, että tarvitsee tuottaa koodia. Prototyyppejä voidaan tehdä sykleittäin useassa vaiheessa, jolloin jokaisen vaiheen välissä voidaan testata prototyypin käytettävyyttä ja pohtia kehityskohteita seuraavaan sykliin. (Sinkkonen ym. 2009, 204; Wells 2023, 144.)

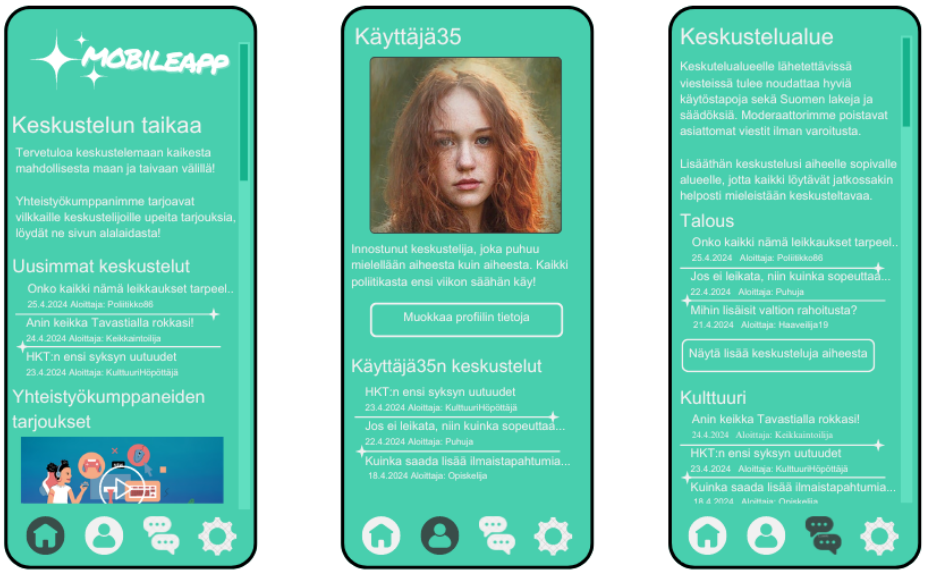
Sekä Wells (2023, 144) että Sinkkonen ja muut (2009, 205) ehdottavat ensimmäisten prototyyppien piirtämistä paperille. Wells (2023, 144) kertoo paperisten prototyyppien käyttötarkoitukseksi erityisesti navigaation ja toiminnallisuuksien testaamisen. Hyötynä tälle vaiheelle hän mainitsee sen, että toiminnallisuuksia voidaan testata ilman, että käyttäjä kiinnittää liikaa huomiotaan sovelluksen ulkoasuun. Sinkkonen ja muut (2009, 205) luettelevat paperisille prototyypeille enemmänkin

hyötyjä, esimerkiksi niiden tuottamisen helppouden ja sen, että käyttäjien on usein helpompi arvostella prototyyppiä, joka on käsin luonnosteltu, eikä siten liian viimeistelty. Paperiset prototyypit koostuvat useista luonnoksista, jotka kuvaavat näkymiä eri toiminoissa ja niiden vaiheissa.

Paperisten prototyyppien pohjalta voidaan siirtyä teräväpiirtoisten prototyyppien valmistamiseen. Tässä vaiheessa suunnitelmiin lisätään kaikki tyylitekijät; värit, kuvat, halutut fontit, sisältöä kuvaavat tekstit. Teräväpiirtoisen prototyypin tarkoitus on näyttää sovelluksen visuaalinen yhteneväisyys, toiminnallisuuksien selkeys, kirjasimien ja kuvien koot ja tyylit, tekstissä käytetyn kielen tyyli ja sommittelun periaatteet. Valmis prototyyppi kuvaa tarkalleen, miltä lopullinen sovellus tulee näyttämään. (Wells 2023, 144–147.)

Vaikka Wells (2023, 144–147) painottaa teräväpiirtoisen prototyypin valmistamisessa ulkonäöllisiä seikkoja, Filipova ja Vilão (2018, 97–98) huomauttavat etteivät värit, fontit ja muut tyylilliset yksityiskohdat ole välttämättömiä. Heidän mukaansa tällaiset tekijät kuuluvat yritysten brändiin, ja mikäli suunnitelmaa ei tehdä yritykselle, nämä määrittelyt eivät ole pakollisia. Tällöin prototyyppi voidaan kuvata vaikka mustavalkoisena sen esittäessä vain eri elementtien koot ja sijainnit.

Kuvassa 7 näytetään esimerkkejä, miltä mobiilisovelluksen teräväpiirtoiset prototyypit voivat näyttää. Esimerkkien prototyypeissä ulkoasun tyyli on viimeistelty loppuun saakka väreineen ja fontteineen.



Aloitusnäkymä | Profiilinäkymä | Keskustelualueen näkymä

Kuva 7. Mobiilisovelluksen teräväpiirtoisia prototyyppejä

3 Opas suunnittelun tueksi

Opinnäytetyön idea syntyi pitkällisen pohdinnan seurauksena. Opinnäytetyön tekijä halusi valita työlleen aiheen, jossa voisi hyödyntää omia vahvuuksiaan, tuottaa jotain uutta ja hyödyllistä muille alalla oleville tai alalle haluaville sekä kehittää omia taitojaan ohjelmoijana. Kirjallinen opas mahdollistaa opinnäytetyön tekijän kirjallisten taitojen hyödyntämisen ja graafinen puoli tuo tekijälle uutta opeteltavaa. Muiden ohjaaminen on tekijälle entuudestaan tuttua, sovellusten suunnittelun ohjeistaminen taas uutta.

Tuotettavan oppaan tavoitteena on toimia tuleville IT-alan opiskelijoille tukena ensimmäisten projektien suunnittelussa ja sen tarkoitus on myös selventää uusille tekijöille suunnittelun tärkeyttä. Oppaan laadulliseksi mittariksi asetettiin se, että sen sisältö on ymmärrettävissä myös sellaiselle henkilölle, joka ei ole opiskellut tietojenkäsittelyä, vaan haluaa vasta tutustua alaan. Tämän ehdon täyttäessään oppaan voidaan katsoa olevan tarpeeksi selkeästi kirjoitettu ensimmäisiä sovelluksiaan tekeville henkilöille.

Opinnäytetyöllä ei ollut toimeksiantajaa, vaan se toteutettiin itsenäisenä työnä. Projektiin ryhtyminen ilman toimeksiantajaa tuntui luonnolliselta vaihtoehdolta, sillä opinnäytetyön tekijällä on työkokemusta ohjelmistojen suunnitellusta sekä korkeakouluopintoja mobiilisovellusten kehittämisestä, vaatimusmäärittelystä ja ohjelmistoprojekteista. Lisäksi tekijällä on työkokemusta opiskelijoiden ohjauksesta toisella alalla, josta on hyötyä oppaan kirjoittamisessa.

Opinnäytetyön suurin rajoittava tekijä on aika. Opinnäytetyön aikarajat asetettiin 4 viikon päähän projektin aloituksesta. Vaikka tiukat aikarajat auttavat sisäisen motivaation ylläpitämisessä, ne asettavat myös rajoituksia työlle. Ohjelmistojen suunnittelu on laaja aihe ja siitä voisi kirjoittaa useitakin teoksia. Kuitenkaan aika opinnäytetyössä ei tällaiseen urakkaan riitä, joten työlle piti asettaa tiukkoja rajauksia. Rajausten tekemisestä kerrotaan lisää alaluvussa 3.2 Oppaan suunnittelu.

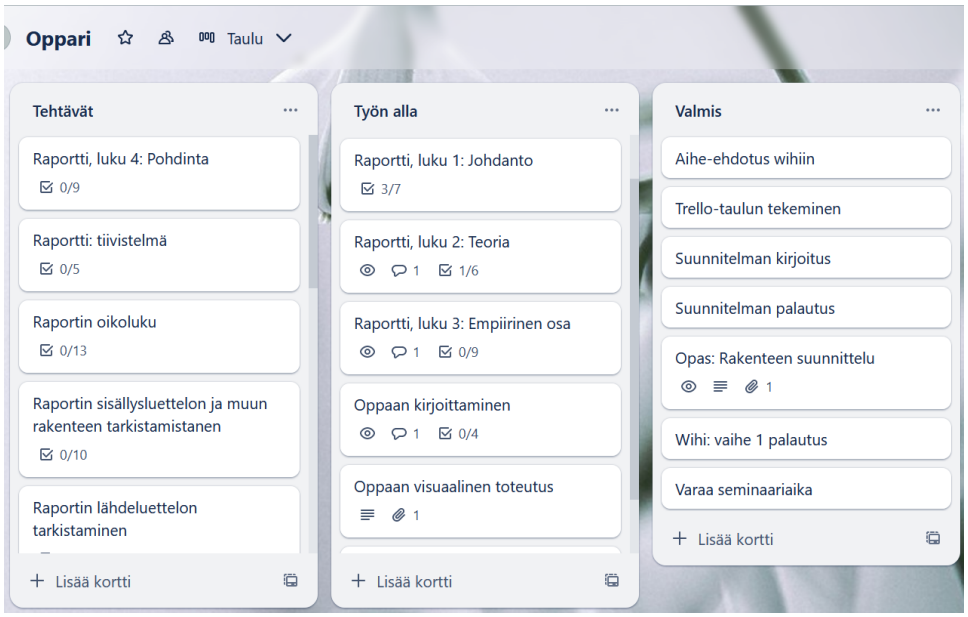
3.1 Projektin hallinta

Projektinhallinnassa päätettiin hyödyntää Kanban-menetelmää. Kanban on projektinhallintaan tarkoitettu työkalu, jolla tehdään suurista projekteista helpommin hallittavia, pienempiä kokonaisuuksia sekä tuodaan työvaiheet ja niiden eteneminen näkyväksi. Tämä auttaa havaitsemaan projektin hitaasti etenevät ja pysähtyneet tehtävät, löytämään projektista kehityskohteita sekä rajoittamaan keskeneräisten tehtävien yhtäaikaista määrää. (Koskinen 26.3.2021.)

Kanbanissa projektin eri vaiheet visualisoidaan omiksi korteikseen yhteiselle taululle. Taululla kortit jaetaan eri kategorioihin riippuen siitä, missä vaiheessa tehtävä on. Kategoriat voidaan nimetä aina

projektikohtaisesti. Tavallisesti korttien jaotteluna käytetään; tekemättä, työn alla ja valmis. Projektin alussa tehtävät sijoitetaan tekemättä-kategoriaan ja sieltä ne siirtyvät seuraavaan, kun ne otetaan työn alle. (Koskinen 26.3.2021.)

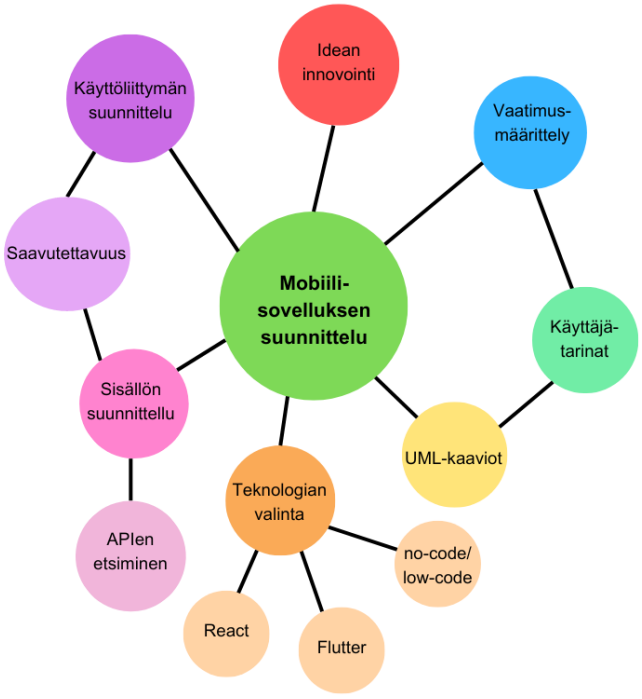
Opinnäytetyön Kanban-taulu toteutettiin Trello-alustalle. Trello on opinnäytetyön tekijälle entuudestaan tuttu alusta, joten oli luontevaa valita se käyttöön. Kategorioiksi taululle valittiin “tehtävät”, “työn alla” sekä “valmis”. Tekemättä-sanaa ei valittu kategorian nimeksi, sillä siinä on opinnäytetyön tekijän mielestä negatiivinen kaiku, joka vaikuttaa laskevasti motivaatioon. Kortteja taululla oli projektin alussa 29 kappaletta. Kuva 8 näyttää, miltä projektin Kanban-taulu näytti, kun opinnäytetyö siirtyi 2. vaiheeseen.



Kuva 8. Projektin Kanban-taulu Trellossa

3.2 Oppaan suunnittelu

Opinnäytetyön valmistelu aloitettiin innostuneella suunnittelulla. Suunnittelun ensimmäisenä vaiheena syntyi ajatuskartta siitä, mitä kaikkea mobiilisovelluksen suunnitteluun liittyy. Kuva 9 esittää opinnäytetyön tekijän alkuperäiset ajatukset siitä, kuinka laaja aihe on kyseessä.



