LAB University of Applied Sciences

Asiakastukikysymyssivuston ohjelmointi Avenla Oy:lle

Stack Overflow -tyyppinen web-sovellus

LAB-ammattikorkeakoulu
Tradenomi (AMK), Tietojenkäsittely
2023
Aleksander Väliaho

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Väliaho, Aleksander	Opinnäytetyö, AMK	2023 Kevät
	Sivumäärä	
	91	

Työn nimi

Asiakastukikysymyssivuston ohjelmointi Avenla Oy:lle

Tutkinto ja koulutusala

Tradenomi (AMK), tietojenkäsittely

Toimeksiantajan nimi, titteli ja organisaatio (jos opinnäytetyöllä on toimeksiantaja)

Johannes Gangsö, Web-kehittäjä, Avenla Oy

Tiivistelmä

Työelämän edustajan toimeksiannosta työssä ohjelmoitiin Stack Overflow -tyyppinen asiakastukikysymyssivusto. Sivustolle kuka tahansa Avenla Oy:n asiakas voi lähettää kysymyksiä Wordpress Multisite Shared Media -sovellukseen liittyen ja saada vastauksia niihin. Sivuston tekninen toteutus nojasi Angular-käyttöliittymä-frameworkin sekä Wordpress-sisällönhallintajärjestelmän REST-rajapinnan ja tietokannan yhdistelmään. Avenla Oy saa sähköpostitse toistuvia kysymyksiä heidän ohjelmistotuotteisiinsa liittyen ja niihin yksitellen vastaaminen on kuormittavaa, minkä takia tarvitaan paikka, jossa keskitetysti asiakkaat saisivat vastauksia kysymyksiinsä.

Osana työtä toteutettiin kvantitatiivinen tutkimus, jolla selvitettiin kuinka hyvin tuloksena syntyvä sovellus vastaa vaatimuksiin ja kuinka hyvän käyttökokemuksen se antaa. Tutkimuksen teoriapohjaksi valittiin sopiva lähdeteos. Tulosten mukaan vastaanotto artefaktille oli varovaisen positiivinen. Sovellus kaipaa viimeistelyä, mutta hyvä pohja on jo olemassa. Lopputuotteelle nähtiin potentiaalia toimeksiantajan työkaluvalikoimassa.

Asiasanat

Angular, Wordpress, asiakastukikysymys, käyttäjäkokemus

Abstract

Author(s)	Type of Publication	Published
Väliaho, Aleksander	Thesis, UAS	2023 Spring
	Number of Pages	
	91	

Title of Publication

Programming customer support question website for Avenla Ltd.

Degree and field of study

Bachelor of Business Administration, Information Technology

Name, title and organisation of the client (if the thesis work is commissioned by another party)

Johannes Gangsö, Web Developer, Avenla Ltd.

Abstract

As an assignment from the representative of the working life the thesis contains a Stack Overflow like customer support question web site. Any customer of Avenla Ltd.'s Wordpress Multisite Shared Media software can send questions to the site and receive answers to them. The technical implementation of the thesis' software relies on Angular front-end framework including Wordpress CMS REST API and its database. Avenla Ltd. receives numerous questions via email regarding its software products and answering all of them individually in detail is consuming, so there exists a need to have a centralized place for customers to ask questions and get answers. As a part of the work quantitative research was executed to find out how well the resulting software meets the requirements and how good of an experience the user has using the software. A suitable source material was selected as a theoretical foundation for the research. According to the results the reception for the artefact was cautiously positive. The application needs polishing, but the foundation is there. There is potential for the final product to be in the client's toolkit.

Keywords

Angular, Wordpress, customer support question, user experience

Sisällys

1 Johdanto

1.1 Opinnäytetyön tausta, tavoite ja rajaus

Opinnäytetyön aiheena on toteuttaa Avenla Oy:lle ohjelmistoratkaisu, joka kantaa nimeä Wordpress Multisite Shared Media -tukikysymyssivusto. Avenla Oy:llä on ohjelmistotuote, jota he haluavat myydä. Yritys saa usein sähköpostiinsa kysymyksiä asiakkailta ohjelmiston käyttöönottoon ja tukeen liittyen. Näihin vastaaminen on osoittautunut liian työlääksi. Yrityksen toiveena onkin luoda sivusto, johon on koottu yhteen paikkaan asiakkaiden lähettämät kysymykset ja Avenla Oy:n kirjoittamat vastaukset. Aiheen valintaan vaikutti ratkaisevasti pari tekijää. Tärkein tekijä on se, opinnäytetyötä tarjoaa mahdollisuuden tuottaa hyötyä toimeksiantajayritykselle, sillä käyttöön saadaan ohjelmisto, jolla on päivittäin oikeaa käyttöä yrityksen liiketoimintaprosesseissa. Täten asiakkaita voidaan palvella nopeammin ja tehokkaammin. Varmasti myös aikaa säästyy, kun ohjelmisto mahdollistaa keskitetyn asiakaspalvelun, eikä tarvitse vastata kaikkiin asiakastukikysymyksiin yksitellen sähköpostitse. Aihe on myös ajankohtainen: varmasti monet yritykset ohjelmistotuotannon alalla tälläkin hetkellä miettivät sitä, miten voivat virtaviivaistaa liiketoimintaansa digitaalisilla työkaluilla.

Opinnäyteraportin lisäksi opinnäytetyön tuloksena toteutetaan Avenla Oy:lle käytettäväksi toimiva sivusto, jonka avulla asiakkaat voivat lähettää tukikysymyksiä ja saada niihin vastauksia kootusti yhdestä paikasta. Lopputuotteena syntyvä sivusto toimii web-selaimessa, eikä se ole esimerkiksi mobiilisovellus.

Raportin loppuun liitetään kvantitatiivinen tutkimus toteutettavan sovelluksen käyttökokemuksesta. Toteutettavan artefaktin tueksi tehtävä kvantitatiivinen tutkimus rajataan siten, että siinä kysymyksiin vastaavat vain ne Avenla Oy:n työntekijät, jotka tulevat työskentelemään artefaktin parissa myöhemminkin. Tämä tarkoittaa siis sellaista rajausta, että vain toimeksiantajan näkemys kerätään. Anonyymi tutkimus mittaa sovelluksen käyttökokemuksen laatua. Vastaajat arvioivat erilaisten väittämämuotoisten Likert-kysymysten kohdalla näkemyksiään artefaktista asteikolla täysin samaa mieltä—täysin eri mieltä.

1.2 Toimeksiantaja

Avenla Oy on vuonna 2005 perustettu Lahdessa kotipaikkaansa pitävä ohjelmistokonsultointiyritys, jolla on sekä PHP- että .NET-ohjelmistokehitystiimit. Avenla Oy:n tuottamat ohjelmistot ovat pääosin suunnattu webin puolelle. Yrityksen liikevaihto on 1,8 miljoonaa euroa ja 12/2021 päättyneellä tilikaudella yhtiöllä oli 18 työntekijää. (Kauppalehti 2023.)

2 Tutkimuskysymykset

Likert-kysymykset on hahmoteltu väittämämuotoon, jotta automaattisesti assosioitaisiin, että lukuarvoissa isompi on parempi.

- Mitä erityisiä vaatimuksia Avenla Oy:llä on tukikysymyssivustolle?
    - Raportissa mainitut toiminnallisuudet vastaavat Avenla Oy:n mielestä lopputuotteena syntyvän tukikysymyssivuston vaatimuksia. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
    - Miten tukikysymyssivusto auttaa Avenla Oy:tä ja alaa yleensäkin?
- Millainen on sovelluksen käytettävyys?
    - Yksinkertaisten asioiden suorittaminen sovelluksessa on helppoa. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
- Lopputuotteen käytettävyys ja onnistuminen riippuu usein siitä, kuinka hyvin rautalankamallit ovat tehty. Kuinka huolellisesti sivusto on suunniteltu?
    - Raportin ohjelmistotuotanto-osuuden rautalankamallit on tehty hyvin. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
- Kuinka loogisia ovat käyttökokemuksen rakennetason asiat?
    - Painonapit ovat oikeassa paikassa suhteessa muihin ohjaimiin, joita käyttäjä käyttäisi siinä tilanteessa. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
    - Sivustolla navigoiminen tuntuu luonnolliselta, kun halutaan suorittaa asioita. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
    - Sivuston terminologiaa on helppo ymmärtää. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
- Kuinka johdonmukaisia ovat käyttökokemuksen luurankotason asiat?
    - Tukikysymyssivuston käyttöliittymä on johdonmukainen verrattuna tuttuihin, muihin samankaltaisiin sivustoihin. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.

  - Tukikysymyssivuston käyttöliittymä on johdonmukainen itsessään läpi koko sivuston. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
  - On helppoa ymmärtää, missä ollaan, mihin voidaan mennä ja mitkä sivuston valinnat auttavat pääsemään tavoitteisiin. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.

- Kuinka loogisia ovat käyttökokemuksen pintatason asiat?
  - Visuaalinen suunnittelu tukee sivustolle määritettyjä tavoitteita. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
  - Visuaalinen suunnittelu selventää käyttäjille sivustolla tarjottavia vaihtoehtoja vahvistaen rakennetta. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.
  - Sivuston visuaalinen ilme viestii hyvin organisaation brändi-identiteettiä. Tämän väittämän ratkaisumenetelmänä käytetään kvantitatiivisen tutkimuksen Likert-asteikkoa.

3 Sovelluksen arkkitehtuuri

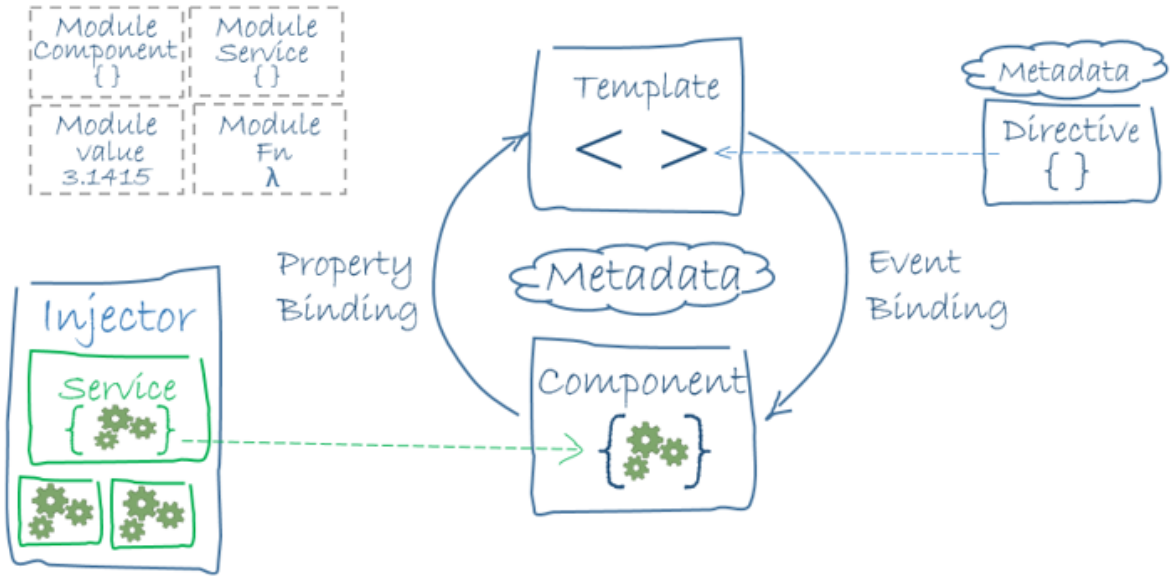
Opinnäytetyön lopputuloksena rakentuva internet-selainpohjainen sovellus rakentuu kahdesta itsenäisestä ohjelmistokokonaisuudesta. Käyttöliittymä ohjelmoidaan Angular 11 -web-frameworkilla ja Wordpress-sisällönhallintajärjestelmän REST-ohjelmointirajapinta toimii linkkinä käyttöliittymän ja tietokannan välillä.

Angularista puhuttaessa termi framework tarkoittaa ohjelmistoalustaa, jolla sovelluksia kehitetään. Se tarjoaa perustukset, jolla ohjelmistokehittäjä voi rakentaa ohjelmia. Framework saattaa myös sisältää ohjelmistokirjastoja, kääntäjän ja muita ohjelmistokehitysprosessin ohjelmia. (Sharpened Productions 2013.) Frameworkin tarkoitus on antaa ohjelmistokehittäjälle mahdollisuus keskittyä rakentamaan ominaisuuksia hänen web-projektilleen sen sijaan, että pyörä täytyisi joka kerta keksiä uudelleen. Frameworkit ovat erikseen kehitetty sitä varten, että ohjelmistokehittäjä voi parantaa ohjelmiston suorituskykyä ja tehokkuutta. (Cuelogic 2013.) Angular on ohjelmistokehitysalusta käyttöliittymäpuolen yksisivuisten sovellusten rakentamiseen HTML:llä ja TypeScriptillä. Angularia kirjoitetaan TypeScriptillä, joka on kuin eräänlainen murre JavaScript-ohjelmointikielestä. TypeScript on lyhyesti ilmaistuna JavaScriptiä, jossa on mukana staattinen tyypitys. Tyypittämisessä on se etu, että voidaan hyvissä ajoin havaita ohjelmiston virheet ja saadaan ehdotuksia korjauksista ennen kuin lähdekoodia ajetaan selaimessa. TypeScript siis validoi lähdekoodia ennenaikaisesti. Ohjelmistokehittäjä tuo sovellukseen TypeScript-kirjastoja, joilla voi toteuttaa erilaisia ominaisuuksia.

Angular sisältää komponenttipohjaisen frameworkin, jolla voi luoda skaalautuvia web-sovelluksia. Lisäksi Angularissa on kokoelma integroituja kirjastoja, jotka kattavat monia ominaisuuksia, kuten reitittämisen, lomakehallinnan, käyttöliittymä-palvelinkommunikoinnin ja paljon muuta. Angularissa on myös joukko työkaluja, jotka auttavat ohjelmoijaa kehittämään, kääntämään, testaamaan ja päivittämään koodia. Viimeisin versio (tunnetaan myös nimellä Angular v2) on Angular 11, joka on täysin puhtaalta pöydältä suunniteltu, uusi versio jo nykyisin vanhentuneesta AngularJS 1.x -frameworkista ja sen on kehittänyt sama tiimi ohjelmoijia Googlella. (Angular 2021.)

Kuten kuviosta 1 näkee, on Angularissa komponentti vahvasti sidoksissa sen templateen, jossa voidaan halutessa käyttää direktiivejä antamaan templaten HTML-syntaksille uusia ulottuvuuksia. Template ja komponentti yhdessä määrittävät view:n, eli selaimessa näkyviin tulevan sivun tai sen osan. Hyvien käytäntöjen mukaisesti etäpalvelimen kanssa kommunikointi ei kuulu komponentin tehtäviin, vaan se on ulkoistettu serviceille, joista http-kutsut sisältävät metodit injektoidaan komponenttien käytettäväksi. Dependency injector tuo komponentille myös reititin-servicen, joka sallii määrittämään selaimessa tapahtuvan sivujen

välisen navigoinnin. (Angular 2022h.) Ymmärtääkseen Angularia tarvitaan jonkin verran kokemusta yleisistä web-ohjelmistokehitystekniikoista, kuten HTML5:stä, CSS3:sta ja JavaScriptistä.