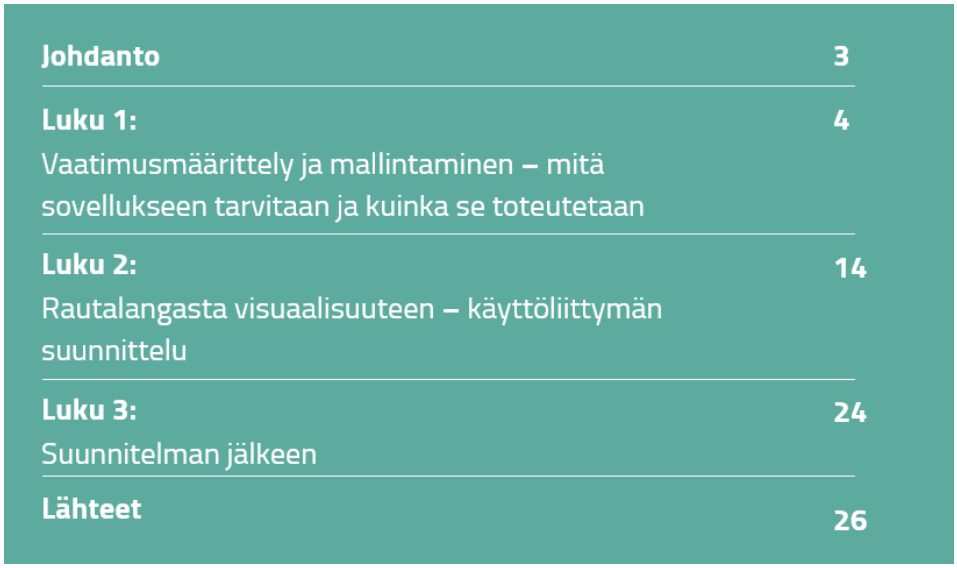
Kuva 9. Ajatuskartta mobiilisovelluksen suunnitteluun liittyvistä eri aiheista ja niiden sisällöistä

Valmiista ajatuskartasta on havaittavissa, että opinnäytetyön aihe on hyvin laaja ja työ voisi sisältää lukuisia eri aiheita. AMK-tason opinnäytetyön työmäärän tulisi kuitenkin vastata 15 opintopistettä. Yksi opintopiste vastaa 27 tuntia työtä, eli opinnäytetyöhön aikaa tulisi käyttää noin 405 tuntia (Ammattikorkeakouluun s.a.). Tuohon työtuntimäärään ei kaikkia ajatuskartasta löytyviä aiheita voisi käsitellä laadukkaasti, joten työtä alettiin rajaamaan.

Aihetta rajattiin pienemmäksi käyttäen kriteeriä “mistä aiheista olisi oman kokemuksen mukaan eniten hyötyä aloittelevalle ohjelmoijalle”. Rajausten jälkeen päädyttiin kahteen pääaiheeseen; vaatimusmäärittely ja käyttöliittymän suunnittelu. Vaatimusmäärittelyyn liitettiin ohjelmistojen mallintaminen UML-kaavioiden avulla, sillä näiden aiheiden yhdistelmä antaa aloittelijallekin selkeän kuvan tuotettavasta sovelluksesta. Käyttöliittymäsuunnittelusta rajattiin saavutettavuus pois, sillä se on aiheena hyvin laaja, eivätkä projektin resurssit riittäisi sen kattamiseen.

Rajausten jälkeen hahmoteltiin oppaan sisällysluettelo; johdanto, luku 1 vaatimusmäärittelystä ja ohjelmiston mallintamisesta sekä luku 2 käyttöliittymän suunnittelusta. Sisällysluetteloa katsoessa syntyi idea oppaan nimestä: Ideasta toteutukseen – opas ensimmäisen mobiilisovelluksen suunnitteluun. Suunnitelma ei kuitenkaan tuntunut täysin valmiilta, sillä hyppy luvusta 2, käyttöliittymän suunnittelu, sovelluksen toteutukseen tuntui liian suurelta. Päätettiin lisätä oppaaseen ekstraluku, jossa sivutaan lyhyesti sitä, kuinka näiden suunnitelmien jälkeen siirrytään kohti toteutusta. Tässä viimeisessä luvussa puhutaan lyhyesti teknologian valitsemisesta ja sisällön suunnittelusta. Lukujen jälkeen listataan vielä tietopohjan teossa hyödynnetty

lähdeluettelo, jotta oppaan lukijat voivat halutessaan perehtyä aiheisiin lisää. Kuvassa 10 esitellään lopullinen oppaan sisällysluettelo, jossa näkyy myös lukujen kuvaavat nimet.



Kuva 10. Tuotetun oppaan sisällysluettelo

3.3 Oppaan toteutus

Oppaan ensimmäinen versio kirjoitettiin Word-sovelluksella. Tämä valinta tehtiin siksi, että opinnäytetyön tekijän kokemuksen mukaan tekstinkäsittelyohjelmista Wordissa on paras oikeinkirjoitus- ja kielioppitarkistus. Kirjoittamalla teksti ensin Word-tiedostoon haluttiin varmistaa, ettei tekstissä esiinny kirjoitusvirheitä. Word-sovelluksen oikeinkirjoituksen tarkistus ei ole täydellinen, mutta sen avulla useimpien ns. huolimattomuusvirheiden korjaaminen onnistuu nopeasti. Wordin tarkistuksen jälkeen teksti oikoluettiin myös tekijän toimesta.

Opasta kirjoittaessa päädyttiin käyttämään apuna käytännön esimerkkejä. Oppaan tekstin edetessä valmistuu kohta kohdalta suunnitelma elokuva-aiheiselle mobiilisovellukselle. Käytännön esimerkkejä haluttiin hyödyntää oppaassa, sillä ne auttavat lukijaa hahmottamaan käsiteltäviä asioita hieman eri tavalla ja laajemmin kuin pelkkä ohjeteksti. Tekstin haluttiin olevan mahdollisimman helposti lähestyttävää, joten kirjoitusasussa päädyttiin käyttämään me-muotoa ja puhtaan kirjakielen sijasta käytettiin siistiä yleiskieltä. Oppaaseen tuotettiin kuvitus käytettyjen esimerkkien mukaan selventämään tekstiä. Kuva 11 näyttää, kuinka oppaan teksti ja kuvitus toimivat yhdessä antaen lukijalle selkeän kuvan käsiteltävästä työvaiheesta.

Kuva 11. Tuotetussa oppaassa vaatimusmäärittely aloitetaan yksinkertaisella ideoinnilla

Visuaalinen puoli oppaaseen toteutettiin pääsääntöisesti Visme-palvelun avulla. Visme on monipuoliseen sisällöntuottoon tarkoitettu web-palvelu (Visme s.a.). Vismen lisäksi oppaan graafisen puolen tuottamisessa on hyödynnetty Canvaa, jonka avulla on valmistettu osa oppaan kuvista. Palvelua valitessa harkittiin myös pelkän Canvan käyttämistä, sillä se oli entuudestaan tuttu opinnäytetyön tekijälle. Visme kuitenkin tarjoaa laajemman valikoiman erilaisia valmiita graafisia pohjia, joita voi myös muokata vastaamaan omia tarpeitaan. Vismen valmiit pohjat ja niiden muokattavuus on saanut kiitosta myös arvioijilta (Duffy 29.9.2021; Melvin 10.1.2024; Paris 28.2.2023). Palvelusta löytyy pohjia myös muistioille ja oppaille, jotka sisältävät eri sivuille erilaiset, mutta tyylisuunnaltaan yhtenevät rakenteet. Opinnäytetyössä käyttöön valittiinkin yksi Vismen monista dokumenttipohjista.

Pohjasta muutettiin heti värimaailma rauhallisemmaksi. Pohjan alkuperäinen kirkas punainen vaihtui hieman siniseen taittavaan vihreään. Oppaasta haluttiin luotettavan ja rauhallisen oloinen, joten värimaailman valintaa pohdittiin myös väripsykologian kautta. Vihreä väri on fysiologisesti ihmissilmälle helpoin väri, jonka vuoksi se auttaa rentoutumaan. Rauhoittava efekti korostuu, mitä vaaleampi vihreä on kyseessä. Vihreä liitetään ajatuksissa usein myös kasvuun ja elämään, johtuen sen luonnossa esiintymisestä. Koska tuotettavassa oppaassa opetellaan uutta taitoa ja ”kasvetaan tekijänä”, vihreä oli luonteva valinta värimaailmaan. Sininen väri on myös luonteeltaan rauhoittava, mutta lisäksi se herättää katsojassa luottamusta. Tämän vuoksi oppaan vihreä väri päätettiin

taittaa siniseen, jolloin saadaan molempien värien hyödyt osaksi opasta. (Sutton 2020, luku The Psychology of Color.)

Kuva 12 havainnollistaa oppaan värimaailmaa ja sen luomaa tunnelmaa. Etusivun kuva on valittu oppaan aiheen mukaan ja sen tarkoitus on kiinnittää visuaalisten ihmisten huomio oppaaseen.

Kuva 12. Tuotetun oppaan etusivu

3.4 Oppaan luetuttaminen testilukijoilla

Opas oli kirjoittajansa mielestä valmis, kun siinä käsiteltiin vaatimusmäärittely ja ohjelmiston mallintaminen niin pitkälle, että tehdyistä suunnitelmista pystyi päättelemään sovellukseen tarvittavat luokat, koodikomponentit ja sovelluksen näkymät. Lisäksi käyttöliittymän suunnittelu oli käsitelty sen verran kattavasti, että oppaassa suunniteltuun sovellukseen oli tuotettu tarkkapiirtoiset prototyypit. Tämän jälkeen opas lähetettiin testiluettavaksi kolmelle henkilölle. Testilukemisella haluttiin varmistaa, että valmis opas on tuotettu tarpeeksi kattavaksi ja helposti lähestyttäväksi sellaisille henkilöille, jotka eivät ole vielä ohjelmoineet omia sovelluksiaan.

Testilukijoiksi valittiin kolme henkilöä erilaisista taustoista

- korkeakoulussa tietojenkäsittelyä opiskellut henkilö, joka on valinnut suuntautumisekseen muun kuin ohjelmistokehityksen linjan
- korkeakoulussa opiskellut henkilö, joka ei kuitenkaan opiskellut tietojenkäsittelyä
- henkilö, joka ei ole koskaan opiskellut korkeakoulussa.

Testilukijoille toimitettiin saatekirjeessä (liite 2) lista kysymyksiä, joihin toivottiin vastauksia oppaan arvioinnissa. Mitään kysymystä ei kuitenkaan määritelty pakolliseksi vastattavaksi. Vastaustapa jätettiin lukijoille vapaasti valittavaksi, jolloin he saivat päättää, vastaavatko yhtenäisellä tekstikappaleella kysymyksiin vai lähettävätkö vastaukset kohtakohdalta.

Testilukijoilta saadussa palautteessa (liite 3) korostui tyytyväisyys oppaan laatuun. Aiheiden käsittely koettiin nopeatahtiseksi, mutta selkeäksi. Erityisesti esimerkkien käyttäminen ja kuvitus saivat testilukijoilta kiitosta. Näiden koettiin auttaneen aiheiden ymmärtämisessä. Myös oppaan lopusta löytyvä lähdeluettelo sai kiitosta, sen koettiin olevan helpottavan aiheisiin syvemmin perehtymistä.

Testilukijoilta saatiin myös kehitysideoita. Kaksi kolmesta lukijasta kertoi uusien termien määrän olleen suuri ja sen aiheuttaneen epävarmuuden tunnetta. Lukijat toivoivat termien käsittelyyn vielä lisää ns. rautalangan vääntämistä, jotta uusien termien sisäistäminen helpottuisi. Saadun palautteen perusteella lukuihin 2 ja 3 lisättiin loppuun termiluettelot, joista voi kerrata eri termien merkityksiä. Lisäksi eräs testilukijoista kommentoi oppaassa tuotettua vaatimuslistaa ja toivoi siihen lisäyksiä ja tarkennuksia. Toiveiden toteuttaminen olisi kuitenkin tehnyt 2 luvusta vielä abstraktimman ja vaikeammin sisäistettävän, joten pyydettyihin muutoksiin ei ryhdytty. Kuvassa 13 näytetään luvun 2 perään lisätty käsitelista.

Luvun 2 keskeiset termit:

- Vaatimusmäärittely: työvaihe, jossa mietitään ja listataan tulevalle sovellukselle halutut toiminnallisuudet ja muut vaatimukset
- Toiminnallisuus: toiminto, jonka sovelluksella voi tehdä
- Sovelluksen arkkitehtuuri: kuvaus sovelluksen eri tasoista, esim. tietokanta - koodi - käyttöliittymä
- Rajapinta: mahdollistaa tiedon siirtymisen oikein sovelluksen eri osien välillä
- Luokkakaavio: visuaalinen esitys sovelluksessa tarvittavista tiedoista ja niiden keskinäisitä suhteista
- Tietokanta: sovelluksen erillinen osa, jonne voidaan tallentaa sovelluksessa tarvittavia tietoja, kuten käyttätunnuksia
- Taulu: tietokannan osa, johon tallennetaan erillisille riveille rakenteeltaan samankaltaisia tietueita. Voidaan puhua myös luokista
- Viiteavain: taulun yksittäisellä rivillä sijaitseva tieto, joka viittaa toisen taulun tiettyyn riviin samassa tietokannassa
- Back end -koodi: sovelluksen "takaosan" koodi, liittyy yleensä tietokannan kanssa kommunikointiin
- Front end -koodi: sovelluksen "etuosan" koodi, tuottaa käyttöliittymän
- Olio: ohjelmoinnissa käytettävä tietotyyppi, jonka sisään voidaan tallentaa erilaisia muuttujia, eli tietoja. Voidaan puhua myös luokista, verrattavissa tietokantojen tauluihin
- Komponentti: sovelluksen koodiin tuleva rakenneosa (huomaa, että tämän oppaan ulkopuolella komponentti voi tarkoittaa myös muunkaltaista osaa jostain kokonaisuudesta)

Kuva 13. Testilukijoilta saadun palautteen perusteella lukujen 2 ja 3 loppuun lisättiin käsitelistat

4 Pohdinta

Opinnäytetyön tuotoksena syntyi 26-sivuinen opas. Opinnäytetyön tekijän oma arvio tuotoksesta on onnistunut, visuaalisesti miellyttävä, loogisesti etenevä ja helposti lähestyttävä opas. Prosessi eteni suurimmaksi osaksi jouhevasti ja aikataulussa. Lopullinen työn palautus myöhästyi opinnäytetyöntekijästä riippumattomista syistä. Vastaavan kaltaisiin myöhästymisiin olisi ollut hyvä varautua jo opinnäytetyön aikataulua laatiessa.