Kuvio 1. Angular-sovelluksen arkkitehtuurin yleisnäkymä (Angular 2021)

Wordpress on yksinkertaisin ja suosituin tapa luoda verkkosivusto tai blogi. Itse asiassa Wordpressia käyttää yli 40 % kaikista internetin sivustoista. Hieman teknisemmin ajateltuna Wordpress on GPLv2-lisensoitu avoimen lähdekoodin sisällönhallintajärjestelmä, mikä tarkoittaa sitä, että kuka tahansa voi käyttää tai muokata Wordpressiä ilmaiseksi. Sisällönhallintajärjestelmä on pohjimmiltaan työkalu, joka tekee sisällön hallitsemisesta helppoa ilman että tarvitsee osata ohjelmoida. Lopputulos on se, että Wordpress tekee verkkosivustojen rakentamisesta saavutettavaa kenelle tahansa, myös niille, jotka eivät ole kehittäjiä. Monta vuotta sitten Wordpress oli pääasiassa bloginluontityökalu, eikä sillä voinut tehdä perinteisiä verkkosivuja. Se ei kuitenkaan ole pitänyt paikkaansa enää pitkään aikaan. Nykyisin, kiitos lähdekoodin muutosten, sekä Wordpressin massiivisen laajennusten ja teemojen ekosysteemin, Wordpressilla voi luoda aivan minkä tahansa tyyppisen verkkosivuston. Wordpressilla voi nykyisin luoda muun muassa verkkokauppoja, bisnessivustoja, blogeja, portfolioita, ansioluettelosivuja, foorumeita ja sosiaalisia verkostoja sekä oikeastaan mitä tahansa muuta kuviteltavissa olevaa. Wordpress jakautuu kahteen erilliseen toimintalogiikkaan, wordpress.orgiin ja wordpress.comiin. Wordpress.org-sivusto tarjoaa ilmaisen ladattavan version sen lähdekoodista, jonka voi lähettää omalle palvelimelleen isännöitäväksi, kun taas wordpress.com on voittoa tavoitteleva versio Wordpressista, jolla voi helposti ja opastetusti luoda oman verkkosivuston haluamalleen verkko-osoitteelle. (Kinsta 2021.)

Opinnäytetyössä Wordpressin kautta hyödynnetään REST-ohjelmointirajapintaa (API, application programming interface). REST API on arkkitehtuurillinen tyyli ohjelmointirajapinnalle, joka käyttää HTTP-pyyntöjä datan siirtämiseen. Tätä dataa voidaan käyttää GET-, PUT-, POST- ja DELETE-pyynnöillä, jotka viittaavat tietojen lukemiseen, päivittämiseen, luomiseen ja poistamiseen, kun ne kohdistetaan niille kuuluviin resurssiosoitteisiin. (Gillis 2020.) Ohjelmointirajapinta tarkoittaa ohjelmistovälittäjää, joka sallii kahden itsenäisen sovelluksen kommunikoida keskenään. Esimerkiksi joka kerta kun käytetään Facebookia, lähetetään viesti pikaviestimellä tai tarkistetaan sää puhelimella, käytetään ohjelmointirajapintaa. (Mulesoft 2021.) Puhuttaessa web-rajapinnoista, kuten REST:istä, tarkoitetaan tässä tapauksessa Angular-käyttöliittymän ja web-palvelimen MySQL-tietokannan välistä yhteysväylää, joka välittää informaatiota edestakaisin JSON-muodossa (JavaScript Object Notation). Freemanin (2019) mukaan JSON on tekstipohjainen esitysmuoto rakenteellisesta datasta, joka pohjautuu avain-arvo-pareihin. Vaikka JSON pohjautuu JavaScriptiin, se on tuettuna joko luontaisesti tai kirjastojen kautta useimmissa suurissa ohjelmointikielissä, kuten esimerkiksi PHP:ssä, jota myös Wordpress käyttää.

Työ tullaan rajaamaan siten, että esimerkiksi käyttäjäkohtaista sisäänkirjautumista tai ilmoitusjärjestelmää uusista kysymyksistä ei tulla toteuttamaan.

4 Sovelluksen tuottaminen

4.1 AppComponent

Kehitettäessä Angular-sovelluksia tarvitaan luonnollisesti IDE lähdekoodin kirjoittamista varten. Tähän tarkoitukseen on olemassa useita eri vaihtoehtoja, mutta suositeltavinta lienee käyttää Microsoftin Visual Studio Code -editoria, koska siitä löytyy luontainen tuki Angularissa käytettävälle TypeScript-kielelle, joka on sekin Microsoftin kehittämä ohjelmointitekniikka. Visual Studio Code on kattavat ominaisuudet sisältävä IDE (integrated development environment) ja sen kanssa alkuun pääseminen on nopeaa ja suoraviivaista. Kuten useissa muissakin IDE:issä, Visual Studio Code sisältää muun muassa tekstieditorin, älykkään, ehdottavan tekstinsyötön, virheenkorjaustyökalut ja Git-versionhallinnan. Vaikka Git-versionhallinta on olennainen osa ohjelmistokehittäjän päivittäistä rutiinia, ei sitä tulla tässä raportissa sen syvällisemmin käsittelemään.

Mikäli Angularin perusteiden opettelulle on tarve, löytyy Angularin dokumentaatiosivuilta Tour of Heroes -niminen opas. Siinä käydään yksityiskohtaisesti läpi Angular-sovellusten eri osaset ja toimivan sovelluksen kirjoittaminen.

AppComponentin rooli sovelluksessa

Sivu, joka nähdään selaimessa osoitteessa http://localhost:4200/ on sovelluksen runko. Runkoa kontrolloi Angularin komponentti nimeltään `AppComponent`. Komponentit Angularissa ovat perustavaa laatua olevia rakennuspalikoita. Ne saavat datan ilmestymään ruudulle, kuuntelevat käyttäjän näppäinsyötteitä ja suorittavat syötteiden perusteella haluttuja toimintoja.

Sovelluksen `AppComponent`-komponentilla on aivan erityinen rooli sovelluksessa. Yksinkertaisissa, yksisivuisissa ohjelmissa `AppComponentin` luokkakoodit sisältävään tiedostoon `app.component.ts` voitaisiin suoraan lähteä ohjelmoimaan sovelluksen etusivua, mutta tämän tukikysymyssivuston tapauksessa, jossa on tarkoituksenmukaista tehdä useampi websivu, ei ole suotavaa tehdä raportissa myöhemmin mainittavaa etusivua. Sen sijaan `AppComponentin` vastuulle jätetään koko sovelluksen laajuiset HTML-elementit, kuten otsakkeen hakupalkki. `AppComponentin` rooli sovelluksessa on siis olla niin sanottuna ”bootstrapperina”. Ohjelmistokehityksessä ”bootstrap” tarkoittaa tekniikkaa, jossa ohjelma ladataan alustavasti tietokoneen keskusmuistiin muutaman alkukomennon kautta, mikä mahdollistaa muun sovelluksen esiin tuomisen hieman myöhemmin. Tämän tekniikan käyttäminen antaa mahdollisuuden hallita sitä, miltä sovellus käynnistymisvaiheessa näyttää.

”Bootstrap”-tekniikkaa ei tule sekoittaa samannimiseen front-end-frameworkkiin, jonka sisällyttäminen projektiin näytetään seuraavassa vaiheessa.

Koko sovelluksen kattavien HTML-elementtien lisääminen AppComponentiin

Tukikysymyssivustolle on tarkoitus luoda koko sivuston kattava, johdonmukainen graafinen ulkoasu. Lisäksi halutaan, että sivusto toimii mukautuvasti päätelaitteen ruudun koosta riippumatta. Näitä ominaisuuksia varten on olemassa Bootstrap-framework. Maininnan arvoista on myös se, että Bootstrap-framework sisältää myös JavaScript-liitännäisiä tuomaan front-end-sovelluksille lisätoiminnallisuuksia. Tämän projektin kohdalla kuitenkin olennaisin saatu hyöty Bootstrapista on nykyaikainen, selkeä graafinen ulkoasu ja mukautuvaisuus.

Bootstrapin CSS-kirjaston tuominen projektiin on melko suoraviivainen operaatio. Mikäli komentokehotteella on vieläkin auki Angularin kehitysympäristöä ylläpitävä `ng serve` -komento, suljetaan se painamalla `Ctrl+C`-näppäinyhdistelmää kahdesti. Annetaan sitten komentokehotteelle projektipolun sisällä komento `npm install bootstrap`. Seuraavaksi Angularin kehitysympäristö voidaan palauttaa takaisin toimintaan antamalla komento `ng serve`. Visual Studio Codessa avataan `src/`-polun `styles.css`-tiedosto. `Styles.css`-tiedosto sisältää koko sovelluksen kattavat globaalit CSS-määrittelyt. Importoidaan äsken projektiin lisätty Bootstrap-kirjasto seuraavalla rivillä koodia:

```
@import "~bootstrap/dist/css/bootstrap.css"
```

Selainpäässä, mikäli äsken `ng serve` -komennolla palautettiin kehitysympäristö käyntiin, voidaan jo havaita muutos graafisessa ilmeessä. `AppComponentiin` lisätään seuraavaksi HTML:ää, joka saa otsakepalkin hakukentän näkymään kaikilla tulevilla sovelluksen sivuilla.

`App.component.html`-tiedoston sisältö näyttää nyt seuraavanlaiselta:

```
<nav class="navbar navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">
            <img src="../assets/avenla-
logo.svg" alt="Avenla Ltd. Logo" width="120" height="24"
                class="d-inline-block align-text-top">
            Wordpress Multisite Shared Media
        </a>
        <form class="d-flex">
            <input class="form-control me-2" type="search" place-
holder="Search" aria-label="Search">
            <button class="btn btn-outline-light" type="submit">Search</but-
ton>
        </form>
    </div>
```

```
</nav>
```

Avenla Oy:n omia verkkosivuja mukaillen yläpalkki on määritetty värimaailmaltaan tummaksi `navbar-dark`- ja `bg-dark`-CSS-määreillä, siihen on liitetty Avenla Oy:n logo vasempaan reunaan `img`-tagin avulla ja oikeaan reunaan tulee näkyviin hakukenttä `input`-tagissa. Kuten aiemmin mainittu, kaikki edellä mainittu näkyy joka ikisellä tulevaisuudessa luotavalla web-sivulla.

Seuraavaksi lähdetään ohjelmoimaan hakukentän toiminnallisuutta. Idea lyhykäisyydessään on se, että sovellus kaappaa hakukentässä olevan merkkijonon ja TypeScript-funktion `router`-kirjaston avulla käyttäjä siirretään toiselle sivulle.

Muutetaan `app.component.html`-tiedostossa `input`-tagia siten, että se näyttää tältä:

```
<input [(ngModel)]="haku" name="haku" class="form-control me-2"
type="search" placeholder="Search" aria-label="Search">
```

Jotta `form`-tagin sisällä olevan `input`-tekstikentän arvo saadaan kaapattua, on Angular 11:ssa vaatimus lisätä myös `name`-attribuutti. Annetaan tälle arvoksi `haku`. Opinnäytetyön artefaktin toteutuksessa käytetään suomenkielisiä muuttujien nimiä, jotta on helpompi erottaa mikä osuus lähdekoodista on omaa tuotantoa ja mikä osa on Angularin syntaksia. `[(ngModel)]` on kaksisuuntainen datan yhteen kytkentä -syntaksi. Hakasulkeiden sisällä olevia kaarisulkeita voidaan myös nimittää niin sanotuksi ”banaani laatikossa”-syntaksiksi. `ngModel` saa arvokseen `app.component.ts`-luokkatiedoston `haku`-muuttujan. Käyttämällä `ngModelia` komponentin luokkatiedoston muuttujan arvo lähetetään HTML-tekstikenttään ja HTML-tekstikentästä takaisin luokkatiedostoon. (Angular 2022n.) Lisätään `app.component.ts`-tiedostoon muuttuja `haku` ja annetaan sille tyypiksi `string` eli `haku: string;`

Mikäli nyt siirrytään selaimeen tarkastelemaan miltä sovellus näyttää, huomataan, että sovellus on kaatunut ja selain näyttää seuraavanlaisen virheilmoituksen: `Can't bind to 'ngModel' since it isn't a known property of 'input'`. Vaikka `ngModel` on ihan oikea Angularin direktiivi, se ei ole oletusarvoisesti saatavilla. Se kuuluu valinnaiselle `FormsModulelle` ja ohjelmoijan täytyy päättää haluta ottaa se käyttöön.

AppModule

Angularin tarvitsee saada tietää miten sovelluksen eri palaset sopivat yhteen sekä mitä muita tiedostoja ja kirjastoja sovellus vaatii. Tätä kutsutaan kuvaustiedoksi eli metadataksi. Osa kuvaustiedosta on `@Component`-dekoraattorissa, joka lisättiin komponentin luokkatiedostoon. Muu kriittinen kuvaustieto sijaitsee `@NgModule`-dekoraattorissa. Kaikista tärkein `@NgModule`-dekoraattori merkitsee `AppModule`-luokan. Kun projekti alun perin luotiin, Angularin CLI loi `AppModule`-luokan `src/app/app.module.ts`-tiedostoon. Tämä on se paikka,

jossa ohjelmoija päättää ottaa käyttöön `FormsModulen`. Avataan `AppModule` (`app.module.ts`) ja importoidaan `FormsModule`-symboli `@angular/forms`-kirjastosta seuraavanlaisesti:

```
import { FormsModule } from '@angular/forms';
```

Sitten lisätään `FormsModule` `@NgModulen` metadatan `imports`-taulukkoon, joka sisältää listan ulkoisista moduuleista, joita sovellus tarvitsee:

```
  imports: [
    BrowserModule,
    FormsModule
  ],
```

Kun selaimen välilehti päivittyy, sovellus toimii taas. Seuraavaksi sovellukseen tuodaan `Router`-moduuli, jonka avulla tekstikentästä kaapattu merkkijonoa voidaan käyttää siirtymiseen hakusivulle.

AppRoutingModule

Angularissa `RouterModule` eli reititin on moduuli, jonka vastuulla on yhdistää komponentit selainpäässä näkyviin URL-osoitteisiin, kuten esimerkiksi tulevaisuudessa tehtävä `HakuComponent` voisi näkyä selaimessa osoitteessa http://localhost:4200/search/. Vielä tässä vaiheessa, kun `AppComponent` on keskeneräinen, käytämme `Router`-moduulia `AppComponentissa` vain siihen, että sivuilla ylhäällä näkyvä hakupainike tekstikentän vieressä ohjaa käyttäjän edellä mainittuun URL-osoitteeseen. Myöhemmin tullaan kyllä ohjelmoimaan varsinainen sovelluksenlaajuinen reititys.

Angularissa on paras käytäntö ladata ja konfiguroida reititin erillisessä, korkean tason moduulissa, joka on omistettu reitittämiselle, ja joka importoidaan `AppModule`-juurimoduulissa. Tavanomaisen käytännön mukaan moduulin luokkatiedoston nimi on `AppRoutingModule` ja se kuuluu `app-routing.module.ts`-tiedostoon `src/app`-polussa. Annetaan Angularin CLI:ssä komento `ng generate module app-routing --flat --module=app`. `--flat`-lisämääre laittaa tiedoston `src/app`-polkuun oman polkunsa sijasta ja `--module=app`-lisämääre kertoo CLI:lle, että se rekisteröidään `AppModulen` `imports`-taulussa. (Angular 2022a.) Luodaan samalla tulevaisuudessa tarvittava `EtusivuComponent`, jonka vastuulla nimensä mukaisesti on ylläpitää etusivun templatea ja luokkakoodeja. Uuden komponentin luominen CLI:ssä hoituu yksinkertaisesti komennolla `ng generate component etusivu` (Angular 2022n). Luotu reititysmoduulitiedosto, kun siihen on tehty halutut muutokset, näyttää seuraavanlaiselta:

```
import { NgModule } from '@angular/core';
```

```
import { RouterModule, Routes } from '@angular/router';
import { EtusivuComponent } from './etusivu/etusivu.component';

const reitit: Routes = [
  { path: 'frontpage', component: EtusivuComponent },
  { path: '', redirectTo: '/frontpage', pathMatch: 'full' }
];

@NgModule({
  imports: [RouterModule.forRoot(reitit)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Ensiksi `app-routing.module.ts` importoi `RouterModulen` ja `Routesin`, jotta sovelluksella voi olla reititystoiminnallisuuksia. Seuraava `import EtusivuComponent` antaa `Routerille` paikan mihin mennä, sitten kun reitit on konfiguroitu.

Seuraava osa tiedostoa on se paikka, jossa reitit konfiguroidaan. Reitit kertovat reitittimelle mikä näkymä tulee näyttää, kun käyttäjä klikkaa linkkiä tai liittää URL-osoitteen selaimen osoitepalkkiin. Koska `app-routing.module.ts` jo valmiiksi importoi `EtusivuComponentin`, se voidaan lisätä `reitit`-tauluun:

```
const reitit: Routes = [
  { path: 'frontpage', component: EtusivuComponent },
  { path: '', redirectTo: '/frontpage', pathMatch: 'full' }
];
```

Tyypillisessä Angularin `Routessa` on kaksi ominaisuutta. `path` on merkkijono URL-osoitteeseen selaimen osoitepalkissa. `component` on se komponentti, jonka reitittimen pitäisi luoda, kun siihen navigoidaan. Tämä kertoo reitittimelle sen, että reitittimen pitäisi sovittaa yhteen se URL-osoite `path: 'etusivu'`:n kanssa ja näyttää `EtusivuComponent`, kun URL on jotakin tyyliin `localhost:4200/etusivu`. (Angular 2022n.)

AppComponentin viimeistely

`app.component.html`-templatessa lisätään `button`-tagiin klikkitapahtuman käsittelijä funktiolle:

```
<button (click)="meneHakuSivulle()" class="btn btn-outline-light" type="submit">Search</button>
```

Seuraavaksi vuorossa on tuoda `Router`-moduuli `app.component.ts`-luokkatiedoston käyttöön ja luoda `meneHakuSivulle()`-funktio, joka lopulta ohjaa käyttäjän nappia painettaessa toiselle sivulle.

Heti `app.component.ts`-tiedoston alkuun `import`-joukkoon lisätään sopiva `import Router`-moduulille:

```
import { Router } from '@angular/router';
```

Jotta varsinaisesti reititin saadaan komponentissa käyttöön, täytyy konstruktorissa alustaa yksityinen muuttuja `reititin`, jonka tyyppi on `Router`. Tätä ohjelmointitekniikkaa kutsutaan Angularissa injektoimiseksi. (Angular 2022m.)

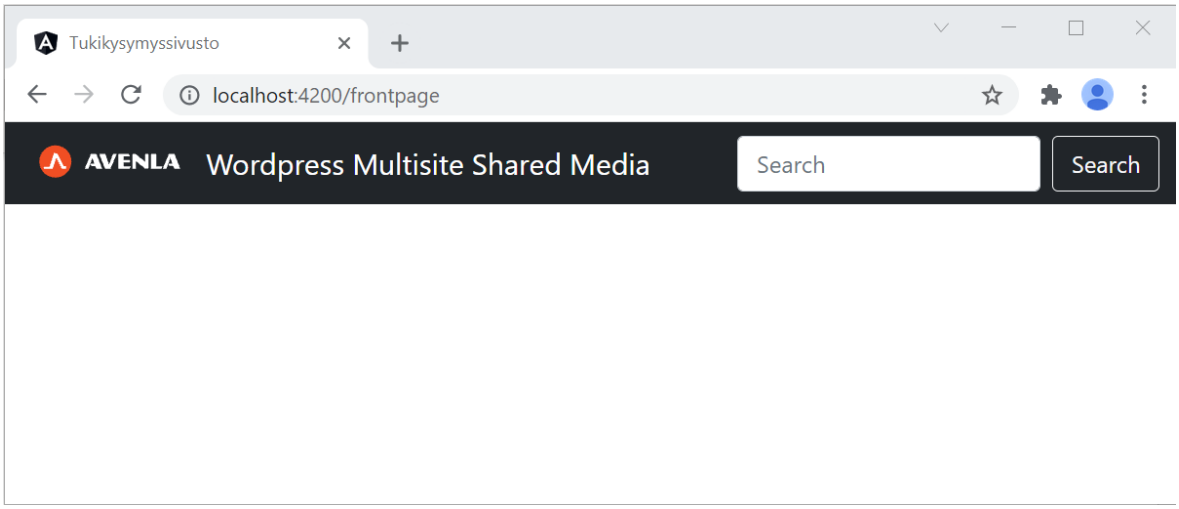
```
  constructor(private reititin: Router){
  }
```

Sitten päästäänkin varsinaiseen asiaan. Luodaan `meneHakuSivulle()`-metodi, jossa aiemmin luodun `reititin`-muuttujan `navigate`-metodilla siirrytään halutulle sivulle, kuten seuraava koodinpätkä täsmentää. Tässä kohtaa käytetään TypeScriptin *template literal* -syntaksia. Käyttämällä backtick-merkkejä `` ` `` se mahdollistaa datan sisällyttämisen tavanomaiseen merkkijonoon. Tämä ominaisuus esiteltiin alun perin ES6:ssa. (Mozilla.)

```
  meneHakuSivulle() {
    this.reititin.navigate([`search/${this.haku}`])
  }
```

Mikäli nyt mennään selaimen puolella testaamaan hakunappia, se toimii, mutta puuttuvan komponentin `HakuComponent` reitityksen takia ei voi ohjata uudelle sivulle vielä. Virheilmoituskin tulostuu kehittäjän konsoliin: `Error: Cannot match any routes. URL Segment: 'search'`.

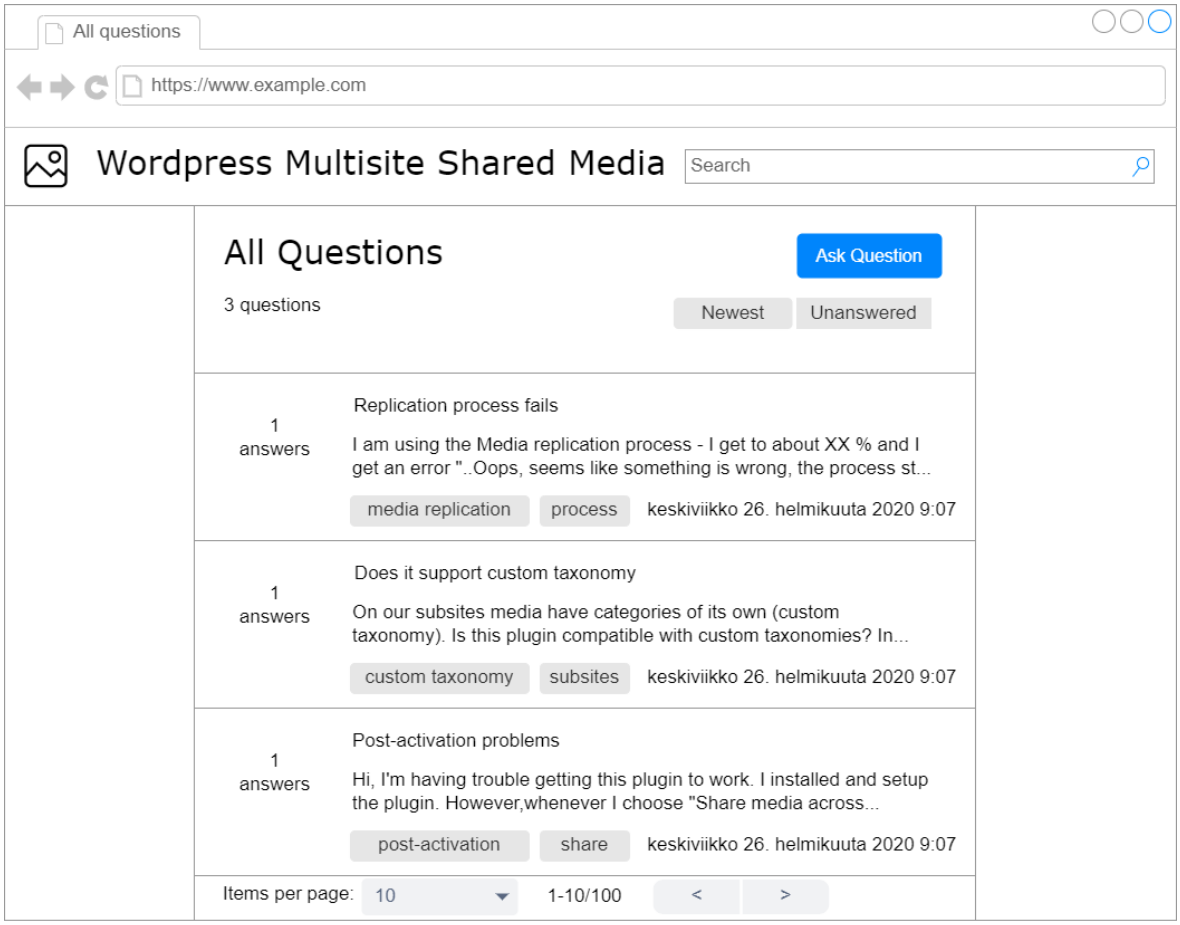
AppComponent sisältää nyt logon, otsikon, hakukentän ja painonapin (kuva 2).



Kuva 2. AppComponent selaimessa

4.2 Etusivu

Itse ohjelmistototeutus jakautuu kolmeen pääsivuun. Etusivulla (kuva 3) ylhäällä on otsikkopalkki, josta löytyy Avenla Oy:n logon lisäksi sivuston otsikko linkkinä ja hakupalkki, johon hakutermin syöttämällä käyttäjä siirretään erilliselle hakutuloksia esittelevälle sivulle. Alempana sivulla näkyvät laskuri siitä, montako kysymystä tietokannasta löytyy yhteensä ja painikkeet, joiden avulla voi suodattaa alla olevaa listanäkymää joko uusimpien tai vastaamattomien perusteella. Vieressä on myös Ask Question -painike, joka ohjaa käyttäjän toiselle sivulle kysymään uutta kysymystä. Sivuston kieli on englanti, koska Avenla myy ohjelmistotuotettaan kansainvälisesti. Listanäkymässä näytetään kunkin kysymyksen olennaisimmat tiedot ja kunkin listatietueen otsikkoa klikkaamalla käyttäjä ohjataan toiselle sivulle tarkastelemaan sen vastauksia. Listanäkymässä vasemmalla näkyy vastauksien määrä, jonka jälkeen näytettäviä tietoja ovat otsikko, pieni ote kysymyksen leipätekstistä, kysymyksen tunnisteet (tags) ja luontipäivämäärä. Pääsivun lopussa on ohjaimet tulosten jakamiseksi useille sivuille (pagination).



Kuva 3. Etusivun käyttöliittymäprototyyppi.

`EtusivuComponentin` tärkein tehtävä on näyttää listaa tietokannasta löytyvistä kysymyksistä, joita Wordpressin REST-rajapinta lähettää pyydettäessä. Tämän lisäksi periaatteessa on mahdollista, että `EtusivuComponentin` vastuulle asetettaisiin myös http-palvelinkutsujen lähettäminen Wordpressin REST-rajapintaan. Angularissa koodin uudelleenkäytettävyyden nimissä tätä ei kuitenkaan suositella. Paljon parempi ratkaisu palvelimen kanssa kommunikointiin on luoda http-palvelinkutsut muodostava `service`, joka importoidaan mukaan kaikkiin sitä tarvitseviin komponentteihin. Näin kaikki palvelinkutsut saadaan luotua yhdessä paikassa keskitetysti (Angular 2022g).

Uuden sovelluksen kysymyksiä käsittelevän `Kysymysservicen` luonti tapahtuu Angularin komentokehotetyökalulla. Annetaan sille seuraava komento: `ng generate service kysymys`. (Angular 2022g.)

Palvelimen vastauksen tyypittäminen

Jotta palvelimen lähettämää JSON-muotoista dataa osataan käsitellä, täytyy ensin ymmärtää hieman sen rakennetta. Tähän tarkoitukseen on suunniteltu Postman-niminen ohjelma, joka kykenee lähettämään http-kutsuja palvelimille ja näyttämään vastauksena saatua JSON-muotoista dataa. Alla on ote palvelimen vastauksesta, kun Wordpressin REST-rajapinnalle lähetettiin kaikkia kysymyksiä koskevaan `/wp/v2/posts`-päätepisteeseen GET-muotoinen pyyntö (Wordpress 2022d). Otteesta voi havaita muun muassa kysymyksen identifioivan tunnisteen, päivämäärän ja linkin:


```
[
    {
        "id": 1,
        "date": "2021-08-04T22:24:09",
        "guid": {
            "rendered": "http://localhost/wordpress/?p=1"
        },
        "modified": "2021-08-04T22:24:09",
        "link": "http://localhost/wordpress/2021/08/04/moikka-maailma/"
```


Koska Angular hyödyntää TypeScriptiä, kehittäjät voivat käyttää vahvaa tyypitystä. Tämä auttaa ohjelmistotuotannossa, koska IDE auttaa löytämään ohjelmistovirheet jo ennen kuin sovellusta ajetaan. Tällä tavoin voidaan korjata ongelmat heti, kun IDE ilmoittaa, että jokin meni pieleen. Tämä konsepti viedään astetta pidemmälle `interfaceilla`. Vaikka ”perus-JavaScript” ei tue `interfaceja`, TypeScript tukee. `Interfacet` ovat yleisiä olio-ohjelmointikielissä, kuten Javassa, PHP:ssä ja C#:ssa. TypeScriptilla niitä voidaan nyt hyödyntää myös selainpäässä. Alla on ote interface-tiedostosta:

```
export interface Kysymys {
    "id": number;
    "date": string;
    "content": {
        "rendered": string;
    };
        "version-history": [
            {
                "count": number;
                "href": string;
            }
        ];
        "wp:attachment": [
            {
                "href": string;
            }
        ];
    }
}
```

TypeScript-kielen dokumentaatiosivut eivät tätä mainitse, mutta kieli ei tue tavuviivoja tai kaksoispisteitä `version-history`- ja `wp:attachment`-avainten nimissä. Jotta vältytään virheilmoituksilta, täytyy avainten nimet kääriä lainausmerkkeihen, kuten edellä on osoitettu. Lainausmerkkeihin kääriminen on tuettuna vasta myöhemmissä TypeScriptin versioissa, sillä internetistä löytyvän keskustelun pohjalta ainakaan vielä vuonna 2013 tukea tälle ei ollut (Cavanaugh 2012).

HttpClient-ominaisuuden tuominen sovellukseen

`HttpClient` on Angularin mekanismi etäpalvelimen kanssa kommunikointiin http-protokollan ylitse (Angular 2022m). `HttpClientin` tuominen Angulariin vaatii kaksi työvaihetta. Ensin se importoidaan `AppModule`-juurimoduuliin seuraavanlaisesti:

```
import { HttpClientModule } from '@angular/common/http';
```

Lopuksi (edelleen `AppModulessa`) `HttpClientModule` lisätään `imports`-tauluun seuraavanlaisesti:

```
@NgModule({
  imports: [
    HttpClientModule,
  ],
})
```

Seuraavaksi `HttpClient` importoidaan `KysymysServicessä`. Samalla kertaa voidaan tuoda `servicelle` myöhemmin tarvittava `Observable`-ominaisuus `rxjs`-kirjastosta palvelinkutsuja varten ja äsken esitelty `Kysymys-interface` tyypittämistä varten:

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Kysymys } from './kysymys';
```

Kommunikointikyvyt etäpalvelimen kanssa saadaan käyttöön, kun lopuksi injektoidaan yksityinen `http`-muuttuja `KysymysServicen` konstruktorissa. `http`-muuttujan tyypiksi määritetään aiemmin mainittu `HttpClient`:

```
constructor(private http: HttpClient) { }
```

Kaikkien kysymysten näyttäminen etusivulla

`KysymysServicen` olennaisin toiminto on pitää sisällään http-palvelinkutsu-funktiot, jotka pyytävät dataa Wordpressin REST-rajapinnalta. Angularissa ohjelmoidessa asynkronisesti täytyy tehdä valinta kahden hieman erilaisen lähestymistavan väliltä. Valittavina ovat JavaScriptiin kuuluva `Promise`-objekti tai RxJS-kirjaston `Observable` (Angular 2022m). `Promise`-objekti edustaa asynkronisen operaation lopullista täyttymistä tai epäonnistumista sekä siitä tulevaa arvoa. `Observablen` idea yleisesti ottaen on samankaltainen. Vaikka molemmat tavat voivat saavuttaa saman halutun lopputuloksen, on `Promisen` syntaksi hieman yksinkertaisempi ja kenties aloittelijalle helpompi omaksua. `Observablen` etuna on laajemmat mahdollisuudet käsitellä datavirtaa eri tavoilla käyttäen sisäänrakennettuja ominaisuuksia. Opinnäytetyössä valittiin käytettäväksi RxJS:n `Observable`-lähestymistapa. Se on Angularin virallisesti suosittelema keino ohjelmoida asynkronisesti, sillä virallinen Tour of Heroes -opas Angularin dokumentaatiosivuilla käyttää tätä keinoa. Kaikki kysymykset palvelimelta pyytävä kutsumetodi näyttää seuraavanlaiselta:

```
  /** Haetaan kaikki kysymykset palvelimelta */
  haeKysymykset(): Observable<Kysymys[]> {
    return this.http.get<Kysymys[]>("http://localhost/wordpress/wp-
json/wp/v2/posts");
  }
```

Kaikki `HttpClient`-metodit palauttavat RxJS:n (Reactive Extensions for JavaScript) `Observablen` jonkin tietueen. HTTP on pyyntö/vastaus-periaatteella toimiva protokolla. Kun tehdään pyyntö, palautetaan yksi vastaus. Yleisesti ottaen `Observable` voi palauttaa useamman arvon ajan kuluessa. `HttpClientin` sisällä toimiva `Observable` laskee kuitenkin liikkeeseen aina yhden arvon ja lopettaa sitten toimintansa, eikä eritä toista arvoa enää koskaan. Tämä tietty `HttpClient.get()`-funktiokutsu palauttaa `Observable<Kysymys[]>`-tyyppisen tietueen, joka tarkoittaa `Kysymys`-taulujen (arrays) kanssa toimivaa

`Observablea`. Käytännössä se palauttaa yhden Kysymys-taulun. `Observableista` puheen ollen tärkeää ei ole se, mihin tarkkaan aikaan vastaus saapuu eli kestääkö datan saapumisessa sekunti vai viisi minuuttia. Olennaisempaa on se, saapuuko data perille normaalisti vai tapahtuiko tietovirrassa matkalla sen keskeyttävä virhe. Näihin kahteen tilanteeseen reagoiminen on asynkronisessa ohjelmoinnissa olennaista. Tietokoneohjelmoinnissa asynkronisuus viittaa tapahtumiin, jotka ovat riippumattomia pääohjelmavirrasta. Toisin sanoen asynkroninen funktio lähdekoodissa suoritetaan itsenäisesti täysin eri aikaan kuin muu lähdekoodi, joka tyypillisesti valmistuu selkeästi nopeammin kuin tuo funktio.