Tuotettu opas luetutettiin ennen palautusta myös testilukijoilla, joiden taustat eroavat opinnäytetyöntekijän taustasta. Testilukijoilta saatiin sekä positiivista palautetta että kehitysideoita. Koska palaute ei ollut pelkästään positiivista, vaan sisälsi myös rakentavaa kritiikkiä, voidaan todeta testilukijoiden kertoneen palautteissa rehellisiä mielipiteitään. Mielessä on silti pidettävä myös vaihtoehto, että osaa palautteesta on voitu kaunistella. Positiiviset kommentit keskittyivät keskenään samoihin asioihin ja myös kehitysideoilla oli selkeä teema eri henkilöiden vastauksissa. Tämä korostaa tunnetta, että palaute on ollut rehellistä, sillä valitut testilukijat eivät tunne toisiaan, eivätkä siten ole lukukokemuksen aikana keskustelleet toistensa kanssa.

Mikäli opinnäytetyön tekemiseen olisi ollut enemmän aikaa käytettävissä, testiluentaa olisi voinut tehdä laajemmallakin otannalla erilaisia testilukijoita. Hyvin rajallinen aika rajasi monia mahdollisia lukijoita pois. Useammalla testilukijalla palautteen voisi saada entistä luotettavammaksi. Lisäksi testilukijoissa olisi ollut hyvä olla myös sellaisia henkilöitä, joilla on ohjelmointitaustaa. Tämänkaltaisilta lukijoilta olisi voinut saada hyviä näkemyksiä siitä, millaisia asioita he olisivat kaivanneet aikanaan oman ensimmäisen sovelluksen suunnitteluun. Jälkikäteen tällaisia asioita on usein helpompi hahmottaa kuin vasta-alkajana ilman kokemusta. Tällainen palaute olisi ollut arvokasta opinnäytetyön laadun arviointiin.

Opinnäytetyön tietoperusta tehtiin ammattikirjallisuutta hyödyntäen ja mahdollisimman tuoreita lähteitä käyttäen. Vaatimusmäärittely tai ohjelmistojen mallintaminen eivät ole kokeneet mullistavia uudistuksia viime vuosina, joten niiden lähteissä hieman vanhemmatkin materiaalit olivat hyväksyttävissä. Käyttöliittymän suunnittelussa haluttiin varmistaa ajantasainen tieto, joten uusin lähde on vain vuoden vanha. Sen tietoja verrattiin vanhempiin lähteisiin, jonka perusteella vanhempien lähteiden luotettavuutta ja ajantasaisuutta arvioitiin. Yhdessä nämä vanhemmat ja tuore lähde antoivat tietopohjaan mielenkiintoisia ja vain pieneltä osin toisistaan eroavia näkemyksiä. Tietoperustan lähteiden valintaan käytetty harkinta parantavat opinnäytetyön tulosten luotettavuutta, sillä tuotettu teos on perusteltavissa ohjelmistokehitysalan ammattilaisten mielipiteillä ja ammattitaidolla.

Suurin haaste opinnäytetyön tekemisessä oli tiukka aikaraja. Kuten jo aiemmin on mainittu, se aiheutti konkreettista haittaa, kun testilukijoiden värvääminen hankaloitui ja lopulta työ hieman

myöhästyi aikatauluista. Tämä toimii hyvänä opetuksena opinnäytetyön tekijälle, että aikatauluja laatiessa tulee ottaa huomioon oman työskentelyn lisäksi projektin ulkopuoliset tekijät ja sidosryhmät. Tässä projektissa siinä ei onnistuttu.

Opinnäytetyö opetti tekijälleen myös graafista puolta, kun oppaan visuaalista toteutusta alettiin valmistamaan. Tekijällä ei ole aiempaa kokemusta näin laajan visuaalisen kokemuksen tekemisestä, joten haasteita projektissa riitti. Oppaan erilaisten kuvien aiheiden harkinta ja päättäminen veivät osaltaan ison osan opinnäytetyöhön käytetystä ajasta. Oli mielenkiintoista pohtia, miten eri teemoja voisi kuvilla selventää mahdollisimman hyvin. Testilukijoilta saadun palautteen perusteella tässä harkinnassa on onnistuttu ja se on kenties opinnäytetyön suurin onnistuminen.

Opinnäytetyö onnistuu kehittämishankkeena, sillä se tuo saataville jotain alalle uutta; oppaan juuri ensimmäisen oman mobiilisovelluksen suunnitteluun. Aihe on herättänyt kiinnostusta opinnäytetyöntekijän tuttavapiirissä muidenkin alojen opiskelijoissa ja saanut kiitosta IT-alan ammattilaisilta hyvästä ja mielenkiintoisesta valinnasta.

Projekti jättää myös kehittämistarpeita tulevaisuuteen, sillä opasta voisi jatkaa tulevaisuudessa monellakin eri tavalla. Jatko-osan oppaalle voisi kirjoittaa esimerkiksi siitä, kuinka tekoälyä voisi hyödyntää mobiilisovellusten suunnittelussa. Lisäksi mitä tahansa tuotetussa oppaassa käsiteltyä aihetta voisi laajentaa ja käsitellä syvällisemmin omassa teoksessaan. Erityisesti saavutettavuus mobiilisovelluksissa kaipaisi lisäpanostusta ja voisi toimia hyvänä aiheena jollekin seuraavalle opiskelijalle opinnäytetyöhön.

Lähteet

Ammattikorkeakouluun. s.a. Opiskelu. Luettavissa: https://www.ammattikorkeakouluun.fi/opiskelu/. Luettu: 24.4.2024.

Ballard, B. 2007. Designing the Mobile User Experience. John Wiley & Sons, Ltd. West Sussex.

Bennett, S. 2015a. UML Fundamentals. Class Diagrams. Associations. Katsottavissa: https://learning.oreilly.com/course/uml-fundamentals/9781771373630/. Katsottu: 22.4.2024.

Bennett, S. 2015b. UML Fundamentals. Class Diagrams. Attributes And Operations - Part 1. Katsottavissa: https://learning.oreilly.com/course/uml-fundamentals/9781771373630/. Katsottu: 22.4.2024.

Bennett, S. 2015c. UML Fundamentals. Class Diagrams. Distinction Between Objects And Classes. Katsottavissa: https://learning.oreilly.com/course/uml-fundamentals/9781771373630/. Katsottu: 22.4.2024.

Duffy, J. 29.9.2021. Visme – A one-stop shop for creating branded assets. PC Mag. Luettavissa: https://uk.pcmag.com/old-collaboration/132772/visme. Luettu: 20.4.2024

Filipova, O. & Vilão, R. 2018. Software Development From A to Z. Apress. New York.

Koskinen, I. 26.3.2021. Mikä on Kanban? Katsaus menetelmään ja sen käyttöön ketterässä projektinhallinnassa. Severa by Visma blogi. Luettavissa: https://severa.fi/blogi/mika-on-kanban-katsaus-menetelmaan-ja-sen-kayttoon-ketterassa-projektinhallinnassa/. Luettu: 10.4.2024.

Luukkainen, M. & Laine, H. 2010. Luentomoniste kurssille Ohjelmistojen mallintaminen. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. Helsinki. Luettavissa: https://www.cs.helsinki.fi/u/mluukkai/ohmas10/ohma.pdf. Luettu: 4.4.2024.

Melvin, J. 10.1. 2024. Visme vs Canva – Which is Better in 2024? Luettavissa: https://www.wyzowl.com/visme-vs-canva/. Luettu: 20.4.2024.

Nielsen, J. & Budiu, R. 2013. Mobile Usability. New Riders. Berkeley.

Paris, S. 28.2.2023. Visme review. Luettavissa: https://www.techradar.com/reviews/visme-review. Luettu: 20.4.2024.

Shelton, B. 11.8.2023. UML class diagrams. Video. Katsottavissa: https://www.youtube.com/watch?v=6XrL5jXmTwM. Katsottu: 22.4.2024.

Sinkkonen, I., Nuutila, E. & Törmä, S. 2009. Helppokäyttöisen verkkopalvelun suunnittelu. Tietosanoma. Helsinki.

Statista 2024. Number of mobile app downloads worldwide from 2016 to 2023. Luettavissa: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. Luettu: 5.4.2024.

Sutton, T. 2020. The Pocket Complete Color Harmony. Rockport Publishers. Gloucester, MA. Luettavissa: https://learning.oreilly.com/library/view/the-pocket-complete/9781631599217/. Luettu: 15.4.2024.

Visme s.a. About Visme – Welcome to our World! Luettavissa: https://www.visme.co/about/. Luettu 11.4.2024.

Volle, A. 1.3.2024. App – mobile device software. Luettavissa: https://www.britannica.com/technology/mobile-app. Luettu 5.4.2024.

Wells, M. 2023. User Experience Design – An Introduction to creating interactive digital spaces. Laurence King Publishing. Lontoo.

Liitteet

Liite 1. Ideasta toteutukseen – opas ensimmäisen mobiilisovelluksen suunnitteluun

Sisällysluettelo

Johdanto

Tervetuloa ohjelmistosuunnitelun maailmaan!

Oletko juuri keksinyt aiheen mahtavalle mobiilisovellukselle ja olet aloittamassa ensimmäisen oman mobiilisovelluksesi tekemistä? Mahtavaa, tämä opas on tarkoitettu juuri sinulle! Ennen kuin säntäät koodieditorisi kimppuun, puhutaan hetki ohjelmistosuunnittelusta.

Suunnittelu on tärkeä vaihe ohjelmistoprojekteissa. Vaikka se voikin tuntua alkuun tylsältä, tai jopa turhalta, kannattaa antaa aiheelle hetki aikaa ja keskittyä huolella pohtimaan, millaisen sovelluksen haluat toteuttaa. Tekemällä suunnitelman projektillesi saat aikaan selkeät suuntaviivat, joita seurata, kun aloitat itse koodaamisen. Erityisesti ensimmäisissä laajemmissa projekteissa suunnitelman tekeminen kannattaa, koska muutoin koodista ja sovelluksen arkkitehtuurista voi tulla hyvinkin järjestämätön ja kaoottinen. Tämä voi puolestaan johtaa erilaisiin ongelmiin, pahimmillaan projektin epäonnistumiseen.

Tässä oppaassa käydään läpi erilaisia ohjelmistosuunnittelun vaiheita, joita sinunkin kannattaa pohtia ja soveltaa omaan projektiisi. Opas ei ole kaikenkattava tietoteos, etkä sen luettuasi ole valmis ohjelmistoarkkitehdin töihin. Kuitenkin opas antaa sinulle hyvät ohjeet ensimmäisen sovelluksesi suunnitteluun ja toivottavasti herättää kiinnostuksen opiskella aihetta lisää.

Lähdemme oppaassa liikkeelle vaatimusmäärittelystä, eli vaiheesta, joka seuraa ohjelmistoidean keksimistä. Vaatimusmäärittelyn jälkeen siirrymme ohjelmiston mallintamiseen ja siitä käyttöliittymän suunnitteluun. Lopuksi saat vielä jotain vinkkejä siihen, mitä kannattaa ottaa huomioon sovelluksen toteutukseen siirtyessä.

Jotta tylsiltä kuulostavat termit tulisivat helpommin ymmärrettäviksi, on niitä käsitelty tässä oppaassa käytännön esimerkkien kautta. Oppaassa käytetyt esimerkit koskevat mobiilisovellusta elokuvaharrastajille. Vaikka oma sovellusideasi eroaisi käytetystä esimerkistä, voit soveltaa kaikkia oppaassa käytettyjä suunnittelun vaiheita omaan projektiisi.

Tämä opas on tuotettu opinnäytetyönä Haaga-Helia ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmassa. Oppaan tietopohjan tekemiseen käytetyt lähteet listataan oppaan lopussa.

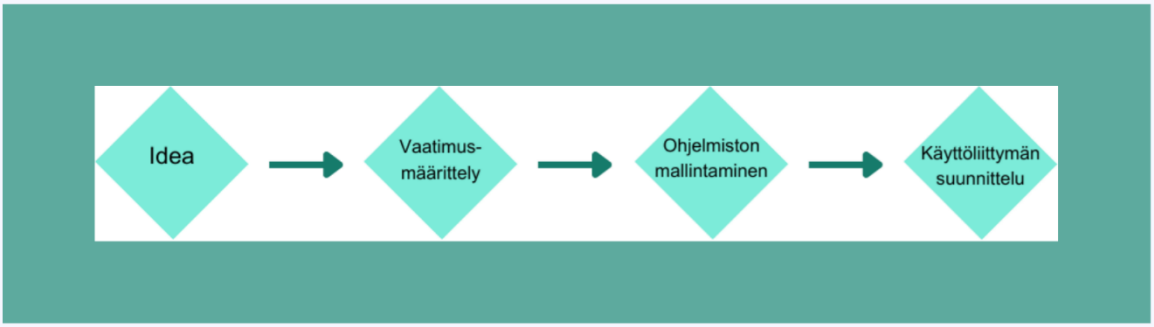
Vaatimusmäärittely ja mallintaminen

mitä sovellukseen tarvitaan ja kuinka se toteutetaan

Olet saanut loistavan idean omaan mobiilisovellukseesi – ja olet heti valmis toteuttamaan sen? Ehkäpä kuitenkaan et. Alkuinnostuksen kanssa voi tuntua hyvältä idealta hypätä heti toteutuksen pariin, mutta todellisuudessa se ei ole paras tapa uuden ohjelmiston luomiseen. Annapas, kun selitän.

Olet saanut idean; mobiilisovellus elokuvien ystäville, jotka haluavat viestitellä samasta aiheesta kiinnostuneiden henkilöiden kanssa. Loistava idea ja kertoo kyllä jotain sovelluksen tarkoituksesta. Mutta kuvaileeko se sovelluksen ominaisuuksia niin yksiselitteisesti, että osaisit tuottaa sen pohjalta koodia? Kertooko idea, millaisia viestejä on mahdollista lähettää tai varmennetaanko sovelluksessa mitenkään, kuka on viestin lähettänyt? Entä onko palvelussa muita toimintoja kuin viestien lähetys?

Kun ideaa pohditaan hetki, huomataan että itseasiassa aiheeseen liittyy vielä reilusti vastaamattomia kysymyksiä. Onneksi on olemassa helppoja prosesseja, joiden avulla näihin kysymyksiin saadaan vastauksia. Ja toisaalta keksitään taas uusia kysymyksiä – ja löydetään niihin vastauksia. Ja huomataan uusia kysymyksiä... Lopulta näiden kysymysten vastauksista syntyy ohjelmisto-suunnitelma, joka toimii suuntaviittanasi, kun siirryt toteuttamaan sovellustasi.

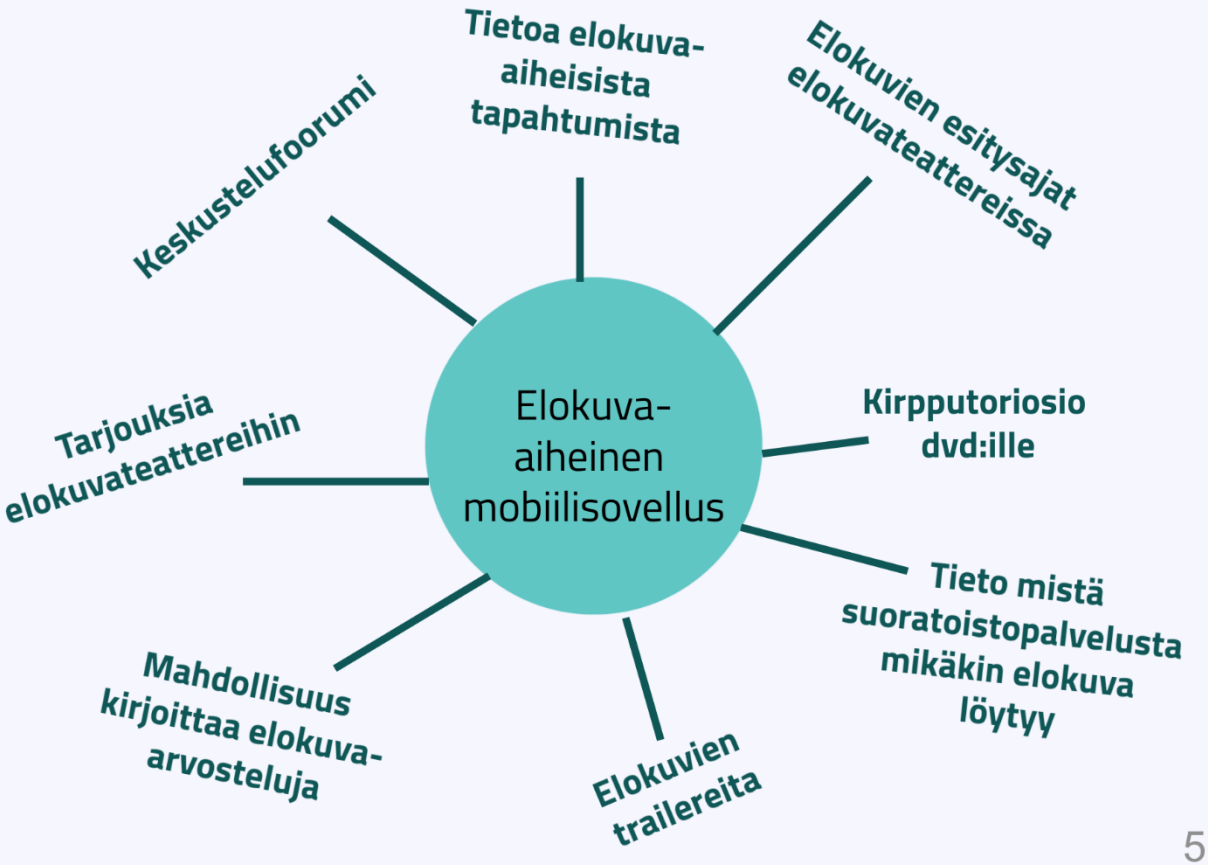


Aloitamme suunnitellun vaatimusmäärittelystä. Kuten nimikin sen jo kertoo, alamme siis määrittelemään vaatimuksia uudelle sovelluksellemme.

Perinteisessä ohjelmistosuunnitelmassa määritettään useita erilaisia vaatimuksia, kuten toiminnallisuuksiin, käytettävyyteen, turvallisuuteen, käyttäjiin tai tietoihin liittyviä vaatimuksia. Lisäksi vaatimusmäärittelyssä pohditaan mahdollisia toimintaympäristön ja teknologian aiheuttamia rajoituksia.

Meidän työmäärämme ei onneksi ole yhtä valtava, sillä me suunnittelemme vasta ensimmäistä sovellustamme, emmekä tähtää ammattitason määritelmiin. Me selviämme määrittelytyöstä sillä, että määrittelemme toiminnalliset vaatimukset.

Sovelluksen toiminnalliset vaatimukset vastaavat kysymykseen ”mitä”. Aloitamme suunnittelu-urakan pohtimilla, mitä tarkalleen haluamme elokuva-aiheisen sovelluksemme tekevän. Apuna tässä pohdinnassa voi käyttää vaikkapa ajatuskarttaa.



Ajatuskartasta käy ilmi useita sovellukseen sopivia toimintoja. Kuitenkaan sovelluksen toiminnallisuudet eivät lopu niihin. Kun katsomme ideoitamme tarkemmalla silmällä, huomaamme siellä turvallisuusriskejä. Keskustelufoorumi tai kirpputori ilman minkäänlaista valvontaa voisi mahdollistaa esimerkiksi internet-kiusaamista tai huijausyrityksiä. Tarvitsisimme siis myös 1. mahdollisuuden puuttua huonoon tai epäilyttävään käytökseen ja 2. tiedon, kuka viestejä lähettää. Tästä löysimme taas kaksi uutta toiminnallisuutta sovellukseemme.

Osa ajatuskartan ideoista taas tuntuu kovin korkealentoisilta ensimmäiseen sovellukseemme. Mikä elokuvateatteri suostuisi antamaan tarjouksia sovellukselle, jolla ei ole vielä yhtäkään käyttäjää? Korkealentoisetkaan ideat eivät kuitenkaan haittaa, sillä kaikkia ideoita ei tarvitse ottaa mukaan toteutukseen. Voimme jaotella ideat sen mukaan, mitä haluamme lähteä heti toteuttamaan, mitkä laitamme säästöön kehitysideoiksi ja mitkä laitamme roskiin, esimerkiksi vastuullisuusriskien vuoksi.

Vaikka nyt hylkäisimmekin jonkin idean, se ei tarkoita, että kyseinen idea olisi ikuisesti roskakorin vanki. Mikäli idea on jonnekin kirjoitettu ylös (tai vain satumme muistamaan sen vuosien päästä ilman muistiinpanoja), voimme palata sen pariin myöhemmin.

Homma valmis ja eteenpäin? Ei ihan vielä. Vaatimusmäärittelyn tuloksen tulisi olla selkeästi kuvailtu ja ristiriidaton lista vaatimuksista. Esimerkiksi keskustelufoorumia ei ole vielä lainkaan kuvailtu, eikä se tällaisenaan kerro juuri mitään toivotuista toiminnallisuuksista. Käytetään siis vielä hetki ja tarkennetaan näiden olemassa olevien vaatimusten kuvailua.

Lopulta saamme valmiiksi listan vaatimuksista, jotka kuvailevat vaatimuksiamme ja niiden vaikutuksia mahdollisimman yksityiskohtaisesti:

- Sovellukseen kirjaudutaan sisään henkilökohtaisella käyttäjätunnuksella ja salasanalla
- Käyttäjätunnuksella voi olla eri tasoisia oikeuksia: peruskäyttäjä tai moderaattori
- Peruskäyttäjä voi lukea ja lähettää viestejä keskustelufoorumilla. Lisäksi omia viestejä voi muokata tai poistaa
- Moderaattori voi poistaa asiattomia viestejä sekä määrätä peruskäyttäjiä viestienlähettämiskieltoon
- Keskustelualueen viestit voivat sisältää tekstiä tai kuvia
- Videoiden liittämistä viesteihin ei tueta
- Jos viestiketjusta poistetaan viesti, sen tilalle tulee ilmoitus "tämä viesti on poistettu käyttäjän/ylläpidon toimesta"
- Viestin poistaminen ketjusta ei vaikuta muihin ketjun viesteihin, ei edes aloitusviestin ollessa kyseessä
- Jos moderaattori poistaa käyttäjän viestin tai arvostelun, siitä lähtee ilmoitus käyttäjälle
- Sovellus käyttää tietokantaa, josta löytyy erilaisten elokuvien perustiedot
- Käyttäjät voivat kirjoittaa elokuvista arvosteluita, joita pääsee tarkastelemaan elokuvan tietosivulta
- Elokuvan tietosivulta löytyy myös arvosteluiden keskiarvo (1-5 tähteä asteikolla)
- Moderaattorit voivat poistaa asiattomat arvostelut
- Jos arvostelu poistetaan, myös sen numeraalinen arvo poistetaan keskiarvosta
- Arvostelun poistamisesta ei jää merkintää
- Elokuvien tietosivu sisältää myös elokuvan trailerin
- Jos elokuvan traileria ei ole saatavilla, sen tilalla näkyy sivulla viesti aiheesta
- Käyttäjä voi lisätä itselleen profiilikuvan, joka näkyy sovelluksessa nimimerkin yhteydessä
- Käyttäjän on mahdollista muokata omia tietojaan sovelluksessa

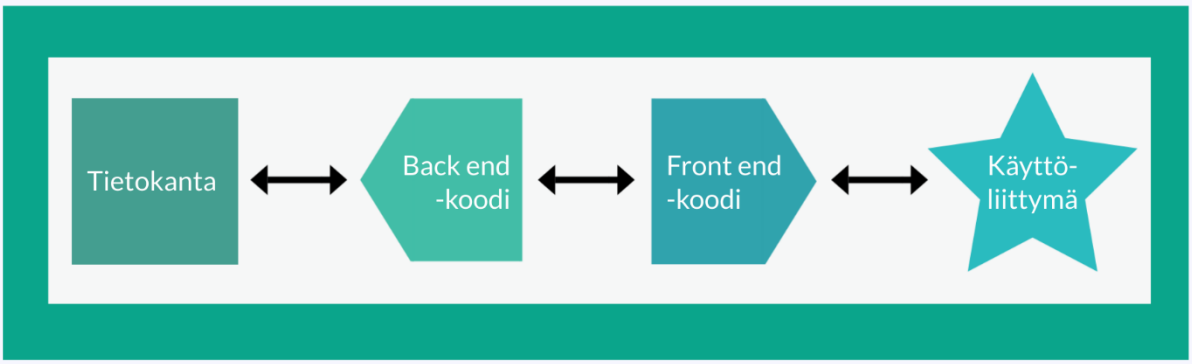
Tätäkin vaatimuslistaa voisi vielä hioa, mutta emme pyri täydellisyyteen ensimmäisen sovelluksemme kohdalla. Jos vain täysin virheetön kelpaa, et tule koskaan valmiiksi. Eikä tule projektisikaan.

Nyt, kun olemme saaneet sovelluksemme toiminnalliset vaatimukset määriteltyä, voimme siirtyä suunnittelemaan itse sovellusta. Aloitamme hahmottelemalla sovelluksen arkkitehtuuria. Koska sovellus on ensimmäinen projektimme, emme sukella suoraan sovellusarkkitehtuurin syvään päätyyn, vaan pohdimme melko pinnallisesti eri osioita, joita sovelluksemme tarvitsee.

Tutkitaan taas vaatimuslistaamme. Haluamme toteuttaa sovellukseen mm. keskustelualueen, jonka kautta eri käyttäjät voivat viestiä toisilleen. Tarvitsemme tätä varten keinon tallentaa käyttäjien tietoja sekä viestejä, eli siis tietokannan. Tietokanta toimii arkkitehtuurissamme ns. 'takimmaisena' osana, jonka päälle tuodaan muut sovelluksen kerrokset, kunnes meillä on valmis mobiilisovellus.

Tässä vaiheessa meidän ei tarvitse vielä pohtia, kuinka tuo tietokanta toteutetaan. Teknisten vaihtoehtojen selvittäminen ja valinta tapahtuu vasta, kun sovelluksen suunnitteluvaihe on valmis.