`HttpClient.get()` palauttaa vastauksen oletusarvoisesti tyypittämättömänä JSON-datana. Asettamalla valinnainen tyyppimääritys `<Kysymys[]>` saadaan käyttöön TypeScriptin kyvykkyydet, jotka vähentävät virheiden määrää sovelluksen kääntämisessä. Etäpalvelimen datarajapinta määrittelee JSON-datan muodon. Wordpressin datarajapinta palauttaa kysymysdatan tauluna. On huomionarvoista, että muut rajapinnat saattavat haudata datan objektin sisälle. Silloin voidaan joutua kaivamaan data ulos prosessoimalla `Observable`-vastaus `RxJS`:n `map()`-operaattorilla.

Seuraava vaihe on injektoida servicen toiminnallisuudet `EtusivuComponentin` luokkatiedostoon. Se tapahtuu importoimalla `Kysymys`-tyypitystiedosto ja `KysymysService` seuraavasti:

```
import { Kysymys } from '../kysymys';
import { KysymysService } from '../kysymys.service';
```

Injektoiminen viimeistellään määrittelemällä luokkatiedoston konstruktorissa yksityinen muuttuja, joka on tyyppiä `KysymysService`:

```
constructor(private kysymysService: KysymysService) { }
```

Nämä kaksi edeltävää vaihetta toistetaan kaikissa tulevissa komponenteissa, joissa tarvitaan servicen tarjoamia palveluita.

Seuraavassa koodinpätkässä on esitelty kaksi eri tapaa komponentin ottaa yhteyttä serviceen ja hakea kysymykset komponentin paikalliseen muuttujaan:

```
kysymykset: Kysymys[] = [];

// synkroninen funktio
haeKysymykset(): void {
  this.kysymykset = this.kysymysService.haeKysymykset();
}

// asynkroninen funktio
haeKysymykset(): void {
  this.kysymysService.haeKysymykset()
      .subscribe(kysymykset => this.kysymykset = kysymykset);
}

ngOnInit(): void {
  this.haeKysymykset();
}
```

`Observable.subscribe()` on olennainen ero näissä kahdessa funktiossa. Ylempi funktio asettaa komponentin `kysymykset`-muuttujaan arvoksi taulun. Tämä arvon asettaminen tapahtuu synkronisesti, ikään kuin palvelin voisi palauttaa kysymysdatan välittömästi tai selain voisi jäädyttää käyttöliittymän siksi, kunnes se odottaa palvelimen vastausta. Tämä ei tule toimimaan, kun `KysymysService` tekee oikeita pyyntöjä etäpalvelimelle. (Angular 2022n.)

Alempi funktio odottaa, kunnes `Observable` on laskenut liikkeelle kysymysten taulun, mikä voi tapahtua sekunnissa tai useammassa minuutissa tästä hetkestä. `subscribe()`-metodi välittää liikkeelle lasketun taulun callbackiin, joka asettaa arvon komponentin `kysymykset`-muuttujaan. Tämä asynkroninen lähestymistapa toimii varmasti, kun `KysymysService` pyytää kysymyksiä palvelimelta (Angular 2022n).

Etusivun http-kutsut ja komponentin haeKysymykset()-funktio

Etusivulla on tarkoitus esittää kysymyksistä otsikko, pieni pätkä runkotekstistä, luontipäivämäärä sekä avainsanat ja vastausten lukumäärä. Kuten aiemmin näytettiin, kaikki kysymykset saatiin haettua palvelimelta, kun sille lähetettiin `GET`-muotoinen pyyntö palvelimen osoitteeseen `/wp/v2/posts` (Wordpress 2022d). Tämä ohjelmointirajapinnan päätepiste ei kuitenkaan suoraan palauta kysymysten avainsanojen ihmisen luettavia nimiä, vaan vain avainsanojen `id`-tunnistetiedot. Tästä syystä haettaessa kaikkia kysymyksiä palvelimelta Angularin `HttpClient.get()`-kutsulla, täytyy samassa yhteydessä kutsua myös palvelimen REST-ohjelmointirajapinnan päätepistettä `/wp/v2/tags?include=`. Tämä päätepiste vastaa pyydettäessä tarjoamalla tiedot kaikista kysymysten avainsanoista niin, että

vastauksessa näytetään vain `include`-parametrin osoittamat avainsanat. `include`-parametri rajaa tulosjoukon siten, että kaikkien avainsanojen joukosta hakunumeron perusteella palautetaan vain halutut avainsanat. (Wordpress 2022e.) Seuraava koodinpätkä osoittaa `KysymysServicen` metodin, joka hakee palvelimelta halutut avainsanat. Metodille annetaan merkkijonomuotoinen `numerot`-muuttuja parametriksi, jotta komponentista käsin voidaan lähettää haluttujen avainsanojen `id`:t servicen kautta palvelimelle kutsuksi.

```
  /** Haetaan haluttujen id-numeroiden perusteella tietyt avainsanat */
  haeAvainSanat(numerot: string): Observable<any> {
    return this.http.get<any>(`http://localhost/wordpress/wp-
json/wp/v2/tags?include=${numerot}`);
  }
```

Seuraava koodinpätkä osoittaa `KysymysServicen` metodin, joka hakee palvelimelta haluttujen ID-tunnisteiden perusteella tietyt vastaukset kysymyksiin. `Post`-parametri rajaa tulosjoukon siten, että vain aiemmin pyydettyihin kysymyksiin liittyvät vastaukset haetaan, ei suinkaan aivan kaikkia tietokannasta löytyvät vastaukset. (Wordpress 2022e.)

```
  /** Haetaan haluttujen id-numeroiden perusteella tietyt vastaukset */
  haeVastaukset(numerot: string): Observable<any> {
    return this.http.get<any>(`http://localhost/wordpress/wp-json/wp/v2/com-
ments?post=${numerot}`);
  }
```

Kun työskennellään Wordpressin REST-rajapinnan kanssa, tarvitsee sille tässä etusivun tapauksessa lähettää kolme `GET`-muotoista pyyntöä. Ensimmäinen pyyntö lähetetään osoitteeseen `/wp/v2/posts`. Tämän pyynnön JSON-muotoinen vastaus näyttää seuraavanlaiselta:

```
[
    {
        "id": 9,
        "title": {
            "rendered": "Hello World! 3"
        },
        "tags": [
            2,
            3,
            4
        ]
    }
    ...
]
```

Numerot `tags`-tauluissa eivät ole varsinaisia ihmisen luettavia avainsanoja, vaan vain avainsanojen yksilöiviä ID-tunnisteita. On myös huomionarvoista, että edellä esitetty JSON-koodinpätkä edustaa vain demotarkoituksessa luotua kysymysdataa. Rakenne on kyllä

lopullista tuotantovalmista ohjelmistoa edustava, mutta sisältö on vain nopeasti luotua täytettä. Kun vastaus ensimmäiseen pyyntöön on saatu palvelimelta, tarvitsee lähettää toinen `GET`-muotoinen pyyntö osoitteeseen `/wp/v2/tags?include=2,3,4` samalla lähettäen yksilöllisten avainsanojen ID-tunnisteet. Toisin sanoen, toinen palvelinpyyntö on riippuvainen ensimmäisen pyynnön vastauksesta. Toisen pyynnön JSON-muotoinen vastaus näyttää seuraavanlaiselta:

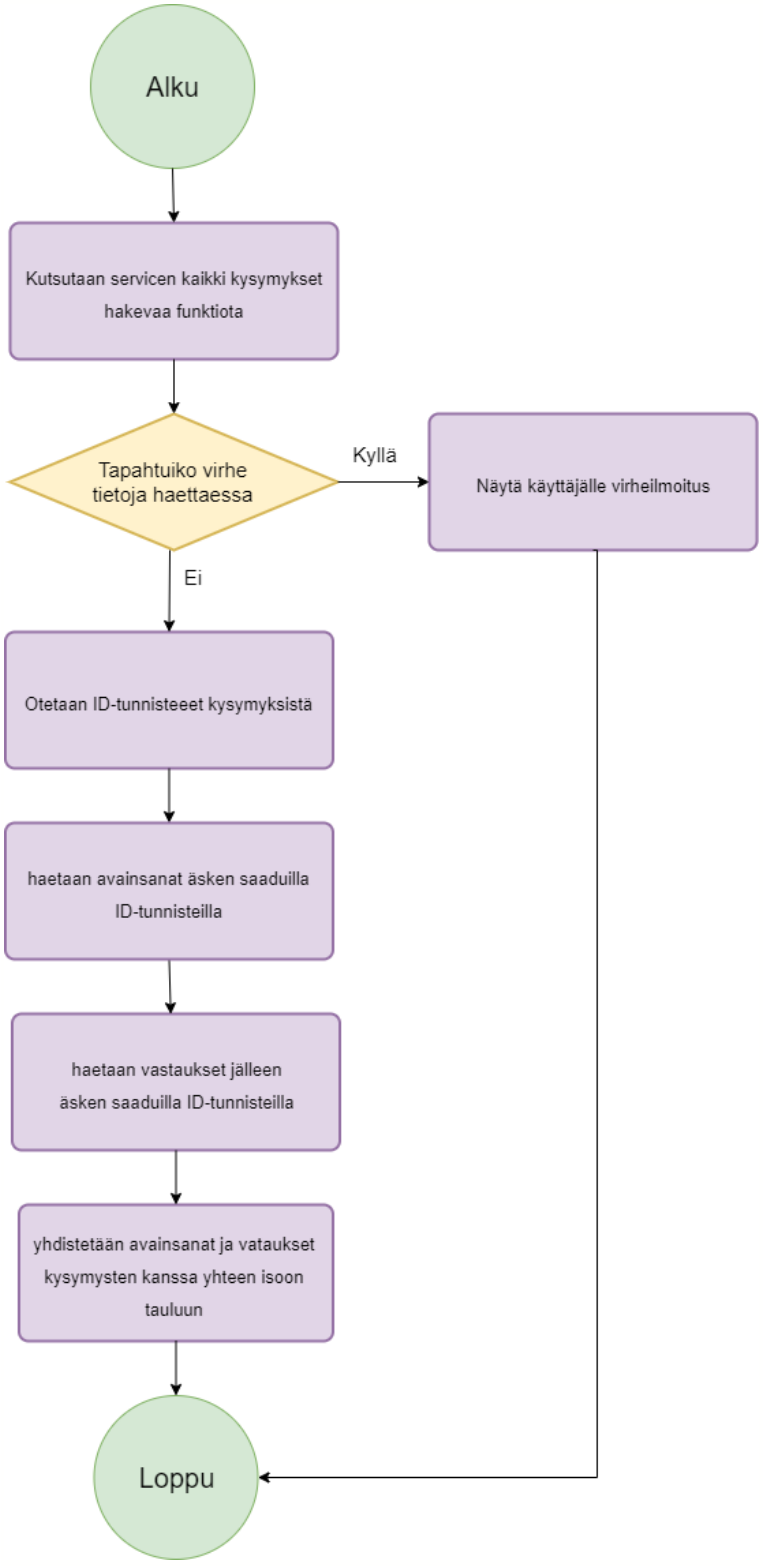
```
[
    {
        "id": 2,
        "name": "test"
    },
    {
        "id": 3,
        "name": "test2"
    }
    ...
]
```

Kolmas ja viimeinen palvelimelle lähetettävä kutsu koskee kysymysten vastauksia, tarkemmin kuinka monta niitä on määrältään kutakin kysymystä kohden. Palvelimelle täytyy lähettää `GET`-muotoinen pyyntö osoitteeseen `/wp/v2/comments?post=9,7,1`. Tähän pyyntöön palvelin vastaa tarjoamalla dataa kaikista vastauksista, jotka se löysi annetuille kysymysnumeroille. Koska kyseessä on käytännössä kuin mikä tahansa Wordpress-sovellus, on täysin mahdollista, että jotkin kysymykset eivät sisällä lainkaan vastauksia, ainakaan vielä. Kolmannen pyynnön JSON-muotoinen vastaus näyttää seuraavanalaiselta:

```
[
    {
        "id": 2,
        "post": 9,
        "date": "2022-01-15T14:22:16",
        "content": {
            "rendered": "<p>Example comment for Hello World! 3</p>\n"
        }
    }
]
```

Avainsanojen ja vastausten hakeminen tietokannasta muun kysymysdatan ohella on hieman mutkikas prosessi, ja siihen on olemassa pääasiassa kaksi tapaa. Ensimmäinen tapa on yksinkertaisesti luoda sisäkkäiset `subscribe()`-metodit, jotka hakevat datan servicen kautta tietokannasta. Tämä keino on kuitenkin melko huono lähestymistapa, sillä koodista tulee sotkuista ja käytännössä joudutaan yhden asian hoitamiseksi lähettämään kolme erillistä http-kutsua tietokantaan ja odottamaan vastausta kaikkiin. Parempi keino asian hoitamiseksi on käyttää RxJS:n `switchMap`- ja `forkJoin`-operaattoreita. `SwitchMap`-operaattori sallii `Observablesta` toiseen hyppäämisen datavirran pysyessä eheänä, kun taas

`forkJoin`-operaattorilla voidaan lähettää palvelimelle peräkkäisesti http-kutsuja ja yhdistää data siistiin pakettiin. RxJS:n `switchMap`-operaattorin toimintaperiaate tässä tapauksessa on seuraavanlainen:



Kuvio 4. Vuokaavio etusivun datan hakuprosessista.

Vuokaavio osoittaa kuinka data haetaan etusivua varten. Ensin kutsutaan servicen kaikki kysymykset hakevaa funktiota. Mikäli yhteyttä tietokantaan ei saada, näytetään käyttäjälle virheilmoitus. Onnistuessaan prosessin seuraava vaihe on kaapata ID-tunnisteet haetuista kysymyksistä. Sitten kutsutaan avainsanat hakevaa funktiota äsken saaduilla ID:illä, minkä jälkeen on vuorossa kutsua vastaukset hakevaa funktiota jälleen äsken saaduilla ID-tunnisteilla. Lopuksi koodi yhdistää avainsanat ja vastaukset kysymysten kanssa isoon kokonaistauluun.

Liian pitkien merkkijonojen katkaiseminen

Tässä vaiheessa, kun kysymykset, niiden avainsanat ja vastaukset on saatu näkymään etusivulla, on helppo hieman muokata komponentin funktiota ja lisätä merkkijonon tiettyyn pituuteen katkaiseva funktiokutsu. Seuraava koodinpätkä osoittaa sen, kuinka merkkijonot, joita ovat etusivun otsikot ja leipätekstit, lyhennetään. Siinä tarkistetaan annetun merkkijonon pituus siten, että jos se on pidempi kuin annettu maksimimäärä `n`, lyhennetään se JavaScriptin `substr`-funktiolla ja lisätään päätemerkki (…) loppuun. Muutoin annettu merkkijono saa olla rauhassa.

```
  lyhenna(merkkijono: string, n: number) {
    return (merkkijono.length > n)
      ? merkkijono.substr(0, n - 1) + '...' : merkkijono;
  };
```

Komponentin `haeKysymykset()`-funktioon on nyt helppo lisätä forOf-silmukka, joka kutsuu edellä esitettyä funktiota seuraavanlaisesti:

```
        for (let kysymys of yhdistettyArray) {
          kysymys.title.rendered = this.lyhenna(kysymys.title.rendered, 100)
          kysymys.excerpt.rendered = this.lyhenna(kysymys.excerpt.rendered,
300)
        }
```

Kokeilun jälkeen vaikuttaa sille, että sopiva pituus kysymysten otsikoille on 100 merkkiä ja leipäteksteille 275 merkkiä. Näin etusivua lukeva ihminen saa nopeasti selville, mitä kukin kysymys pääpiirteittäin sisältää, ilman, että informaatiota olisi liian paljon luettavaksi tai että sivuston ulkoasusta tulisi sotkuinen.