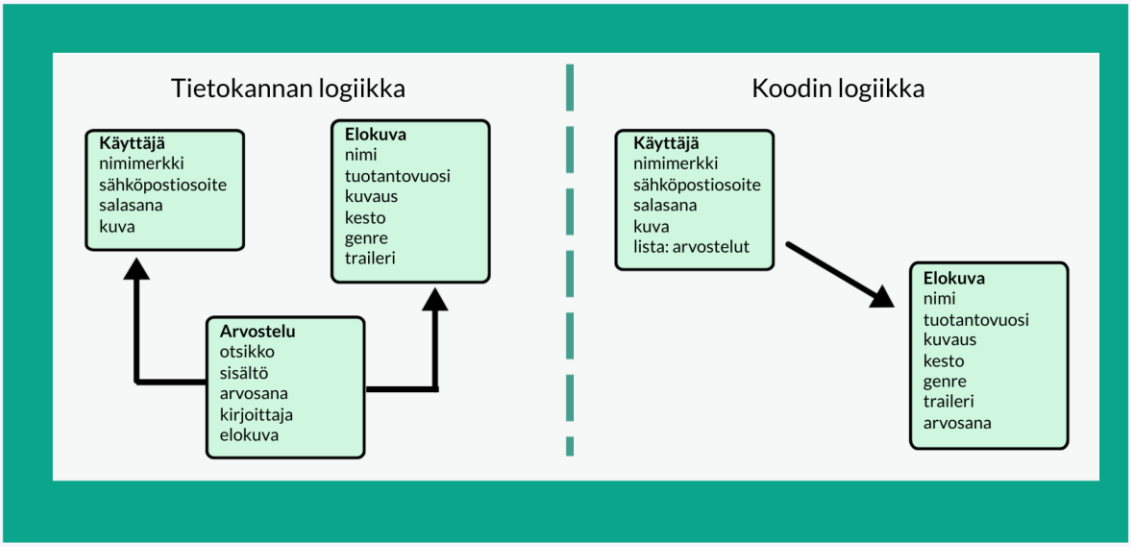
Jos tietokanta on sovelluksemme takimmainen osa, niin sen päällimmäiseksi kerrokseksi tulee puolestaan käyttöliittymä. Käyttöliittymä mahdollistaa nimensä mukaan sovelluksen käyttämisen ja on ainut osa, joka on selvästi näkyvillä sovelluksen käyttäjille. Käyttöliittymän ja tietokannan väliin tarvitsemme ohjelmistokoodin, jolla voimme toteuttaa käyttöliittymän ja käsitellä tietokantaa. Tuo ohjelmistokoodi voidaan tarvittaessa jakaa vielä kahteen osaan; sovelluksen takaosan kanssa toimivaan back end -koodiin sekä käyttöliittymän tuottavaan front end -koodiin.



Arkkitehtuurin piirtäminen auttaa hahmottamaan sovelluksen eri tasoja ja niiden välisiä suhteita. Mustat nuolet kuvaavat tiedonkulkua tasojen välillä.

Hahmotellessamme sovelluksen arkkitehtuuria huomaamme tarvitsevamme sovellukseen erilaisia tasoja ja niiden välille yhteyden, jotta tieto kulkisi sovelluksessa oikein ja luotettavasti. Eri tasojen välissä onkin rajapintoja, joiden kautta voimme siirtää tietoa tasolta toiselle. Saadaksemme tästä tiedonsiirrosta mahdollisimman sujuvaa ja koodista helposti tuotettavaa on tärkeää pohtia, miten tiedot muotoillaan ja kuinka sitä voidaan käsitellä. Jos emme käytä aikaa tiedon ja tietotyyppien miettimiseen, kasvaa riski siihen, että tietoa ei saada välitettyä oikein. Tämä riski suurenee entuudestaan, mikäli sovelluksen parissa työskentelee useita henkilöitä, jotka toteuttavat eri vastuualueiden tehtäviä.

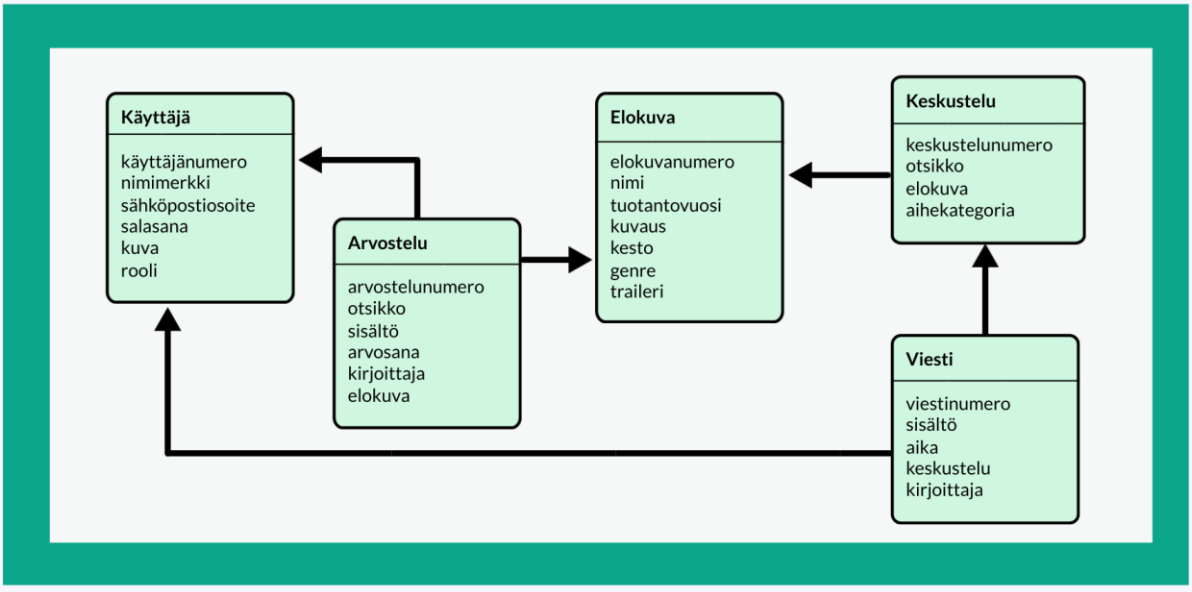
Mietitään vaikkapa tilanne, jossa tietokannan on suunnitellut yksi henkilö ja sovelluksen back end -koodin tuottaa toinen henkilö. Tietokannan parissa työskentelevän mielestä on loogista käsitellä käyttäjiä, elokuvia ja niiden arvosteluita erillisinä tietueina, jotka sijoitetaan tietokannassa erillisille tauluille. Taulujen välille luodaan viittauksia muihin tauluihin, jolloin erillisellä kyselyllä voidaan yhdistää tiettyyn arvosteluun sen kirjoittanut käyttäjä sekä siihen liittyvä elokuva. Back end -koodin parissa työskentelevä taas kokee loogiseksi, että elokuvat ja käyttäjät ovat erillään, mutta kirjoitetut arvostelut tallennetaan aina käyttäjän tietojen yhteyteen ja elokuvan saama arvosana on muiden elokuvan tietojen kanssa yhdessä.



Koodissa voidaan käyttää erilaista logiikkaa tietojenkäsittelyn suhteen kuin tietokannoissa, mutta se vaatii, että tekijä tietää kuinka tieto on tallennettu ja miten sen voi muokata ohjelmassa haluamaansa muotoon. Tämän vuoksi on tärkeää käyttää aikaa suunnitteluun ennen toteutukseen siirtymistä.

Aloitamme luokkakaavioiden tekemisen erottelemalla tarvittavien tietojemme pääaiheet eli luokat. Osa luokista voi olla helpommin eroteltavissa kuin toiset. Sovelluksessamme on selkeää, että yksi luokka on käyttäjät ja toinen elokuvat. Kun tarkastelemme muita tarvitseviamme tietoja, huomaamme että osa liittyy pelkästään toiseen luokkaan (esim. käyttäjän nimimerkki) ja osa selvästi molempiin (käyttäjä voi kirjoittaa arvostelun, arvostelu liittyy aina johonkin elokuvaan). Ne tiedot, jotka liittyvät vain yhteen aiempaan luokkaan, kuuluvat kyseisen luokan alle. Jos taas tieto liittyy molempiin luokkiin, se erotellaan omaksi luokakseen. Tämä mm. helpottaa tiedonkäsittelyä tietokannassa.

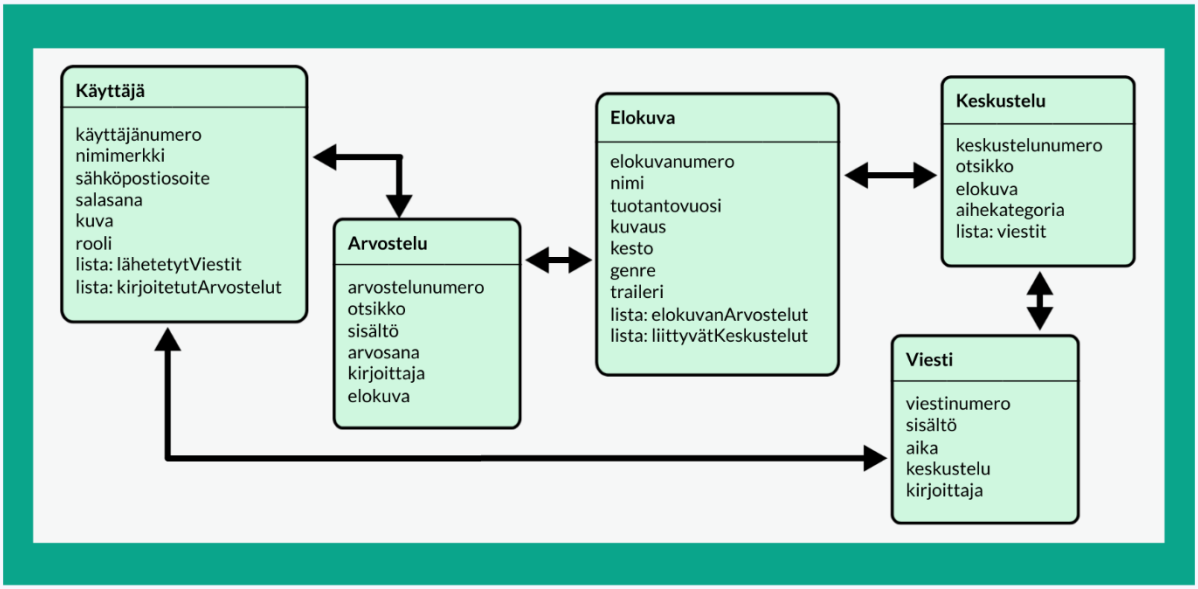
Katsoessamme aiemmin tekemäämme vaatimusten listaa ja pohtiessamme niiden tarvitsemia tietoja, voimme erottaa viisi erillistä luokkaa; käyttäjä, elokuva, arvostelu, keskustelu ja viesti. Voimme siis aloittaa luokkakaavion tekemisen, luomalla kaavion, jossa on viisi erillistä osiota. Sen jälkeen mietimme jokaiseen osioon erikseen, minkälaisia tietoja ne sisältävät. Nämä luokkien alle sijoittuvat tiedot ovat muuttujia, sillä niiden sisältämät arvot muuttuvat tapauskohtaisesti. Nimeämme nämä muuttujat kuvaavalla tavalla kaavioomme. Viimeiseksi lisäämme eri luokkien välille nuolet kuvaamaan niiden yhteyksiä toisiinsa.



Jokaisesta luokasta löytyvä numero-muuttuja on lisätty tietokantaa varten. Tietokannoissa jokaisen taulun jokaisella rivillä tulee olla yksilöllinen, uniikki tunniste. Juokseva numerointi toimii siihen tarkoitukseen erinomaisesti. Kuvan nuolet kuvaavat missä taulussa on viiteavain, eli mistä viitaus lähtee, ja mihin tauluun sillä viitataan. Oikeaoppisissa UML-kaavioissa yhteysviivoja voidaan merkitä eri tavoin.

Seuraavaksi lähdemme suunnittelemaan ohjelmointia. Koodissa tietoa käsitellään usein hieman eri tavalla kuin tietokannoissa, joten aloitamme pohtimalla, kuinka tietokantamme tiedot toimisivat ohjelmoinnissa. Koodatessa emme puhu enää tauluista, vaan yleensä eri luokista puhutaan oliomuuttujina. Koodissa siis jokaista tietokannan taulua vastaamaan luodaan oma olioluokka, jonka kautta tietoja käsitellään. Luokkien nimet ja muuttujat pysyvät lähes muuttumattomina.

Kaaviota tulee kuitenkin hieman täydentää. Tietokannassa taulujen väliseen yhteyteen riittää, että toisessa taulussa on muuttuja, joka toimii viiteavaimena ja viittaa liittyvään taulun johonkin riviin. (Esim. Tekemässämme kaaviossa Viesti-taulussa oleva kirjoittaja-muuttuja viittaa Käyttäjä-taulun käyttäjänumero-muuttujaan.) Ohjelmoinnissa lisäämme liittyvät tiedot listana toiseen oliomuuttujista. Luomme siis Käyttäjä-olion, jonka yksi muuttuja on viestit-niminen lista, joka puolestaan sisältää kaikki ne viestit, jotka tietokannassa viittaavat kyseiseen käyttäjään. Teemme vastaavan lisäyksen jokaiseen olioon, johon viitataan toisesta luokasta.



Tämä päivitetty luokkakaavio toimii suunnitelmanamme ohjelmointiin liittyviin luokkiin. Nyt meillä on valmiina suunnitelmat tietokannan rakenteista ja koodimme oliomuuttujista. Näiden perusteella voimme luoda tietokantaan oikean määrän tauluja ja nimetä niihin attribuutit sekä luoda koodissamme oliomuuttujat omine attribuutteineen. Tiedämme myös, mistä tauluista haluamme hakea olioihin liittyviä tietoja lista-attribuutteihin.

UML-kaavioita voi tehdä erilaisilla sovelluksilla tai käsin piirtäen. Kenties helpoin tapa tuottaa kaavioita tietokoneella on Microsoft Vision avulla. Myös kuvankäsittelyohjelmat toimivat kaavioiden tekoon, mutta niillä työskentely on usein hitaampaa.

Voimme vielä käyttää hetken aikaa pohtiessamme alustavaa komponenttilistaa, jotta hahmotamme, kuinka voimme tuottaa koodia hallittavissa osioissa. Ensimmäiset rakenneosamme voimme päätellä luokkakaaviostamme. Kun luokkaan liittyy jonkinlainen lista, sen esittäminen käyttöliittymässä edellyttää kaikkien sen osien näyttämistä. Yleensä listan osat halutaan näyttää samankaltaisina, joten voimme luoda koodissa erillisiä komponentteja, jotka tuottavat listan muuttujille halutun esitysasun.

Näin ollen meillä on ainakin neljä erilaista komponenttia; Elokuva, Arvostelu, Keskustelu ja Viesti -komponentit. Mikäli haluamme viestien ja arvosteluiden näkyvän eri tavoilla riippuen siitä, minkä luokan yhteydessä ne esitetään, erilaisia komponentteja voi olla myös 6. Tähän lisäämme vielä navigaation, jolloin komponenttilistamme on valmis:

- Keskustelulistaus
- Viestiketju
- Elokuva
- Elokuvan arvostelut
- Käyttäjän viestit
- Käyttäjän tekemät arvostelut
- Navigaatio

Elokuva-käsitettä käyttäessä meidän pitää aina huomioida, puhummeko yhden elokuvan tiedoista vai kaikkien tietokannasta löytyvien elokuvien listasta. Käytetään siis listasta monikkomuotoa 'Elokuvat' ja tietyn elokuvan tiedoista puhuttaessa yksikkömuotoa 'Elokuva'.

Vaikka käytimmekin komponenttilistasta sanaa 'valmis', muokkaamme sitä todennäköisesti vielä, kun pääsemme toteuttamaan sovellusta. Ensimmäistä sovellusta tehdessä emme välttämättä tiedosta etukäteen kaikkia tarvitsemiamme komponentteja. Tämä ei kuitenkaan haittaa. Voimme kuitenkin hyödyntää nykyistä listaamme suunnittelun seuraavassa vaiheessa, kun alamme hahmottelemaan sovelluksemme käyttöliittymää.

Luvun 2 keskeiset termit:

- Vaatimusmäärittely: työvaihe, jossa mietitään ja listataan tulevalle sovellukselle halutut toiminnallisuudet ja muut vaatimukset
- Toiminnallisuus: toiminto, jonka sovelluksella voi tehdä
- Sovelluksen arkkitehtuuri: kuvaus sovelluksen eri tasoista, esim. tietokanta - koodi - käyttöliittymä
- Rajapinta: mahdollistaa tiedon siirtymisen oikein sovelluksen eri osien välillä
- Luokkakaavio: visuaalinen esitys sovelluksessa tarvittavista tiedoista ja niiden keskinäisitä suhteista
- Tietokanta: sovelluksen erillinen osa, jonne voidaan tallentaa sovelluksessa tarvittavia tietoja, kuten käyttätunnuksia
- Taulu: tietokannan osa, johon tallennetaan erillisille riveille rakenteeltaan samankaltaisia tietueita. Voidaan puhua myös luokista
- Viiteavain: taulun yksittäisellä rivillä sijaitseva tieto, joka viittaa toisen taulun tiettyyn riviin samassa tietokannassa
- Back end -koodi: sovelluksen "takaosan" koodi, liittyy yleensä tietokannan kanssa kommunikointiin
- Front end -koodi: sovelluksen "etuosan" koodi, tuottaa käyttöliittymän
- Olio: ohjelmoinnissa käytettävä tietotyyppi, jonka sisään voidaan tallentaa erilaisia muuttujia, eli tietoja. Voidaan puhua myös luokista, verrattavissa tietokantojen tauluihin
- Komponentti: sovelluksen koodiin tuleva rakenneosa (huomaa, että tämän oppaan ulkopuolella komponentti voi tarkoittaa myös muunkaltaista osaa jostain kokonaisuudesta)

Rautalangasta visuaalisuuteen
käyttöliittymän suunnittelu

Käyttöliittymän suunnittelu kannattaa tehdä vaiheittain, kuten mikä tahansa muukin suunnitteluprosessi. Lähdemme siis liikkeelle suurista kokonaisuuksista ja suunnitelmien edetessä teemme niistä yksityiskohtaisempia.

Käyttöliittymän suunnitelman toteutamme 3 vaiheessa:

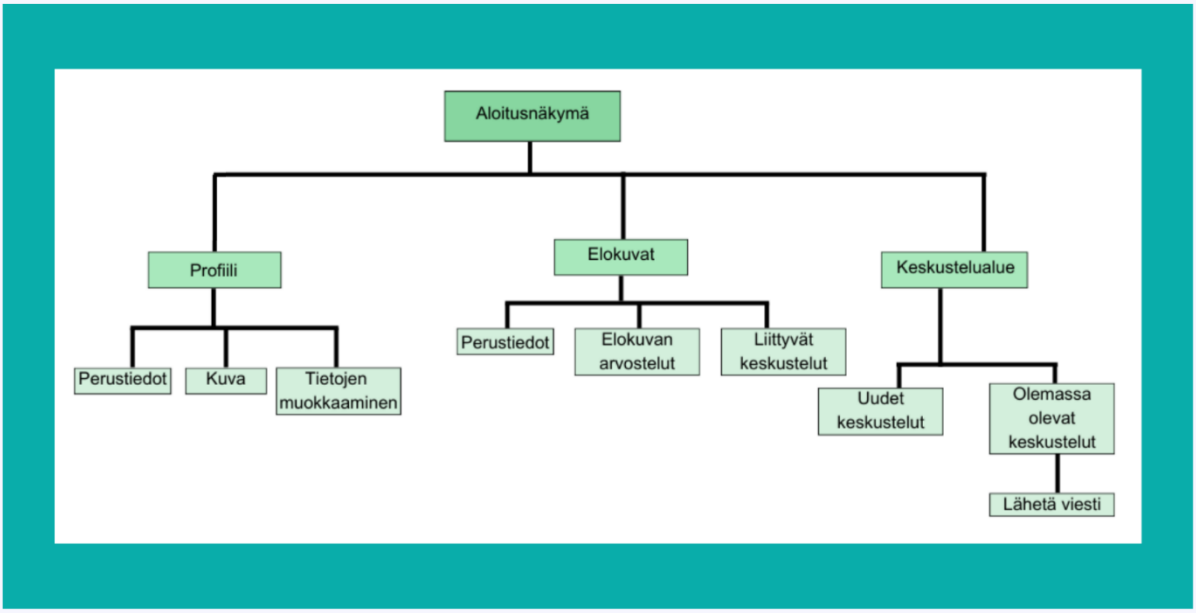
1. Sovelluskartta

2. Rautalankamalli

3. Tarkkapiirtoisempi prototyyppi

Sovelluskartan teko on yksinkertainen prosessi. Mietimme vain, mitkä ovat ne näkymät sovelluksessa, joita kävijä voi selata. Sitten laitamme ne hierarkkiseen järjestykseen. Lopuksi mietimme, mitkä ovat näkymistä löytyvät päätoiminnallisuudet.

Aloitusnäkymä (vertaa websivuilla koti-sivuun) tulee hierarkiassa ylimmäiseksi, sillä se eroaa käyttötarkoitukseltaan muista näkymistä. Aloitusnäkymä aukeaa käyttäjälle aina sovelluksen käynnistyessä, mutta sisältönsä puolesta sille ei ole tarvetta palata navigaation kautta (vaikka se mahdollista yleensä onkin). Muut näkymät valitsemme sen mukaan, millaisia toiminnallisuuksia sovelluksessamme on ja kuinka haluamme ne sovellukseemme ryhmittää.



Joskus voi olla vaikeampaa pohtia, mitkä ovat sovellukseen tarvittavat tai parhaimmat näkymät. Tällaisessa tilanteessa voimme pohtia asiaa sovelluksen käyttäjän näkökulmasta. Mieti, mitkä toiminnallisuudet liittyvät yhteen ja kuinka käyttäjä liikkuu näiden välillä. Käyttäjän liikkumisesta eri toiminnallisuuksien välillä voit johtaa näkymät, joissa turhat siirtymiset on minimoitu.

Sovelluskartan jälkeen on aika siirtyä pohtimaan erilaisia näkymiä tarkemmin. Millaisia komponentteja, eli rakenneosia, tarvitsemme ja kuinka voimme sijoittaa ne fiksusti eri näkymiin? Vaihtelevatko komponenttien paikat näkymän mukaan? Näihin kysymyksiin ei ole vain yhtä oikeaa vastausta, vaan saamme itse tehdä päätökset sen mukaan, minkä koemme parhaaksi sovelluksellemme tai sen käyttäjille.

Kun harkitsemme rakenteiden järjestystä, hahmottamistamme helpottaa sovelluksen rautalankamallit. Rautalankamallissa osoitamme eri osioiden järjestyksen karkeasti puuttumatta sovelluksessa käytettäviin väreihin, fontteihin tai muihin yksityiskohtiin. Mallissa sijoitamme komponentit paikoilleen esimerkiksi erilaisten ruudukoiden avulla. Sisältöön emme ota tässä vaiheessa vielä kantaa.

Voimme kuvata erilaisia komponentteja graafisilla kuvioilla tai tekstillä, joka itsessään ei tarkoita mitään tai liity sovelluksen sisältöön. Rautalankamallin tekoon ei ole vain yhtä oikeaa tapaa. Onnistuneeseen malliin riittää, että erilaiset komponentit erottuvat toisistaan. Alla esitetään muutamia esimerkkejä käytettävistä vaihtoehdoista.

Nelikulmiot, joille määritellään taustasta eroava pohjasävy, voivat kuvastaa rautalankamallissa kuvia tai videoita.

Navigaatio

Nelikulmiot, joille ei määritellä pohjasävyä, voivat kuvastaa painikkeita tai kiinteitä rakenteita. Niiden merkitystä voi selventää myös tekstillä.

Otsikko

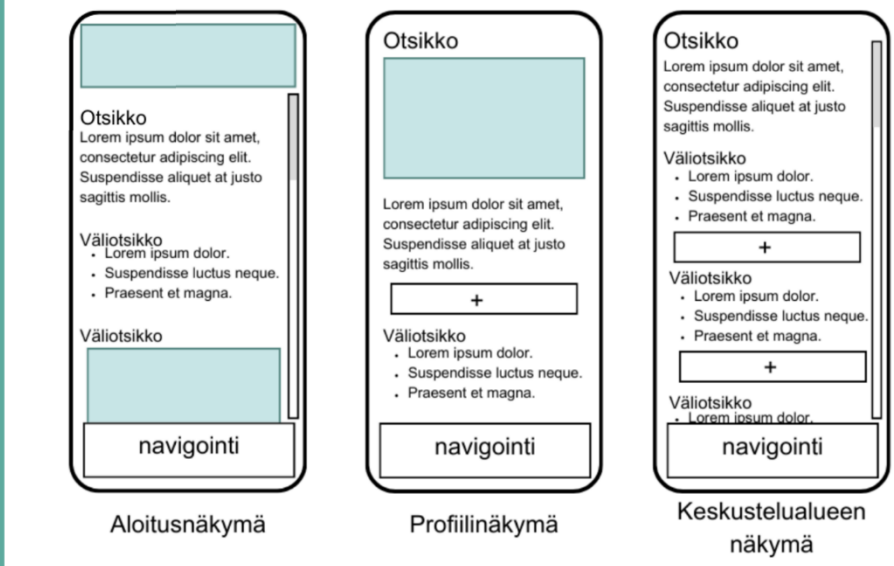
Väliotsikko

Otsikoiden ja väliotsikoiden paikat voi kuvata sanoilla. Ero fonttikoossa kertoo komponentin viemästä tilasta, ei käyttöön tulevien otsikoiden todellisista fonttiko'oista.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod

Tekstikappaleiden sijainnin voi esittää esimerkiksi Lorem ipsum -tekstillä, joka on laajasti käytössä oleva malliteksti, eikä itsessään tarkoita mitään.

Rautalankamallia tehdessä kannattaa aina pohtia, kuinka paljon komponenttien sijainti vaihtelee erilaisten näkymien välillä. Emme tee erillistä mallia jokaisesta näkymästä, vaan tuotamme rautalankamallit vain niistä näkymistä, joissa komponenttien sijainti on erilainen muihin näkymiin verrattuna.



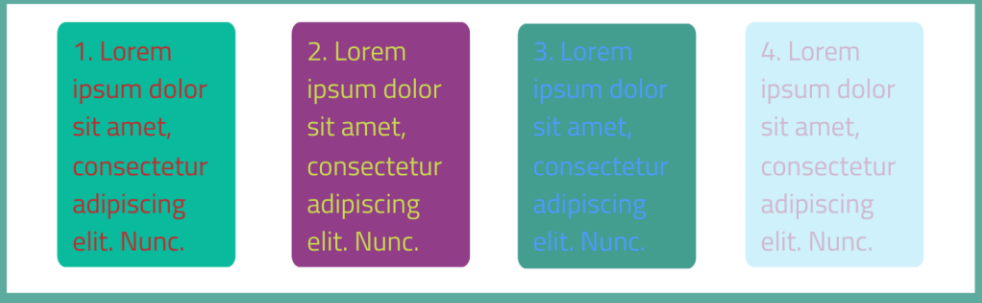
Rautalankamallin voit tuottaa lähes millä tahansa ohjelmistolla, joka soveltuu millään tasolla kuvankäsittelyyn. Kaikki vaihtoehdot Paintista aina ammattilaisten ohjelmistoihin asti toimivat. Voit piirtää yksinkertaiset mallit myös käsin, jos koet sen paremmaksi keinoksi itsellesi.