Kysymysten kokonaismäärän hakeminen http-headerista

Haettaessa kysymysten kokonaismäärää Wordpressin rajapinnasta törmättiin ensimmäiseen isoon ohjelmistotekniseen ongelmaan. Jokainen http-kutsu rajapinnalle palauttaa vastauksen, joka sisältää varsinaisen http-rungon sekä muuta oheisdataa sisältävät http-headerit. Wordpressin dokumentaatiosivujen (2022c) mukaan kysymysten kokonaismäärä

on löydettävissä `X-WP-Total`-nimisestä headerista. Jotta Angularissa voitaisiin päästä käsiksi headereihin, tarvitsee `KysymysServicen` http-metodiin tehdä muokkauksia.

Metodin palautusarvoksi täytyy asettaa `Observable<HttpResponse<Kysymys[]>>` tavanomaisen `Observable<Kysymys[]>`-palautusarvon sijaan. Metodi viimeistellään lisäämällä `HttpClient.get()`-metodiin `{ observe: 'response' }` -lisäoptio. (Angular 2022m.) Tässä vaiheessa tuleekin eteen tyypittämisen yhteensopivuusongelma. Komponentin `switchMap()`-funktio, jossa käytetään parametrina `Kysymys[]`-tyyppistä muuttujaa, ei ole yhteensopiva KysymysServicen `HttpResponse`-palautusarvollisen metodin kanssa.

Ongelma kierretään siten, että `switchMap()`-funktion tarvitseman servicen `haeKysymykset()`-metodin annetaan palauttaa tavanomainen `Observable<Kysymys[]>`-arvo. Serviceen luodaan uusi `haeHeaderit()`-funktio, joka itsenäisesti käsittelee `observe`-lisäoptiolla http-vastauksen headereita:

```
  /** Haetaan http-vastauksen headerit observe-lisäoptiolla */
  haeHeaderit(): Observable<HttpResponse<Kysymys[]>> {
    return this.http.get<Kysymys[]>("http://localhost/wordpress/wp-
json/wp/v2/posts",
      { observe: 'response' });
  }
```

Komponentin luokassa servicen funktio tilataan `subscribe()`-metodilla. Lopuksi `X-WP-Total`-headerin arvo saadaan kaivettua esille ja asetettua komponentin paikalliseen muuttujaan seuraavalla koodinpätkällä:

```
this.laskuri = kysymykset.headers.get('X-WP-Total');
```

Etusivukomponentin template ja kysymysdata

Kysymysten näyttämiseen `EtusivuComponentin` `templatessa` tarvitaan `NgForOf`-direktiivin luoma silmukka. Sen toimintaidea on luoda väliaikainen `kysymys`-niminen muuttuja komponentin `yhdistettyArray$`-taulukkomuuttujasta kunkin yksittäisen kysymyksen kohdalla, jonka jälkeen se tulostaa selaimeen näkyviin palvelimelta pyydetyt tietueet siten, että uusin kysymys näkyy web-selaimessa ensimmäisenä. (Angular 2022j.) Ote silmukasta näyttää seuraavanlaiselta:

```
<li *ngFor="let kysymys of yhdistettyArray$ | async">
        <div>{{ kysymys?.comments?.length }}<br>answers</div>
        <div><b>{{ kysymys.title.rendered }}</b>
            <br><p [innerHTML]='kysymys.excerpt.rendered'></p>
        </div>
        <div>
            <span *ngFor="let tagin_nimi of kysymys.tag_names">
```

```
                <a href="#">{{ tagin_nimi }}</a> 
            </span>
        </div>
            <span>{{ kysymys.date | date:'medium' }}</span>
    </div>
</li>
```

Koska komponentin `yhdistettyArray$`-muuttuja ei ole varsinainen taulu itsessään, vaan `Observable`, joka sisältää tuon taulun, täytyy `ngForOf`-lauseesen lisätä `async`-pipe, jotta template odottaa `Observablen` valmistumista ja arvon saamista ennen kuin lähdetään luomaan toistuvaa HTML-rakennetta verkkosivulle (Angular 2022c). Etusivulla ei ole tarkoitus näyttää vastausten tekstisisältöä, vaan vain niiden lukumäärä. Tämän vuoksi valinnainen `kysymys?.comments?`-väliaikaismuuttuja saa seurakseen `length`-metodin, joka palauttaa vastausten lukumäärän taulusta. Koodinpätkässä näkyvä `[innerHTML]`-sidonta (property binding) tarkoittaa raa'an HTML-muotoisen merkkijonon asettamisen templaten `p`-tagiin sellaisena, kuin sen sisältämät tagit haluavat sen piirtyvän (Angular 2022l). Avainsanat sijaitsevat omassa taulussaan `yhdistettyArray$`-taulun sisällä. Siksi tarvitaan sisäkkäinen `NgForOf`-silmukka, jotta voidaan käydä kunkin kysymyksen kukin avainsana läpi ja piirtää ne verkkosivulle. Päivämäärä muotoillaan Angularin sisäänrakennetulla `date`-pipellä, jolloin tietokannassa näkyvä päivämäärä, joka näkyy muodossa `2021-08-04T22:24:09` renderöidään web-sivulle näkyviin muodossa Aug 4, 2021, 10:24:09 PM.

Etusivun kysymysdatan jakaminen sivuiksi (pagination)

Kun ollaan Wordpressin taustajärjestelmien kautta tekemisissä sellaisen sovelluksen kanssa, jonka tietokanta saattaa pitää sisällään kymmeniä, ellei jopa satoja tietueita, täytyy kysymysdata sivuttaa. Tämä tehdään siksi, että mikäli haettaisiin tietokannasta yhdelle web-sivulle aivan kaikki mahdollinen kysymysdata, kestäisi datan hakemisessa palvelimelta liian kauan aikaa ja sovelluksen käyttökokemus loppukäyttäjän näkökulmasta olisi melko huono. Lisäksi yhdelle sivulle tulisi liian paljon informaatiota kerralla ja sen perkaaminen olisi liian työlästä ajatustyötä. Kun data sivutetaan, löydetään tasapaino riittävän informaatiomäärän ja tarpeeksi nopeiden sivulatausaikojen kanssa.

Oletusarvoisesti Wordpress sivuttaa datan niin, että kerrallaan näytetään 10 tietuetta. Tätä lukua on kuitenkin tarpeen tullen mahdollista manipuloida asettamalla se mihin tahansa väliltä 1—100. (Wordpress 2022c.)

Sivutustoiminnallisuus luodaan tekemällä muutoksia reititysmoduulin `reitit`-tauluun. `EtusivuComponentin` polkuun lisätään `page`-niminen parametri ja oletusarvoinen, tyhjä reitti uudelleenohjataan polkuun `/frontpage/1`, jossa numero edustaa haluttua sivunumeroa.

Numero yksi valittiin siksi, koska sitä todennäköisimmin käyttäjä olikin etsimässä sovelluksesta, mikäli hän jostain syystä päätyi hukkaan. Se on siis luonnollisin aloituspiste, kun käyttäjä ensimmäistä kertaa vierailee hänelle uudella sivustolla.

```
const reitit: Routes = [
  { path: 'frontpage/:page', component: EtusivuComponent },
  { path: '', redirectTo: '/frontpage/1', pathMatch: 'full' }
];
```

`KysymysServicen` metodien täytyy tukea uutta ominaisuutta. `haeKysymykset()`-funktioon lisätään `sivu`-parametri, ja käyttäen ES6:n niin kutsuttua ”back-tick”-merkintätapaa (`), voidaan `HttpClient.get()`-metodin merkkijonoon upottaa annetun `sivu`-parametrin muuttuja, kuten seuraava koodinpätkä osoittaa.

```
  haeKysymykset(sivu: number): Observable<Kysymys[]> {
    return this.http.get<Kysymys[]>(`http://localhost/wordpress/wp-
json/wp/v2/posts?page=${sivu}&per_page=3`)
  }
```

Wordpressin REST-rajapinta osaa siis ottaa syötteekseen `page`- ja `per_page`-URL-parametrit, joiden perusteella se tietää, montako tietuetta se lähettää vastauksessaan takaisin ja miltä sivulta tietueet napataan. `KysymysServicen` `haeHeaderit()`-funktio ei ota syötteekseen yhtään parametria, mutta sen sisältämän URL-osoitteeen `per_page`-parametriin annetaan sama luku kuin `haeKysymykset()`-funktiossa.

`EtusivuComponentin` luokassa tehdään useita muutoksia. `haeKysymystenMaara()`-funktiossa uutta on rivi koodia, joka asettaa paikallisen `sivumaara`-muuttujan arvoksi `X-WP-TotalPages`-http-headerin palauttaman numeron.

```
this.sivumaara = +kysymykset.headers.get('X-WP-TotalPages');
```

Sivutustoiminnallisuutta luodessa havaittiin, että käyttäjän klikatessa web-sivulla näkyviä sivutuspainikkeita sovellus kyllä siirtyi toiseen sivunumeroa edustavaan osoitteeseen, mutta itse sisältö sivulla ei muuttunut. Tämä johtui alun perin siitä, että `yhdistettyArray$`-muuttujan arvoksi asetettiin `this.haeKysymykset()`-funktiokutsu `ngOnInit()`-funktiossa, joka siis sisältää koodia, joka pitäisi tulla ajetuksi jokaisen sivulatauksen yhteydessä. Kävi ilmi, että sivutustoiminnallisuus saatiin toimimaan ja sisältö vaihtumaan web-sivulla nappien painalluksen jälkeen, kun koodin ajopaikaksi vaihdettiinkin komponentin konstruktori. Ero konstruktorissa ja `ngOnInit()`-funktiossa ajettavan lähdekoodin välillä voi ensi alkuun vaikuttaa mitättömältä, mutta niissä on yksi keskeinen ero. Siinä missä `ngOnInit()`-funktion sisältämä koodi tulee ajetuksi vain, kun web-sovelluksessa vaihdetaan komponentista toiseen, kuten vaikkapa etusivun ja hakusivun välillä, konstruktori osaa tehdä yhteistyötä templaten `routerLink`-linkkien kanssa ja niitä klikatessa päivittää myös sivun kysymys-

sisällön vaihtumaan, kun nappeja painetaan web-sivulla. Ero on hienoinen, mutta merkittävä. Kun käyttäjä klikkaa `routerLink`-linkkiä web-sivulla, ei `ngOnInit()`-funktion koodia ajeta, koska periaatteessa ei koskaan poistuttukaan sen vastuualueelta. Angular ei virallisessa dokumentaatiossaan oikeastaan suosittele tapaa, jossa konstruktorin vastuulle asetetaan paljon toiminnallisuuksia, mutta tämä lienee sopiva poikkeus sääntöön. (Jain 2016.) Lopullinen `ngOnInit()`-funktio näyttää nyt siis seuraavanlaiselta:

```
  ngOnInit(): void {
    this.haeKysymystenMaara();
  }
```

Konstruktori vastavuoroisesti saa seuraavat toiminnot itseensä:

```
  constructor(
    private kysymysService: KysymysService,
    private reitti: ActivatedRoute
  ) {

    this.yhdistettyArray$ = this.reitti.params.pipe(
      switchMap((params) => {
        if (!params.page) {
          return of([]);
        }
        this.nykyinenSivu = +params.page;
        return from(this.haeKysymykset(+params.page));
      })
    );

  }
```

`ActivatedRoute`-tyyppiä olevasta `reitti`-muuttujasta kaivetaan URL-osoitteen sivunumero-parametri ja sitä hyödyntäen kutsutaan komponentin luokkatiedoston `haeKysymykset()`-funktiota. Luonnollisesti komponentin luokkatiedosto viimeistellään lisäämällä halutut kysymykset hakevaan `haeKysymykset()`-funktioon sivunumero-parametri, jonka arvo välitetään edelleen servicelle.

Sivutusjärjestelmän taustatekijät saadaan lopulta käyttöön, kun komponentin HTML-templatessa määritetään sivutusta kontrolloivat painonapit:

```
    <nav aria-label="Page navigation">
        <ul class="pagination">
            <li class="page-item" [ngClass]="{'disabled': nykyinenSivu === 1
}">
                <a routerLink="/frontpage/{{nykyinenSivu-1}}" class="page-
link" href="#">Previous</a>
            </li>
```

```
            <li *ngFor="let sivu of [].constructor(sivumaara); let i = in-
dex" class="page-item"
                [ngClass]="{'active': i+1 === nykyinenSivu }">
                <a routerLink="/frontpage/{{i+1}}" class="page-link"
href="#">{{ i+1 }}</a>
            </li>
            <li class="page-item" [ngClass]="{'disabled': nykyinenSivu ===
sivumaara }">
                <a routerLink="/frontpage/{{nykyinenSivu+1}}" class="page-
link" href="#">Next</a>
            </li>
        </ul>
    </nav>
```

`routerLink`-lähestysmistavalla linkkitageihin lisätään sivunumeromuuttujat luokkatiedostosta siten, että rykelmän vasemmanreunimmainen painonappi ohjaa aina nykyistä URL-osoitteesta haettua sivunumeroa yhtä pienempään sivuun. Vastavuoroisesti rykelmän oikeanreunimmainen nappi ohjaa yhtä lukua suurempaan sivuun. Tärkeää on muistaa `ngClass`-sidonnalla kytkeä ohjausnapit pois päältä, mikäli ollaan numerollisesti jommassakummassa ääripäässä sivuja. `ngFor`-silmukalla varsinaiset sivunumeroita näyttävät napit käydään yksi kerrallaan läpi niin monta kertaa kuin komponentin luokkatiedoston `sivumaara`-muuttujan arvo määrittää. Lopuksi sivutuksen ohjausnapit viimeistellään lisäämällä `active`-CSS-määrite nykyisen valitun sivunumeron päälle antaen sille selkeästi erottuvan sinisen värin.

Tulosten järjestäminen päiväyksen ja vastausten määrän perusteella

Jo tässä vaiheessa etusivu sisältää melko hyvät mahdollisuudet lajitella kysymyksiä. Tätä parannetaan entisestään tärkeillä painonappuloilla, jotka sallivat tulosten järjestämisen sen mukaan, kuinka monta vastausta kullakin kysymyksellä on sekä kuinka uusia kysymykset ovat. Kysymysten järjestäminen vastausten lukumäärän perusteella on erityisesti tärkeä etusivun ominaisuus, sillä se mahdollistaa helposti sellaisten kysymysten näyttämisen, joilla ei vielä ole vastauksia. Tämän avulla sovelluksen pääkäyttäjä voi helposti löytää tietokannasta ne kysymykset, joihin tarvitsee kiinnittää huomiota. Kysymysten järjestäminen luontipäivämäärän perusteella on helppoa, sillä Wordpressin REST-rajapinnan `orderby`-parametri tarjoaa sisäänrakennetun mahdollisuuden antaa sille arvoksi `date` eli päivämäärän.

Kysymysten järjestäminen vastausten määrän perusteella ei suoraan toimi Wordpressin REST-rajapinnan kanssa. Jotta se voidaan mahdollistaa, täytyy `rest_{post_type}_collection_params`-funktioon lisätä `comment_count`-suodatin jokaiselle ”Post Typelle”, joka halutaan järjestää vastausten lukumäärän perusteella. Wordpressin REST-rajapinnassa on jo `orderby`-parametri toiminnassa oletusarvoisesti, mutta se hyväksyy vain seuraavia

arvoja: `author, date, id, include, modified, parent, relevance, slug, include_slugs, title`. Mikäli rajapintaa kutsutaan `comment_count`-arvolla, palauttaa se virheilmoituksen `rest_invalid_param`. Wordpress sisältää kuitenkin suodattimen, joka sallii muokata aiemmin mainittua arvojen listaa ja järjestää kysymykset haluttuun `comment_count`-järjestykseen. Tämän suodattimen nimi on `rest_{post_type}_collection_params`. (Ross 2018.) Seuraava koodinpätkä ottaa käyttöön halutun toiminnallisuuden:

```
// Posts saa käyttöönsä orderby=comment_countin
// ja comment_count lisätään sallittujen orderby-arvojen listalle
add_filter( 'rest_post_collection_params', 'filter_add_rest_orderby_params', 10, 1 );

function filter_add_rest_orderby_params( $params ) {
          $params['orderby']['enum'][] = 'comment_count';
          return $params;
}
```

Koodia varten luodaan uusi Wordpress-liitännäinen. Liitännäisten tekeminen on helppoa ja internetistä löytyy selkokielinen opas, mikäli tämä ei ole entuudestaan tuttua. Oppaan osoite on https://www.wpbeaverbuilder.com/creating-wordpress-plugin-easier-think/

Kun liitännäinen on luotu, se aktivoidaan erikseen Wordpressin selainpohjaisessa hallintapaneelista Activate-nappulaa painamalla. Nyt rajapintaa on mahdollista kutsua seuraavalla osoitteella: `/wp-json/wp/v2/posts?orderby=comment_count&order=asc`. Koodi toimii, koska Wordpress tunnistaa `comment_count`-parametriarvon automaattisesti, se vain täytyi ottaa käyttöön.

Jotta voidaan `EtusivuComponentin` URL-parametreista hakea halutut tiedot eli joko `comment_count` tai `date` sekä nouseva ja laskeva järjestys, täytyy `app-routing.moduleen` tehdä hieman muokkauksia. Polkuihin lisätään parametripaikat edellä mainituille tiedoille, jolloin muokatut polut näyttävät seuraavanlaisilta.

```
const reitit: Routes = [
  { path: 'questions/:page/:orderby/:order', component: EtusivuComponent },
  { path: '', redirectTo: '/questions/1/date/desc', pathMatch: 'full' }
];
```

Tyhjä polku kohdistetaan uudelleenohjattavaksi osoitteeseen http://localhost:4200/questions/1/date/desc. Tämä tarkoittaa, että kun käyttäjä ensimmäistä kertaa navigoi sovellukseen, hänelle näytetään ensimmäisen sivun kysymykset järjestettynä päivämäärän

perusteella laskevaan järjestykseen eli niin, että uusin kysymys näkyy ensimmäisenä. Esimerkiksi Stack Overflow -verkkosivusto näyttää oletusarvoisesti kysymykset samalla toimintaperiaatteella.

EtusivuComponentin template-tiedostoon lisätään Stack Overflow -sivustollakin näkyvät linkkinappi Uusi kysymys -sivulle, joka tehdään hieman myöhemmin sekä painonapit kysymysdatan järjestämiseen edellä kuvatuilla keinoilla:

```
            <div class="col-md-2">
                <div class="btn-group" role="group" aria-label="Basic exam-
ple">
                    <a routerLink="/questions/1/date/desc" class="btn btn-
outline-secondary" role="button" aria-disabled="true">Newest</a>
                    <a routerLink="/questions/1/comment_count/asc"
class="btn btn-outline-secondary" role="button" aria-disabled="true">Unan-
swered</a>
                </div>
            </div>
```

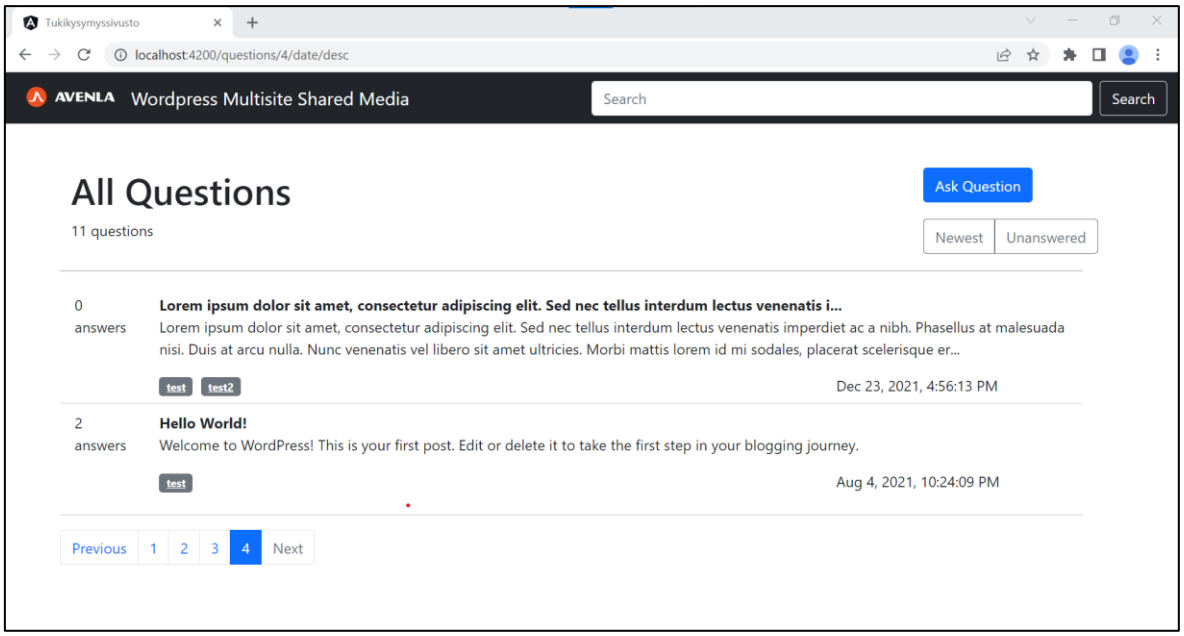
Käyttäen hyväksi Bootstrap ruudukkoasetelmäjärjestelmän `container`-, `col-md-*`- ja `row`-CSS-määreitä voidaan luoda responsiivisesti skaalautuva selain-näkymä, joka näyttää hyvältä (Bootstrap 2022). `KysymysServicen` metodi `haeKysymykset()` täytyy päivittää ajan tasalle, jotta se voisi tukea uusia parametreja. Funktion argumentteihin lisätään `järjestys`- ja `laskevaNouseva`-nimiset muuttujat, jotka lopulta upotetaan `HttpClient-get()`-funktion URL-osoitteeseen, joka kohdistuu Wordpressin REST-rajapintaan.

```
  /** Haetaan kaikki kysymykset palvelimelta */
  haeKysymykset(sivu: number, jarjestys: string, laskevaNouseva: string):
Observable<Kysymys[]> {
    return this.http.get<Kysymys[]>(`http://localhost/wordpress/wp-
json/wp/v2/posts?page=${sivu}&per_page=3&orderby=${jarjestys}&order=${las-
kevaNouseva}`)  }
```

Kun `EtusivuComponentin` luokkatiedoston `haeKysymykset()`-funktioon on lisätty tarvittavat datan järjestämistä koskevat argumentit, voidaan funktiota kutsua konstruktorissa seuraavalla tavalla:

```
this.haeKysymykset(+params.page, params.orderby, params.order)
```

Etusivu on nyt viimeistelty loppuun asti ja se näyttää tältä (kuva 5). Huomionarvoista on, että toistaiseksi kuvassa näkyvät kysymykset ovat vain havainnollistamistarkoituksessa olevia tekstinpätkiä.



Kuva 5. Valmis EtusivuComponent selaimessa.

4.3 Hakusivu

Ohjelmiston toinen sivu, hakusivu, on käyttöliittymältään hyvin samankaltainen kuin etusivukin, mistä syystä tässä aliluvussa ei näytetä hakusivun käyttöliittymäprototyyppiä. Viitteeksi hakusivun käyttöliittymäprototyyppiä voidaan vertailla Etusivun prototyyppiin edellisestä aliluvusta. Kun etusivulla on tarkoitus listata kaikki tietokantapalvelimelta löytyvät kysymykset, on hakusivun olennaisin ero siinä, että se näyttää vain annetun hakusanan perusteella vastaavat kysymykset. Tämä siksi, että käyttäjä toivottavasti hyödyntäisi hakutoimintoa etsiäkseen omaa ongelmaansa koskevia aiempia kysymyksiä. Idea on siinä, että kun käyttäjä löytääkin ongelmaansa ratkaisun jo jonkun toisen henkilön kysymänä, vältytään duplikaattikysymyksiltä ja ylläpitäjän vaivaamiselta. Ideaalitilanteessa opittu tieto löytyy yhdestä ja vain yhdestä paikasta.

Jotta käyttäjä voi lopullisessa sivustossa navigoida etusivulta hakusivulle, täytyy sille määrittää oma Angular-reitti. `HakuComponentin` reitti saa itselleen kaksi verkko-osoiteparametria, `term` ja `page`. Käyttäjän etusivulla hakukenttään syöttämä teksti kaapataan kentästä ja siirretään verkko-osoitteen kautta `HakuComponentin` luokkatiedoston manipuloitavaksi. Lisäksi, koska on mahdollista, että hakusivulla jokin tietty hakutermi johtaa suureen määrään hakutuloksia, annetaan `HakuComponentille` myös `page`-parametri, jotta voidaan luoda edellisessäkin aliluvussa esitelty sivutustoiminnallisuus.

```
  { path: 'search/:term/:page', component: HakuComponent },
```

Hakusivussa törmättiin etusivuakin vaivanneeseen samaan pienimuotoiseen ongelmaan, joka ilmeni, kun tietokantapalvelimelta yritettiin servicen kautta hakea kysymysdataa http-vastauksen rungosta (`body`) ja samaan aikaan kerätä tieto siitä, kuinka monta kysymystä numeromäärällisesti tietokannasta juuri haettiin. Jälkimmäinen informaationpalanen on kätketty palvelimen takaisin lähettämän http-vastauksen `headeriin`. Tyypitysvirheen vuoksi näitä kahta asiaa ei yhdellä funktiolla saada haettua, joten tietovirta jaetaan kahteen erilliseen funktioon, joita kutsutaan komponentissa peräkkäisesti.

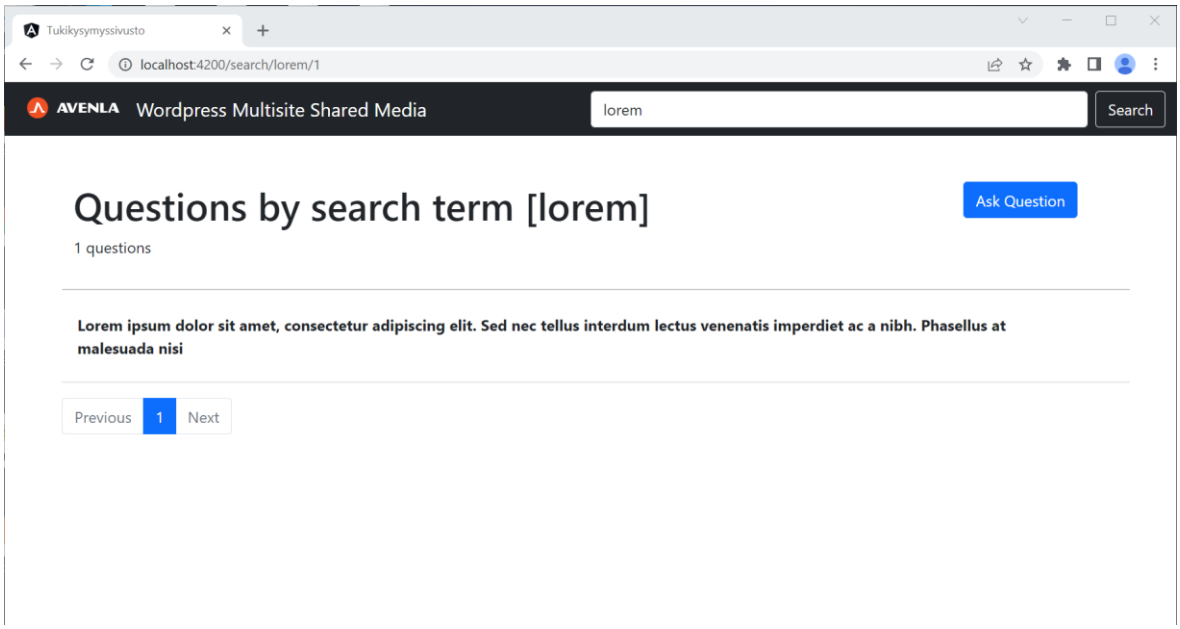
```
  /** Haetaan kaikki hakuehtoja vastaavat kysymykset palvelimelta */
  haeKysymyksetHakusanalla(hakuTermi: string, sivu: number): Observable<Ky-
symys[]> {
    return this.http.get<Kysymys[]>(`${this.ky-
symyksetUrl}/search?search=${hakuTermi}&per_page=10&page=${sivu}`)
  }

  /** Haetaan hakuehtoja vastaavan http-vastauksen headerit observe-lisä-
optiolla */
  haeHeaderitHakusanalla(hakuTermi: string): Observable<HttpResponse<Ky-
symys[]>> {
    return this.http.get<Kysymys[]>(`${this.ky-
symyksetUrl}/search?search=${hakuTermi}&per_page=10`,
    { observe: 'response' });
  }
```

Hakusivu voi pintapuolisesti tarkastellen vaikuttaa hyvin samankaltaiselta kuin etusivu, mutta pinnan alla se on itse asiassa olennaisesti erilainen. Sillä aikaa, kun etusivu ottaa yhteyttä Wordpressin REST-rajapintaan ja kerää kysymyksiin liittyviä tietoja, kuten luontipäivämäärän ja tekstisisällön, ei rajapinta näitä tietoja paljasta kutsuttaessa hakutuloksiin liittyvää päätepistettä. Tästä syystä hakusivu on lähdekoodin osalta selkeästi suoraviivaisempi kuin etusivu ja komponentin luokkatiedostossa yksinkertainen `subscribe()`-metodi kutsuttaessa servicen funktioita riittää.