Kun rautalankamallimme on valmis, voimme siirtyä valmistamaan tarkkapiirtoisempaa prototyyppiä. Otamme käsittelyyn jonkin rautalankamallimme näkymän ja aloitamme muokkaamaan siitä tarkempaa. Etenemme vaihe kerrallaan tehden muokkauksia malliin, ja lopulta rautalankamallimme on muuttunut sovelluksen prototyypiksi.

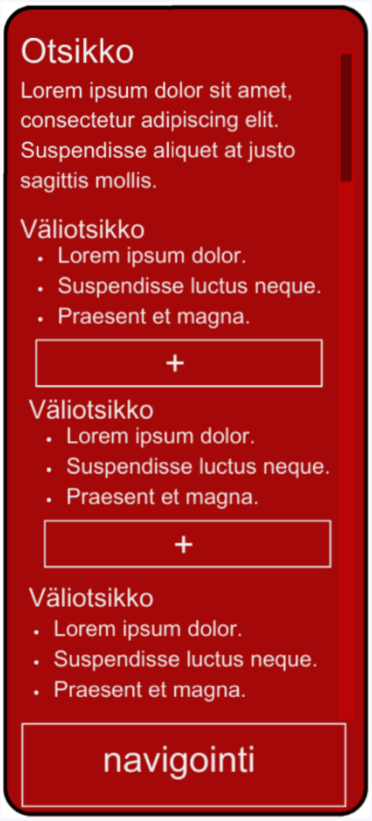
Mikäli haluat määritellä tässä kohtaa tarkat fonttikoot, työtäsi voi helpottaa se, että käytät pohjana kuvaa, joka on pikseleissä yhtä suuri kuin mobiililaitteesi näkymä. Näin näet heti, onko fonttikoko sellainen, että sen näkee lukea tai pystyykö navigaation eri kohteisiin koskemaan tarkasti. Oman laitteesi koon voit tarkistaa monesta eri palvelusta. Löydät nämä googlaamalla ”what is my screen resolution”.

Lähdemme ensin pohtimaan sovelluksemme värimaailmaa. Värejä valitessa on hyvä pohtia sovelluksen saavutettavuutta. Saavutettavuus tarkoittaa sitä, että erilaisten rajoitteiden kanssa toimivat pystyvät käyttämään samoja palveluita kuin ilman näitä rajoitteita toimivat henkilöt. Saavutettavuus on laaja aihe ja kattaa muutakin kuin värimaailman, mutta ensimmäisessä sovelluksessamme keskitymme huomioimaan sen väreissä.

Haluamme valita siis sellaisia väriyhdistelmiä, että esim. yleisimmät värisokeudet eivät estä sovelluksen käyttöä. Emme siis valitse sovelluksellemme vihreää taustaa ja punaista tekstiä, sillä se rajoittaisi käyttäjäkuntaamme kohtuullisen paljon. Lisäksi kiinnitämme huomiota kontrastiin, eli siihen, kuinka eri osissa käytettävät värit erottuvat toisistaan. Emme halua käyttää liian samankaltaisia värejä, jotta kaikki sisältö on selkeästi näkyvissä ja käytettävissä.



Yllä olevat esimerkit eivät ole saavutettavia väriyhdistelmiä. Ensimmäiset kolme voivat näyttää normaalin värinäön omaavista luettavilta, mutta punavihersokea ei näe ensimmäisen tekstejä, eikä sinisokea toisen tai kolmannen. Viimeisessä taas värit ovat samaa tummuusastetta, jolloin kontrastia ei ole tarpeeksi, eikä teksti erotu taustasta riittävästi.



Aiemmat rajoitteet huomioiden voimme valita värit sovellukseemme oman makumme mukaan.

Pohditaan sovelluksemme aihetta. Mitä asioita tai värejä elokuvista tulee mieleen? Pimeä elokuvateatteri ja lapsuudesta tutut punaiset penkit tai esirippu, jotka tuovat ajatuksiin innostuksen ja lämmön tunteita. Haluamme väreillämme herättää tunteita, mutta myös erottua tunnetuista, elokuviin liittyvistä brändeistä, joten vältämme valitsemasta värejä, jotka yhdistämme mielessämme tällaisiin tahoihin. Haemme siis värimaailmaamme tummaa, punaista ja teksteihin värejä, jotka erottuvat siitä. Testaamme eri vaihtoehtoja ja lopulta valitsemme malliimme värit.

Käyttämiemme värien heksadesimaalit eli värikoodit:
Tausta: #A40808
Teksti: #F1F1F1
Vierityspalkin tausta: #BB0505
Vierityspalkin liikkuva osa: #660505

Myös tarkkapiirtoisia prototyyppejä voi tehdä lähes millä tahansa kuvankäsittelyohjelmalla. Prototyyppien tekoon on kuitenkin olemassa myös erillisiä sovelluksia, esim. UI-suunnittelijoiden suosima Figma. Oppaassa tuotettava prototyyppi on tehty Canvalla.

Ennen kuin siirrymme seuraavaan vaiheeseen ja alamme valitsemaan fonttia sovellukseemme, voimme päivittää prototyypin tekstit vastaamaan paremmin tulevaa sisältöä. Näin näemme heti, kuinka valitsemamme fontti oikeasti toimii teemassamme.

Fontin voimme määritellä joko tarkasti haluamaamme fonttiin tai fonttiperheeseen, joka noudattaa tietynlaista tyyliä. Fonttiperheitä ovat:
Serif – kirjainten reunoista löytyy pienet viivat, muodollinen.
Sans-serif – kirjaimissa ei ole viivoja reunoissa, moderni.
Monospace – jokaisella kirjaimella on sama, kiinteä leveys, konemainen.
Cursive – muistuttavat käsinkirjoitusta.
Fantasy – koristeellisia tai leikkimielisiä fontteja.

Fonttia ja fonttikokoa testatessa kannattaa huomioida, että eri fontit voivat näyttää erikokoisilta, vaikka niille olisi asetettu sama pikselikoko. Valitsemme siis ensin fontin ja vasta sen jälkeen siitä käytettävät koot eri tasoille.

Valitsemamme fontti on Times New Roman, otsikossa koko 32 px, väliotsikoissa 24 px ja leipätekstissä 18 px.

Esimerkkejä eri fonttiperheistä. Kaikissa fonteissa on valittuna sama koko, vaikka visuaalisesti fontit vaikuttavat erikokoisilta.

Kun ulkoasun pohja – värit ja fontit – on päätetty, voimme siirtyä suunnittelemaan komponenttejamme. Aloitamme navigaatiosta, sillä se pysyy samankaltaisena jokaisessa näkymässä. Mobiilisovelluksissa voidaan käyttää erilaisia navigaatioratkaisuja; se voi sisältää perinteisiä tekstilinkkejä tai kuvamuotoisia ikoneita, se voi olla piilotettuna erillisen napin taakse tai koko ajan esillä. Rautalankamallissamme määrittelimme jo navigaation näkymään jokaisen näkymän alalaitaan, joten nyt mietimme vain, miten siinä esitetään eri vaihtoehdot.

Voimme testata tekstimuotoisia linkkejä, vaikka niitä harvemmin mobiilisovelluksissa käytetään. Vaikutelma on kuitenkin tönkkö, joten päädymme ikoneihin. Ikonit voimme valita vapaasti niin, että ne kuvaavat omasta mielestämme mahdollisimman tarkasti eri näkymiämme. Löydettyämme mieleisemme ikonit, sijoittelemme ne visuaalisesti miellyttävästi navigaatiolle varattuun tilaan ja määrittelemme niille värit. Koska ikonit ovat mobiilissa tunnettuja linkkeinä, ei niiden väriä ole välttämätöntä erottaa fonttiemme väristä. Käyttäjäystävällisyyttä lisää, mikäli on helposti havaittavissa, millä näkymällä käyttäjä kulloinkin on. Muokkaamme siis aktiivisen ikonin värin erilaiseksi muihin ikoneihin verrattuna.

Valmiita ikoneita voit etsiä hakusanoilla ‘free icon’. Voit myös itse piirtää omat ikonisi ja toteuttaa ne teknisesti kuvilla, mikäli et löydä valmiista vaihtoehdoista sopivia.

Seuraavaksi komponenttilistaltamme muokkaamme keskustelulistauksia, sillä ne kuuluvat nyt työstämäämme näkymään. Ranskalaiset viivat tai bullet pointit eivät ole visuaalisesti sovelluksessa kovin miellyttäviä, joten mietimme, kuinka mieluummin esittäisimme keskustelut, mitä tietoja haluamme niistä tässä kohdin näyttää ja kuinka erottelemme listan kohdat toisistaan.

On selvä, että keskustelunavauksista on hyvä näyttää otsikko, mutta sen lisäksi käyttäjää voisi kiinnostaa keskustelun aloituspäivä ja kuka keskustelun on avannut. Lisätään nämä siis näkyviin. Listakohteiden erottelussa voisimme käyttää ääriviivoja tai taustavärin muutoksia, mutta nämä näyttävät helposti tönköiltä. Päädymme siis tekemään erottelun visuaalisen viivan kautta. Muokataan vielä jäljelle jäänyt painike sellaiseksi, jona haluamme sen toteuttaa ja olemme valmiita tämän näkymän kanssa.

Tarkkapiirtoiset prototyypit on hyvä toteuttaa kaikista erilaisista näkymistä, jotta voit harkita ja kokeilla erilaisia ratkaisuja kaikkiin sovellukseesi tuleviin rakenteisiin. Jos olet sijoittanut rautalankamalliisi kuvia tai vaikkapa logon sovelluksellesi, toteuta ne myös prototyyppiisi. Näin varmistat, että lopputulos miellyttää sinua kokonaisuutena.

Aloitusnäkymä | Profiilinäkymä | Keskustelualueen näkymä

Luvun 3 keskeiset termit:

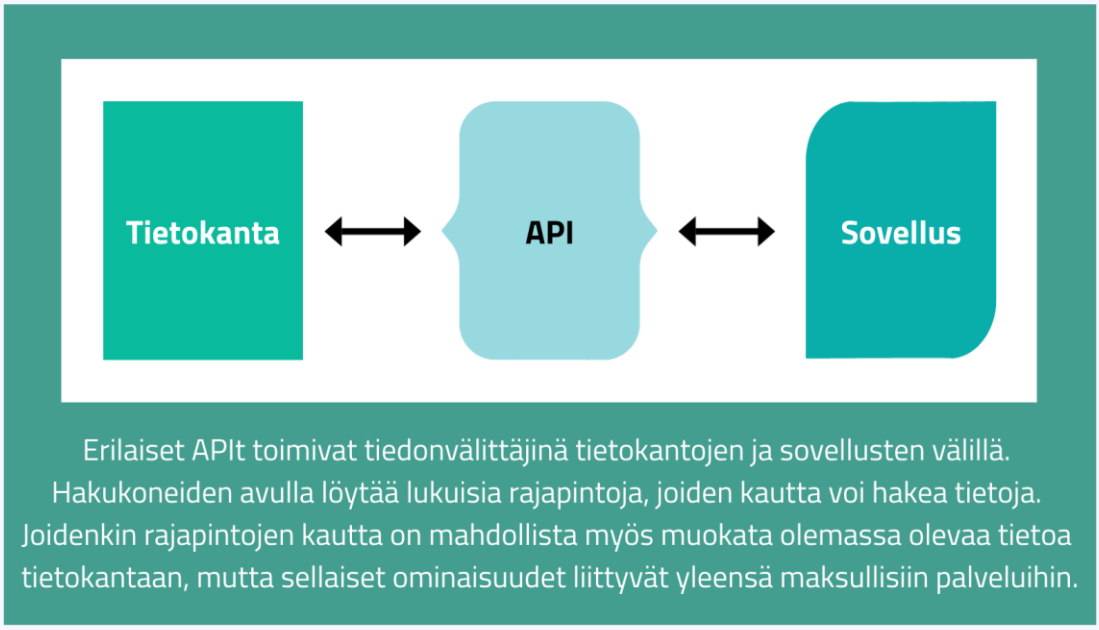
- Käyttöliittymä: sovelluksen osa, joka näkyy käyttäjille ja jonka kautta sovellusta voidaan käyttää
- Sovelluskartta: hierarkkinen malli, jossa luetellaan sovelluksen näkymät ja niiden päätoiminnallisuudet
- Näkymä: yksittäinen "sivu", jonka käyttäjä kerrallaan näkee, verrattavissa yksittäiseen websivuun.
- Rautalankamalli: sovelluksen eri näkymistä tuotetut mallit, joissa erilaisten laatikoiden ja muiden elementtien avulla visualisoidaan sovelluksen tuleva rakenne
- Komponentti: rakenneosa
- Tarkkapiirtoisempi prototyyppi: tarkka kuvaus tulevan sovelluksen ulkonäöstä, sisältää tiedot käytettävistä väreistä, kuvien ja fonttien ko'oista, komponenttien sijainnista ja ulkonäöstä
- Saavutettavuus: erilaisten rajoitteiden kanssa toimivat henkilöt voivat käyttää samoja palveluita kuin ilman rajoitteita elävät
- Heksadesimaali: värikoodi, ensimmäiset kaksi merkkiä kertovat punaisen värin määrän, seuraavat kaksi vihreän ja viimeiset kaksi sininen
- Fontti: tietynlainen kirjasintyyppi
- Fonttiperhe: kokoelma samankaltaisia fontteja
- Ikoni: kuva, joka edustaa jotain toiminnallisuutta tai näkymää sovelluksessa

Suunnitelman jälkeen

Nyt kun suunnitelma on tehty, on aika siirtyä pohtimaan sovelluksen sisältöä ja ohjelmoinnin aloittamista. Tässä kohtaa kannattaa vielä jaksaa keskittyä harkitsemaan sovellukseensa liittyviä ratkaisuja, sillä ne voivat säästää koodarilta paljon aikaa ja vaivaa.