Käyttäjän päätyminen hakusivulle on seurausta `AppComponentin` sisältämästä `navigate()`-metodista. Mikäli kyseinen metodi laukaistaan käyttäjän ollessa jollakin toisella sivulla kuin hakusivulla, metodi toimii normaalisti ja ohjaa käyttäjän haluttuun osoitteeseen. Ongelmia alkaa ilmetä siinä vaiheessa, kun käyttäjä on jo navigoinut itsensä hakusivulle ja sieltä käsin tekee hakuja. Siinä tilanteessa `HakuComponent` ei osaakaan reagoida `.navigate()`-metodin muuttaneeseen URL-parametriin, joka pitää sisällään hakutermin. Ongelma korjataan lisäämällä `HakuComponentin` konstruktoriin koodinpätkä, joka pakottaa Angular-reitin uudelleenlatauksen, mikäli URL-parametrit muuttuvat:

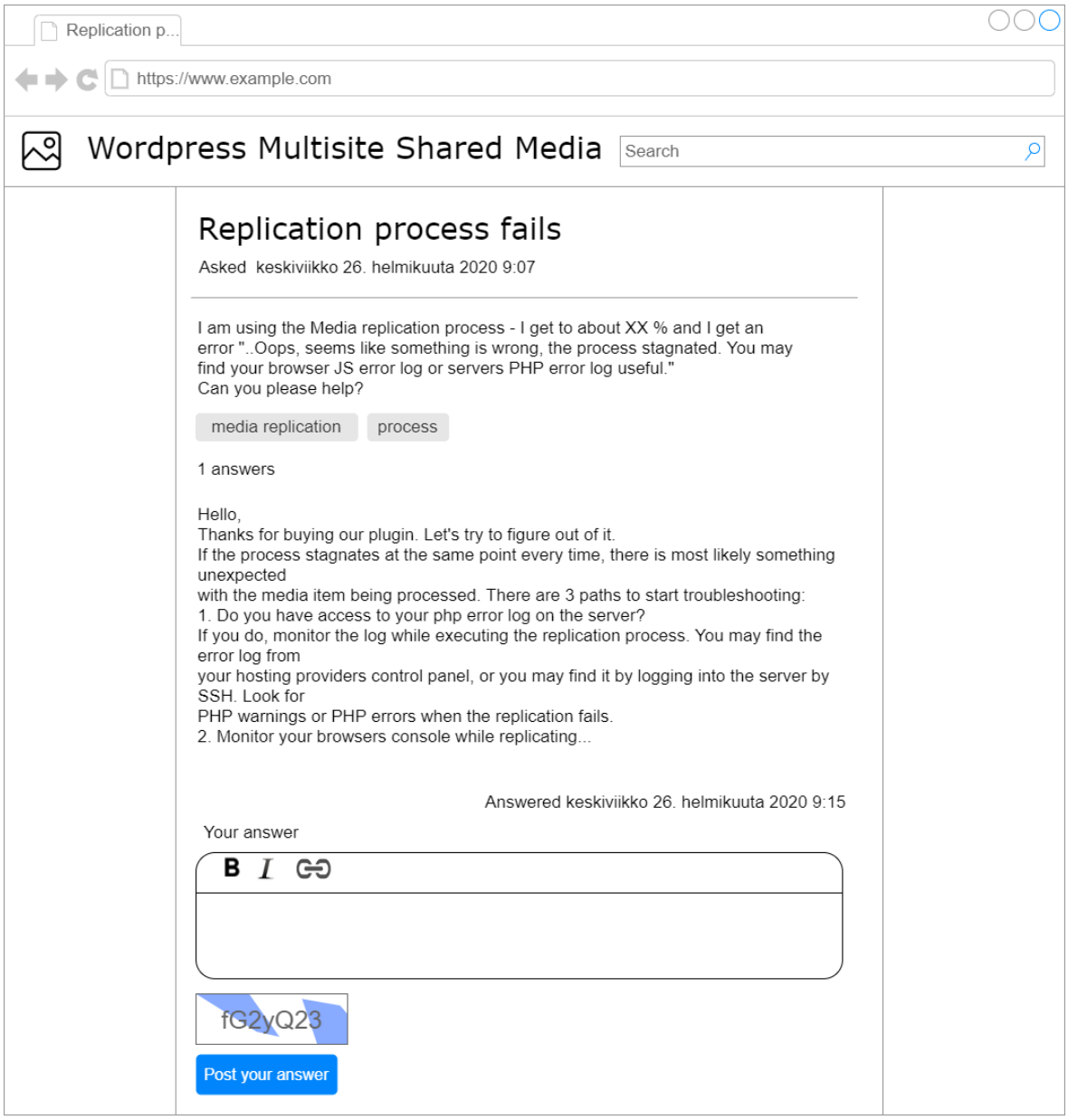
```
constructor(private reititin: Router) {
  // pakota reitin uudelleenlataus kun parametrit muuttuvat
  this.reititin.routeReuseStrategy.shouldReuseRoute = () => false;
}
```



Kuva 6. Valmis Hakusivu selaimessa.

4.4 Valittu kysymys ja sen kommentit -sivu

Ohjelmiston kolmannella sivulla (kuva 7) on tarkoitus näyttää etusivun kautta valitun yksittäisen kysymyksen tarkemmat tiedot. Sivulla tulee näkymään kysymyksen otsikko, luontipäivämäärä, sisältö kokonaisuudessaan ja kysymykset tunnisteet. Tämän alapuolella näkyy vastaukset ja niiden luontipäivämäärät. Alimmaisena sivulla on rich text editor -tekstikenttä vastauksen lisäämiselle sekä käyttäjän tunnistamista varten CAPTCHA-kenttä. Sivun viimeinen elementti on nappi, jota painamalla oma vastaus lisätään muiden sekaan, kunhan käyttäjä läpäisee CAPTCHA-testin. Rich text editor -tekstikenttä toteutetaan TinyMCE-sovelluksella integroimalla se osaksi käyttöliittymää.



Kuva 7. Valittu kysymys ja sen kommentit -sivun käyttöliittymäprototyyppi.

Valittu kysymys -sivun http-kutsut ja haeKysymys()-funktio

Wordpressin dokumentaation (2022d) mukaan yksittäisen kysymyksen tiedot voidaan hakea rajapinnasta, kun sille lähetetään `GET`-muotoinen pyyntö osoitteeseen `/wp/v2/posts/<id>`. `EtusivuComponentin` templateen on tässä vaiheessa lisätty `routerLinkit` `/detail/<id>`-arvoilla ohjaamaan käyttäjä `YksityiskohtaComponentiin`. `AppRoutingModulessa` myös `YksityiskohtaComponentille` täytyy antaa osoitepolku ja `id`-parametri. Seuraava koodinpätkä osoittaa, kuinka kun käyttäjä klikkaa etusivun linkkejä, hänet voidaan tuoda osoitteeseen http://palvelimen_osoite/detail/<id>, missä näytetään olennaisia tietoja.

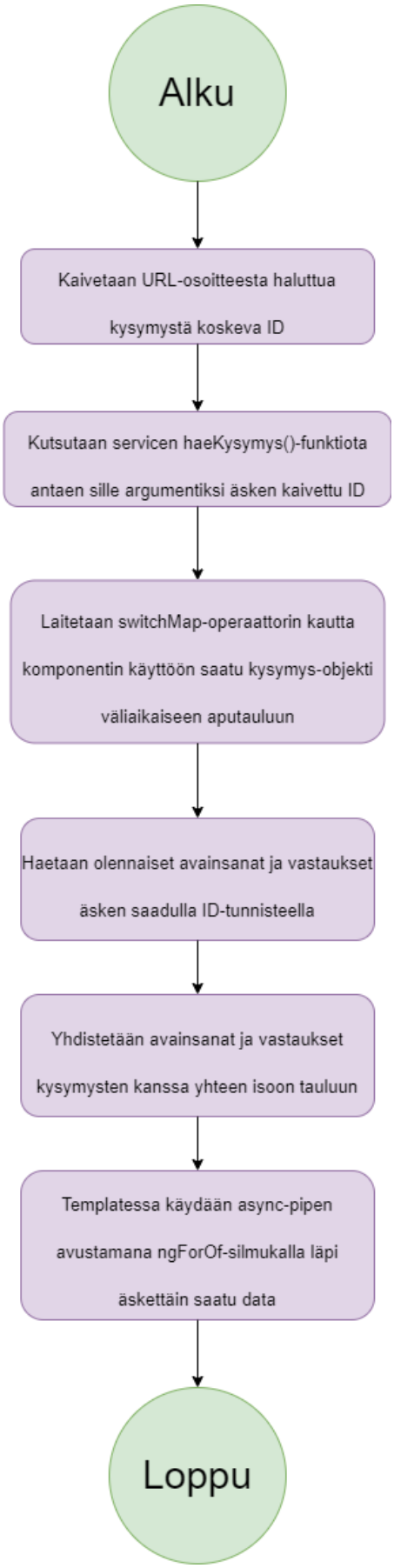
```
{ path: 'detail/:id', component: YksityiskohtaComponent },
```

`KysymyksetServicen` funktio muistuttaa hyvin paljon `EtusivuComponentin` kaikki kysymykset hakevaa koodia. Ero on siinä, että tämä funktio ottaa argumentikseen numeromuotoisen `id`-muuttujan ja kutsuu rajapintaa osoitteessa `/wp/v2/posts/<id>`, kuten aiemmin mainittiin. Palautusarvo on tässä tapauksessa `Kysymys`-muotoinen objekti, sillä rajapinta ei palautakaan taulua, kuten etusivulla.

```
  /** Haetaan tietty kysymys annetun ID:n perusteella palvelimelta */
  haeKysymys(id: number): Observable<Kysymys> {
    return this.http.get<Kysymys>(`${this.kysymyksetUrl}/posts/${id}}`)
  }
```

Tällä sivulla on myös tarkoitus näyttää etusivulla valitun kysymyksen kaikki vastaukset sisältäen niiden tekstirungon ja luontipäivämään. Wordpressin (2022e) mukaan vastaukset saadaan kaivettua rajapinnasta lähettämällä sille `GET`-muotoinen pyyntö osoitteeseen `/wp/v2/comments`, jonka mukana toimitetaan `post`-argumentti, mikä rajaa tulosjoukon koskemaan vain niitä vastauksia, jotka liittyvät annetun kysymyksen id:hen. Servicessä tämä näyttää seuraavanlaiselta:

```
  /** Haetaan halutun kysymysId-numeron perusteella tietyt vastaukset */
  haeYksityiskohdanVastaukset(kysymysId: number): Observable<any> {
    return this.http.get<any>(`${this.kysymyksetUrl}/comments?post=${kysymy-
sId}`);
  }
```



Kuvio 8. Valittu kysymys -sivun datan hakuprosessi.

Vuokaavio osoittaa kuinka data haetaan valittu kysymys ja sen tiedot -sivua varten. Ensin selaimen URL-osoitteesta kaivetaan valittua kysymystä koskeva ID-numero, jolla kutsutaan servicen valittua kysymystä hakevaa funktiota. Servicen funktio palauttaa JavaScript-objektin. Jotta dataa olisi helpompi käsitellä, komponentissa `switchMap`-operaattorilla haltuun saatu objekti sisällytetään väliaikaiseen tauluun, jonka muun muassa RxJS:n `of`-operaattori saa argumentikseen. Sitten kutsutaan avainsanat hakevaa funktiota äsken saadulla ID:llä, minkä jälkeen on vuorossa kutsua vastaukset hakevaa funktiota jälleen äsken saadulla ID-tunnisteella. Lopuksi koodi yhdistää avainsanat ja vastaukset kysymysten kanssa yhteen isoon kokonaistauluun. Templatessa voidaan `async`-pipen avustamana `ngForOf`-silmukalla käydä läpi äskettäin saatu kokonaisdata ja tulostaa se näinollen käyttäjän katseltavaksi.

Lomake vastauksen lisäämistä varten

Valittu kysymys ja sen tiedot -sivulla on tarkoitus antaa loppukäyttäjälle mahdollisuus kirjoittaa vastauksia kysymyksiin eli tarvitaan tekstikenttä ja tallennuspainike. Template näyttää tässä vaiheessa seuraavanlaiselta:

```
<label for="vastaus"><h4>Your Answer</h4></label>
    <textarea class="form-control" id="vastaus" rows="3" name="vastaus" re-
quired minlength="30" [(ngModel)]="vastaus"
        #kentta="ngModel"></textarea>

    <div *ngIf="kentta.invalid && (kentta.dirty || kentta.touched)"
class="omaAlert">
        <div *ngIf="kentta.errors?.required">Body is missing.</div>
        <div *ngIf="kentta.errors?.minlength">
            Body must be at least 30 characters; you entered {{
vastaus.length }}.
        </div>
    </div>

    <br>

    <button [disabled]="kentta.invalid" (click)="tallen-
naVastaus(kentta.value)" class="btn btn-primary" type="submit">Post Your An-
swer</button>
```

Tämä lyhyt koodinpätkä sisältää paljon tärkeää informaatiota, josta keskeisimpänä on lomakkeen syötteen validointi. Validointi on vakiomuotoinen prosessi missä tahansa laadukkaassa web-sovelluksessa, joka sallii käyttäjän lisätä järjestelmiin dataa lomakkeiden avulla.

Angularin dokumentaatiosivun (2022o) mukaan datan laatua voidaan parantaa validoimalla se, jotta saavutetaan tarkkuus ja täydellisyys. Tarkoituksena on validoida käyttäjän syöte käyttöliittymästä ja näyttää hyödyllisiä validointiviestejä.

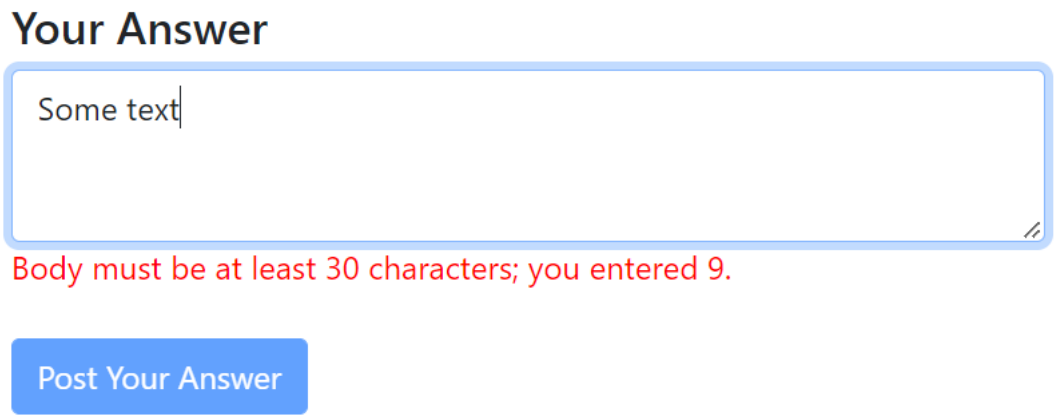
Angularin (2022o) mukaan ottaakseen käyttöön validointi lomakkeissa, lisätään samoja attribuutteja kuin alkuperäisissä HTML-lomakevalidoinneissa. Angular käyttää direktiivejä naittamaan nämä attribuutit kehysalustan validointifunktioihin. Joka kerta kun lomakekentän arvo muuttuu, Angular suorittaa validoinnin ja generoi joko validointivirhelistan, joka johtaa `INVALID`-statukseen, tai null-arvon, joka johtaa `VALID`-statukseen. Sen jälkeen lomakekentän tilaa exportoimalla `ngModel` paikalliseen template-muuttujaan. Edellä esitetty koodinpätkä exportoi `NgModelin` muuttujaan nimeltä `kentta`. Edellä esitetystä koodinpätkästä on huomioitava seuraavat erikoispiirteet:

- `<textarea>`-elementti kantaa HTML-validointiattribuutit `required` ja `minlength`.
- `#kentta=”ngModel”` exportoi `ngModelin` paikalliseen muuttujaan nimeltä `kentta`. `NgModel` heijastaa monia sen sisältämän `FormControl`-instanssin ominaisuuksia, joten tätä voidaan käyttää templatessa tarkistaakseen lomakekentän statusta, kuten esimerkiksi `valid` ja `dirty`.
    - `*ngIf`-direktiivi `<div>`-elementissä paljastaa sisäkkäisiä viestejä näyttäviä `divejä`, mutta vain jos `kentta` on virheellinen ja lomakekentän tila on joko `dirty` tai `touched`.
    - Jokainen sisäkkäinen `<div>` voi näyttää mukautetun virheilmoituksen kullekin mahdolliselle virheelle. Koodinpätkässä on viestejä `required`- ja `minlength`-tilanteita varten.

Angularin dokumentaatiosivujen (2022o) mukaan estääkseen validoijan näyttämästä virheilmoituksia ennen kuin käyttäjällä on ollut mahdollisuus tehdä muutoksia lomakekenttään, pitää tehdä tarkistuksia joko `dirty`- tai `touched`-tiloista lomakekentässä.

- Kun käyttäjä tekee muutoksia lomakekentän sisältöön, kenttä merkitään likaiseksi (`dirty`)
- Kun käyttäjä poistaa fokuksen lomakentästä siirtymällä muuhun käyttöliittymän osaan, lomakekenttä merkitään kosketetuksi (`touched`)

Nyt valittu kysymys -sivun lomakkeessa on toimiva validointi (kuva 9), joka tarkastaa tekstikentän siltä varalta, että käyttäjä joko ei syöttänyt kenttään yhtäkään merkkiä tai että kentässä on alle 30 merkkiä. Kuten aiempi koodinpätkä osoittaa, lomakkeen painonappi poistetaan aktiivisesta käytöstä, mikäli havaittiin yksikin virhe käyttäjän toiminnassa.



Kuva 9. Virheilmoitus yksityiskohtasivun lomakkeessa.

Anonyymien kommenttien lähettäminen sivustolla

Toimeksiantajan pyynnöstä tukikysymyssivustolla ei ole sisäänkirjautumista käyttäjille, joten kaikki sivustolle lähetetty sisältö toimitetaan anonyymisti. Wordpressin REST-rajapinta oletusarvoisesti ja kenties hyvästäkin syystä ei salli kommenttien lähettämistä anonyymisti. Ongelma on kuitenkin helppo kiertää luomalla uusi liitännäinen, joka sallii anonyymit kommentit myös sellaisilta Wordpress-käyttäjiltä, jotka ovat sisäänkirjautumattomia. Seuraava Wordpress-filterin muodostava koodinpätkä uudessa liitännäisessä hoitaa asian:

```
function filter_rest_allow_anonymous_comments() {

    return true;

}

add_filter('rest_allow_anonymous_comments','filter_rest_allow_anony-
mous_comments');
```

Jotta kaikki virheet huomattaisiin jo ennen koodin ajamista, vastaukset tyypitetään interface-tiedoston avulla seuraavanlaisesti:

```
export interface Vastaus {
    content: string;
    date: string;
    post: number;
    author_name: string;
```

```
    author_email: string;
}
```

Sovelluksen näkökulmasta mielenkiintoisimmat vastauksen tiedonmuruset ovat `content` ja `date`, jotka näytetään käyttäjälle selainsivulla. Lisäksi `post`-avain on tärkeä tieto, sillä sen avulla vastaus yhdistetään haluttuun kysymykseen. Wordpressin REST-rajapinta asettaa anonyymeille kommenteille vaatimukseksi työkuorman mukana toimittaa `author_name` ja `author_email`-kentät. Ne tarkoittavat viestin kirjoittaneen Wordpress-käyttäjän henkilötietoja.