Sisältöä pohtiessasi kannattaa hakea tietoa erilaisista API:sta (application programming interface, eli ohjelmointirajapinta), sillä on hyvin mahdollista, että jokin taho tarjoaa valmiiksi tuotettua sisältöä sovelluksesi aiheesta. Kuten tämän oppaan ensimmäisen luvun sivulauseessa mainittiin, eri ohjelmistotasojen välillä toimii rajapintoja, jotka välittävät tietoa eteenpäin. Myös APIt ovat tällaisia rajapintoja ja niiden avulla voit saada käyttöösi muiden tuottamien ja ylläpitämien tietokantojen sisältöä.

Jos hakisimme tietoa elokuvasovellukseemme, jo suomenkielisillä hakusanoilla 'elokuva api' löytyy paljon vaihtoehtoja, joiden kautta voi hakea tietoa elokuvista tai näyttelijöistä sekä elokuviin liittyviä kuvia. Englanninkielellä hakutuloksia tulisi vielä enemmän. Pohdi vaikuttaako muualta saadun sisällön kieli omaan sovellukseesi tai sen sisältöön kuinka suuresti ja tee valintasi käytettävästä hakusanoista/APIsta sen perusteella. Huomioi valinnoissasi myös, että osa API.sta saattaa olla maksullisia, vaikka ilmaisiakin vaihtoehtoja löytyy reilusti.



Erilaiset APIt toimivat tiedonvälittäjinä tietokantojen ja sovellusten välillä. Hakukoneiden avulla löytää lukuisia rajapintoja, joiden kautta voi hakea tietoja. Joidenkin rajapintojen kautta on mahdollista myös muokata olemassa olevaa tietoa tietokantaan, mutta sellaiset ominaisuudet liittyvät yleensä maksullisiin palveluihin.

Suosittelen myös pohtimaan, haluatko toteuttaa sovellukseen koko teknisen puolen itse vai helpotatko urakkaasi käyttämällä joissain kohdin valmiita palveluita. Esimerkiksi mikäli tarvitset tietokantaa sovellukseesi, voit toki luoda sen alusta asti itse esimerkiksi SQL:n avulla omalle palvelimellesi. Markkinoilta löytyy kuitenkin myös valmiita tietokantopalveluita, joiden käyttöönotto on nopeampaa ja jotka soveltuvat hyvin harrastusprojekteihin. Tämän kaltaisiin palveluihin voit määritellä haluamasi tietorakenteet ja voit muokata niitä helposti sovelluksestasi APIen avulla. Näissä ilmaisissa tietokannoissa on rajoitteita liittyen esim. tiedonsiirtomääriin, joten kannattaa vertailla erilaisia palveluita ja valita itsellesi sopivin.

Teknologiaa valitessa voit pohtia, millä käyttöjärjestelmillä haluat sovelluksesi toimivan. Mikäli haluat tuottaa sovelluksen vain iOS tai Android -käyttöön, voit valita käyttöösi Applen tai Androidin kehitystyökalut ja hakea apua ohjelmointiin heidän tutoriaaleistaan. Jos puolestasi haluat sovelluksesi toimivan molemmilla käyttöjärjestelmillä, tarvitset kolmannen osapuolen kehitysympäristön. Tutustu vaikkapa Expoon, jos haluat koodata JavaScript-pohjaisilla kielillä tai Flutteriin, jos et halua käyttää JavaScriptiä.

Kun olet valinnut teknologian, pohdi työjärjestystäsi. Mistä koet loogisimmaksi aloittaa sovelluksen tekemisen? Luotko ensin tarvittavat näkymät ja niiden välille navigoinnin, jonka jälkeen siirryt tekemään eri sisältöä näkymiin? Vai aloitatko luomalla ohjelmaasi tarvitsemasi luokat, jolla varmistat hyvän tiedonkäsittelyn ennen kuin keskityt varsinaisiin toiminnallisuuksiin? Mikä järjestys tuntuu sinusta järkevältä ja mikä auttaa ylläpitämään motivaatiotasi työn edetessä? Haluatko taklata isoimmat haasteesi ensin vai lämmitteletkö projektin alussa helpommilla työvaiheilla? Tähän valintaan en anna vinkkejä suuntaan tai toiseen, sillä eri tiet johtavat eri ihmisillä hyvään lopputulokseen ja sinä tunnet itsesi paremmin.

Jos ohjelmistosuunnittelu alkoi tämän oppaan myötä kiinnostamaan enemmän, suosittelen tutustumaan seuraavalta sivulta löytyvään lähdeluetteloon. Sieltä löytyviä tekstejä on hyödynnetty tämänkin oppaan tietoperustan pohjana ja niiden avulla voit perehtyä kaikkiin aiheisiin tarkemmin.

Lähteet

Oppaan tietopohjan rakentamisen tukena on käytetty alla listattuja lähteitä. Lähteet on järjestetty tekijän sukunimen mukaan aakkosjärjestykseen. Monet lähteistä on tarkoitettu laajempien projektien suunnitteluun ja niihin kannattaa ehdottomasti tutustua tarkemmin, mikäli haluaa perehtyä paremmin ohjelmistosuunnitteluun.

Ballard, B. 2007. Designing the Mobile User Experience. John Wiley & Sons, Ltd. West Sussex.

Bennett, S. 2015. UML Fundamentals. Class Diagrams. Katsottavissa: https://learning.oreilly.com/course/uml-fundamentals/9781771373630/.

Filipova, O., & Vilão, R. 2018. Software Development From A to Z. Apress. New York.

Luukkainen, M. & Laine, H. 2010. Luentomoniste kurssille Ohjelmistojen mallintaminen. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. Helsinki. Luettavissa: https://www.cs.helsinki.fi/u/mluukkai/ohmas10/ohma.pdf.

Nielsen, J. & Budiu, R. 2013. Mobile Usability. New Riders. Berkeley.

Sinkkonen, I., Nuutila, E. & Törmä, S. 2009. Helppokäyttöisen verkkopalvelun suunnittelu. Tietosanoma. Helsinki.

Shelton, B. 11.8.2023. UML class diagrams. Video. Katsottavissa: https://www.youtube.com/watch?v=6XrL5jXmTwM.

Wells, M. 2023. User Experience Design – An Introduction to creating interactive digital spaces. Laurence King Publishing. Lontoo.

Liite 2. Testilukijoille lähetetty saatekirje

Mobiilisovelluksen suunnittelun opas testiluettavaksi

Kiitos, kun olet suostunut testilukemaan kirjoittamani oppaan ensimmäisen mobiilisovelluksen suunnittelusta! Opas on tuotettu opinnäytetyönäni Haaga-Helia ammattikorkeakoulussa.

Toivon sinun lukevan oppaan kriittisellä silmällä pohtien sitä, vastaako sen sisältö aihettaan. Oppaan on tarkoitus toimia selkokielisenä ohjeena siihen, kuinka ensimmäinen mobiilisovellus voidaan suunnitella. Tavoitteena on saada lukija ymmärtämään suunnittelun tärkeys ennen ohjelmointiin siirtymistä ja havainnollistaa suunnittelun eri vaiheita.

Koska opas on suunnattu aloitteleville ohjelmoijille, ei siinä käydä läpi ohjelmistosuunnittelun kaikkia mahdollisia vaiheita. Ammattimaisessa ohjelmistotuotannossa suunnitteluprosessit ovat paljon kattavampia ja sisältävät enemmän työvaiheita kuin harrastusprojekteihin tehtävät suunnitelmat. On siis hyväksyttävää, että opas jättää jotain vaikeimpia kysymyksiä (esimerkiksi aikataulutukseen liittyen) vastaamatta, mikäli lopputulos kuitenkin herättää mielenkiintoa suunnittelua kohtaan.

Laitan alle listan kysymyksistä, joihin voit pohtia vastauksia opasta lukiessasi. Mikään kysymyksistä ei ole pakollinen, vaan saat suorittaa arvioinnin omalla tavallasi. Voit kirjoittaa arviointisi yhtenä tekstikappaleena tai voit vastata kysymyksiin yksitellen, esimerkiksi ranskalaisin viivoin. Toteutustapa on siis täysin sinun valinnassasi.

Kysymyksiä arvioinnin tueksi:

- Onko opas mielestäsi helppolukuinen?
- Oliko käsiteltävät termit avattu tarpeeksi selvästi vai jäikö mahdollinen ammattisanasto ymmärtämättä?
- Käsiteltiinkö aiheita tarpeeksi laajasti?
- Koetko ymmärtäväsi käsitellyistä aiheista, esim. vaatimusmäärittelystä tarpeeksi, jotta voisit omassa projektissasi toteuttaa samanlaisen työvaiheen?
- Koitko käytännön esimerkit hyödyllisiksi?
- Jäikö jostain aiheesta kysymyksiä vastaamatta tai olisitko kaivannut syvällisempää tietoa käsiteltävistä aiheista?

Saat antaa palautetta myös näiden kysymysten ohi, ns. vapaa sana -tyylillä. Kaikki palaute on tervetullutta, erityisesti rakentava kritiikki. Mikäli huomaat parannettavaa jollain osa-alueella, kuulisin mielelläni myös miksi koit kohdassa kehittämisentarvetta.

Kiitos jo etukäteen vastauksista,

Mari Rautanen

Liite 3. Testilukijoilta saadut vastaukset

Henkilö A – opiskellut korkeakoulussa tietojenkäsittelyä:

> ”Kokonaisuudessa vaikuttaa hyvin helppolukuiselta vaikea keksii mitään parannettavaa. Ainoo asia mitä mietin s. 7 ei varmaan ees oo oleellista, mutta vaatimusmäärittelylistassa vois olla kans sovelluksen tietosuojakäytännöistä maininta vaikka siinä kans lukeekin alla että listaus ei oo täydellinen. Visuaalinen puoli toimi hyvin ja koin että se tuki asiasisältöistä tekstiä. Kuvat auttoivat hahmottamaan myös käsiteltävissä olevaa asiaa todella hyvin.”

Henkilö B – opiskellut korkeakoulussa muuta kuin tietojenkäsittelyn alaa:

> ”-Opasta lukiessa ei tule olo, että ns. tippuu kärryiltä. Uudet ja jopa hiukan pelottavan kuuloiset termit selitetään hyvin auki ja käytännön esimerkit tukevat niiden ymmärtämistä sekä sisäistämistä. Väistämättä tietenkin uusien asioiden oppiminen vaatii, että termistöä täytyy kerrata edetessään projektissa eteenpäin. Kuvat oppaassa ovat selkeitä ja ne helpottavat uusien termien sisäistämistä sekä niiden kertaamista.
>
> -Termistä toiseen liikutaan jouhevasti ja niiden yhteys toisiinsa näkyy selkeästi. -Lukijan on helppo ymmärtää mitä sovellus oikeasti pitää sisällään.(mm. Mikä se tietokanta on ja miksi sitä tarvitaan)
>
> -Etenemistahti on hiukan nopea, mutta tämä ei ole välttämättä huono asia. Lukijan mielenkiinto pysyy ja turhat asiat jäävät pois ja keskitytään olennaiseen. Tietenkin tämä voi myös aiheuttaa hetkellistä paniikkia, mutta lukiessa eteenpäin ja kertaamalla termejä asia selkiytyy esimerkkien kautta. Huomasin, että tätä ”paniikkia” itsellä ainakin ilmeni lukiessa ohjelmoinnin suunnittelusta. Uusia termejä käytetään kerralla paljon, mutta ne selkenevät kun malttaa lukea rauhassa. Kannattaa ehkä silti pohtia haluaako jollain tapaa väntää tätä kohtaa vielä enemmän rautalangasta.
>
> -oppaan kirjoittajan tapa muistuttaa, että tässä ollaan vasta harjoittelemassa eikä tekemässä vuoden hienointa sovellusta lisää lukijan itsevarmuutta. Hän voi oikeasti oppia tästä jotain eikä se ole liian vaikeaa.”

Henkilö C – ei ole koskaan opiskellut korkeakoulussa:

> ” Elikkä tämmösen ihmisen silmiin ketä ei tiedä koodauksesta eikä ole ollut siihen koskaan mielenkiintoa niin alku oli mulle vähän hankala ymmärtää mutta sitten ku päästiin siihen sivun suunnitteluun ja fontteihin niin mun oli jostain syystä helpompi ymmärtää ja tosi hyviä vinkejä olit lisännyt sinne mm sen puna viher sokeuden koska sitä ei itse välttämättä tulis ees mieleen. Jotkut noi ammattitermit jäi mulle auki että en tiiä oisko ne voinut jossain kerrata esim back end koodi=tämä, frond end koodi=tämä, ikoni = kuvia esim jotenki näin. Kyllä käsittelit laajasti ja hyvä oli kanssa ne lähde merkinnät että jos jotain haluaa vielä syventyä niin tietää mistä etsii ja oli kanssa hyvä lisä että mitä kannattaa käyttää jos haluaa lähtee työstää omaa sovellusta. Niinku tossa sanoin alussa koska koodaus ei ole koskaan ollut mun intohimoni ni alku oli mulle vaikeampaa ymmärtää mutta ulkoasun suunnittelu vaihe oli mun helpompi ymmärtää. Koin todella hyödylliseksi esimerkit musta on hyvä kerrata ja konkreettisesti nähdä. Sitten ainakin tietää että on ymmärtänyt lukemaansa.”