KysymysServiceen annetaan `lisaaVastaus()`-funktio, joka ottaa argumentikseen `Vastaus`-tyyppisen objektin, ja siinä kutsutaan `HttpClient.post()`-metodia. `POST` on siis yksi monista niin kutsutuista http-verbeistä, joka käytännössä tarkoittaa uuden tietueen luomista. Tämä `post()`-metodi tarvitsee kolme parametria: kohteen URL-osoite, rajapinnalle lähetettävä data-objekti, sekä datan muodon kertovat headeri-optio, joka käytännössä määrittää lähetettävän datan olevan JSON:ia, jota Wordpressin REST-rajapinta osaa käsitellä.

```
  httpValinnat = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
  };

  /** Lisää uusi vastaus kysymykseen */
  lisaaVastaus(vastaus: Vastaus): Observable<Vastaus> {
    return this.http.post<Vastaus>(`${this.kysymyksetUrl}/comments`, vas-
taus, this.httpValinnat);
  }
```

Yksityiskohta-komponenttiin lisätään `tallennaVastaus()`-funktio, joka ottaa argumentikseen templatesta lähetettävän lomakekentän merkkijonon. Kun funktiota lähetään ajamaan, se ensin tyhjentää lomakekentän. Tämän jälkeen se luo `Vastaus`-tietotyyppiä mukailevan dataobjektin. Siinä käytännössä määritetään kaikki uuden vastauksen olennaisimmat tiedot ja kovakoodataan Wordpress-käyttäjä `rootin` nimi ja sähköpostiosoite.

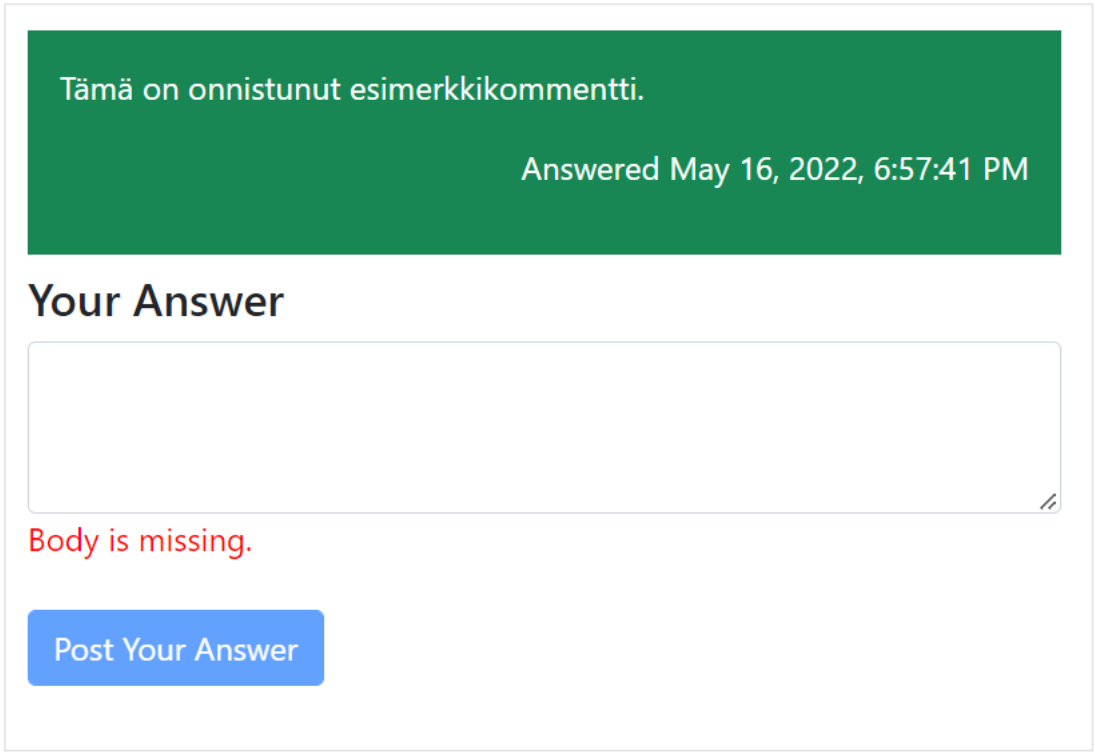
```
  tallennaVastaus(kentanArvo: string) {
    this.vastaus = "";

    let vastausData = {
      "content": kentanArvo,
      "date": new Date().toJSON(),
      "post": this.id,
      "author_name": "root",
      "author_email": "aleksander.valiaho7@gmail.com"
    }
```

```
    if (!kentanArvo) { return; }
    this.kysymysService.lisaaVastaus(vastausData as Vastaus)
      .subscribe(vastaus => {
        this.uusiKommentti = vastaus;
      })
  }
```

`Subscribe()`-metodi palauttaa onnistuneen vastauksen lisäämisen jälkeen komponentin käytettäväksi objektin, joka sisältää nyt luodun vastauksen. Templatessa käyttäjän iloksi lisätään värikäs laatikko (kuva 10) Bootstrapin CSS-määreillä, joka osoittaa, että vastaus on onnistuneesti lisätty tietokantaan.

```
    <div class="p-3 mb-2 bg-success text-white" *ngIf="uusiKommentti !=
null">
        <p [innerHTML]="uusiKommentti?.content?.rendered"></p>
        <p class="text-end">Answered {{ uusiKommentti?.date | date:'medium'
}}</p>
    </div>
```



Kuva 10. Näkymä selaimessa, kun uuden vastauksen lisääminen sivulle onnistui.

Käyttäjän todentaminen lomakkeessa CAPTCHA-varmennuksella

Tässä vaiheessa `YksityiskohtaComponentin` lomake osaa jo estää tietokantaan joutumasta epäsopivaa dataa, mutta nyt aivan kuka tahansa voi kirjoittaa kommentteja, jopa roskapostibotit. Lomakkeelle täytyy siis luoda jonkinlainen estemuuri estämään skriptien

kirjoittamasta roskaa sivustolle. Tätä varten on kehitetty CAPTCHA tai Completely Automated Public Turing test to tell Computers and Humans Apart, joka toimeksiantajan pyynnöstä lisätään tukikysymyssivuston niille sivuille, joilla on dataa tietokantaan lähettävä lomake. CAPTCHA on testi, joka on suunniteltu määrittämään, että käyttäjä on ihminen, eikä botti. CAPTCHA- ja reCAPTCHA-testejä tavataan useilla sivuilla internetissä. Tällaiset testit ovat yksi tapa kontrolloida bottien käytöstä, vaikka näillä lähestymistavoilla on haittapuolensakin. Vaikka CAPTCHA-todennukset ovat suunniteltuja estämään automatisoidut botit, ovat ne itse automatisoituja. Ne ovat ohjelmoitu ilmestymään tietyissä sivustojen osissa ja ne automaattisesti hyväksyvät tai hylkäävät testin suorittajan. (Cloudflare 2022.)

Klassiset CAPTCHA-todennukset, joita jotkin sivut käyttävät yhä, pyytävät käyttäjää tunnistamaan merkkijonoja. Kirjaimia on vääristetty, jotta botit eivät todennäköisesti pysty niitä tunnistamaan. Läpäistäkseen testin, käyttäjän on tulkittava vääristettyä tekstiä, syötettävä oikeat kirjaimet tekstikenttään ja lähetettävä lomake käsiteltäväksi. Mikäli kirjaimet eivät täsmää, käyttäjää pyydetään kokeilemaan uudestaan. Tällaisia testejä näkee kirjautumislomakkeissa, tilirekisteröitymislomakkeissa, äänestyskyselyissä ja verkkokaupan kassalla. Idea on siinä, että tietokoneohjelma, joita botit ovat, on kykenemätön tulkitsemaan vääristettyä tekstiä sillä välin, kun ihminen, joka on tottunut lukemaan kirjaimia eri asiayhteyksissä, eri fonteilla ja eri käsialoilla, pystyy testin läpäisemään. Parasta, mihin jotkin botit pystyvät, on syöttää satunnaisia kirjaimia, jolloin ne luultavasti epäonnistuvat testin läpäisyssä, ja siten niitä estetään käyttämästä verkkopalveluja sillä välin, kun ihminen voi käyttää internetsivua normaalisti. Kehittyneemmät botit osaavat hyödyntää koneoppimista tunnistamaan vääristettyjä kirjaimia, joten tällaiset CAPTCHA-todennukset ovat korvautumassa monimutkaisemmilla testeillä. Googlen reCAPTCHA on kehittänyt useita testejä erottamaan botit ihmisistä. (Cloudflare 2022.)

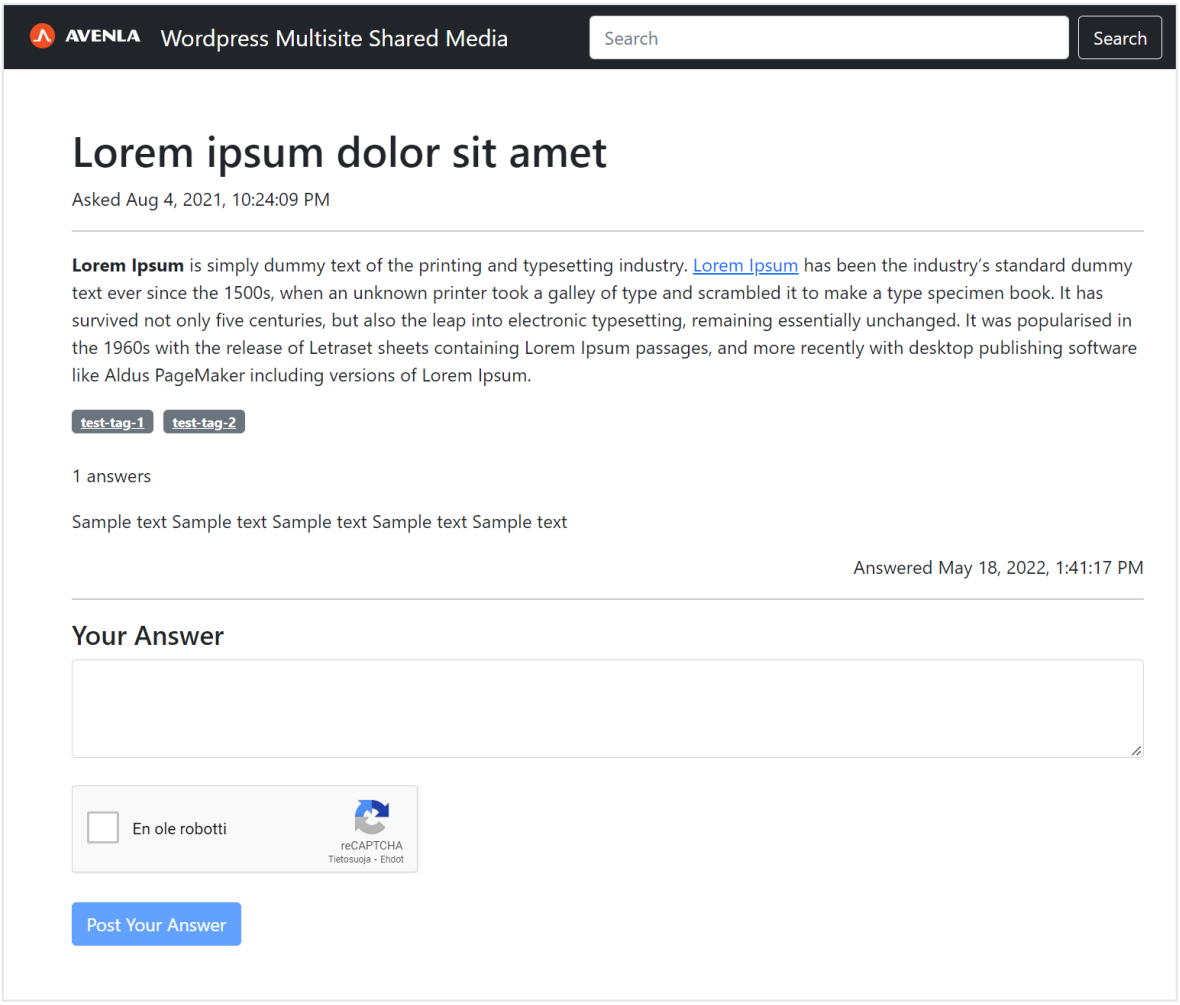
reCAPTCHA on ilmainen palvelu, jota Google tarjoaa vaihtoehtona perinteisille CAPTCHA-todennuksille. reCAPTCHA-teknologian kehitti tutkijat Carnegie Mellonin yliopistolla, minkä jälkeen Google hankki sen vuonna 2009. reCAPTCHA on kehittyneempi kuin tyypilliset CAPTCHA-testit. reCAPTCHA käyttää lähteenään tekstiä oikean elämän kuvista: katuosoitteista, tekstiä tulostetuista kirjoista ja vanhoista sanomalehdistä. Ajan kuluessa Google on laajentanut reCAPTCHA-todennuksien toiminnallisuutta, jotta niiden ei enää tarvitsisi turvautua sumean tai vääristetyn tekstin tunnistamiseen. Yksi uudemmista testeistä on valintaruututesti. (Cloudflare 2022.)

Jotkin reCAPTCHA-testit yksinkertaisesti kehottavat käyttäjää napsauttamaan valintaruutua, jonka vieressä on teksti ”En ole robotti”. Kuitenkin testin idea ei ole valintaruudun painaminen, vaan tapahtumat, jotka johtavat tuon valintaruudun napsauttamiseen. Tämä

reCAPTCHA-testi ottaa huomioon käyttäjän kursorin liikkeet, kun se lähestyy valintaruutua. Jopa kaikista suorin ihmisen suorittama kursorin liike sisältää ainakin vähän satunnaisuutta ja heilahtelua mikroskooppisella tasolla. Se on siis pientä epätietoista liikettä, jota botit eivät kykene matkimaan. Jos kursorin liike sisältää tätä satunnaisuutta, testi päättää, että käyttäjä todennäköisesti on ihminen. Tämä reCAPTCHA saattaa myös ottaa huomioon käyttäjän laitteelle tallennetut evästeet ja selaushistorian, jotta testi voi arvioida, onko käyttäjä botti vai ei. Jos testi on tämänkin jälkeen kykenemätön määrittämään käyttäjän ihmisyyden, se saattaa tuottaa toisen kuvantunnistustestin käyttäjän ratkottavaksi. Kuitenkin suurimman osan ajasta käyttäjän kursorin liikkeet, evästeet ja selaushistoria ovat riittävän vakuuttavia. (Cloudflare 2022.)

reCAPTCHA:n liittäminen Angular-projektiin on suoraviivainen prosessi ja siihen löytyy tiivis opas osoitteesta https://www.npmjs.com/package/ng-recaptcha tai videomuodossa osoitteesta https://youtu.be/270OPZhYxZM. Videomuotoinen opas käy läpi myös Googlen sivustoavaimen hankkimisvaiheen. On huomionarvoista, että komennon `npm i ng-recaptcha –save` oletusarvoisesti asentama versio 9.0.0 ei jostakin syystä toiminut Angular-projektin versio 11.2.14 kanssa. Sovellus kaatui erikoiseen virheilmoitukseen `Namespace has no exported member 'ɵɵFactoryDeclaration'`. `ng-recaptcha`-paketti täytyi manuaalisesti alentaa versioon 8.0.1 komennolla `npm install ng-recaptcha@8.0.1`, minkä jälkeen koodi alkoi toimia ongelmitta.

Yksityiskohtasivu on nyt valmis ja se osaa näyttää valitun kysymyksen tiedot sekä ohjauskentät uuden vastauksen luomiselle (kuva 11).



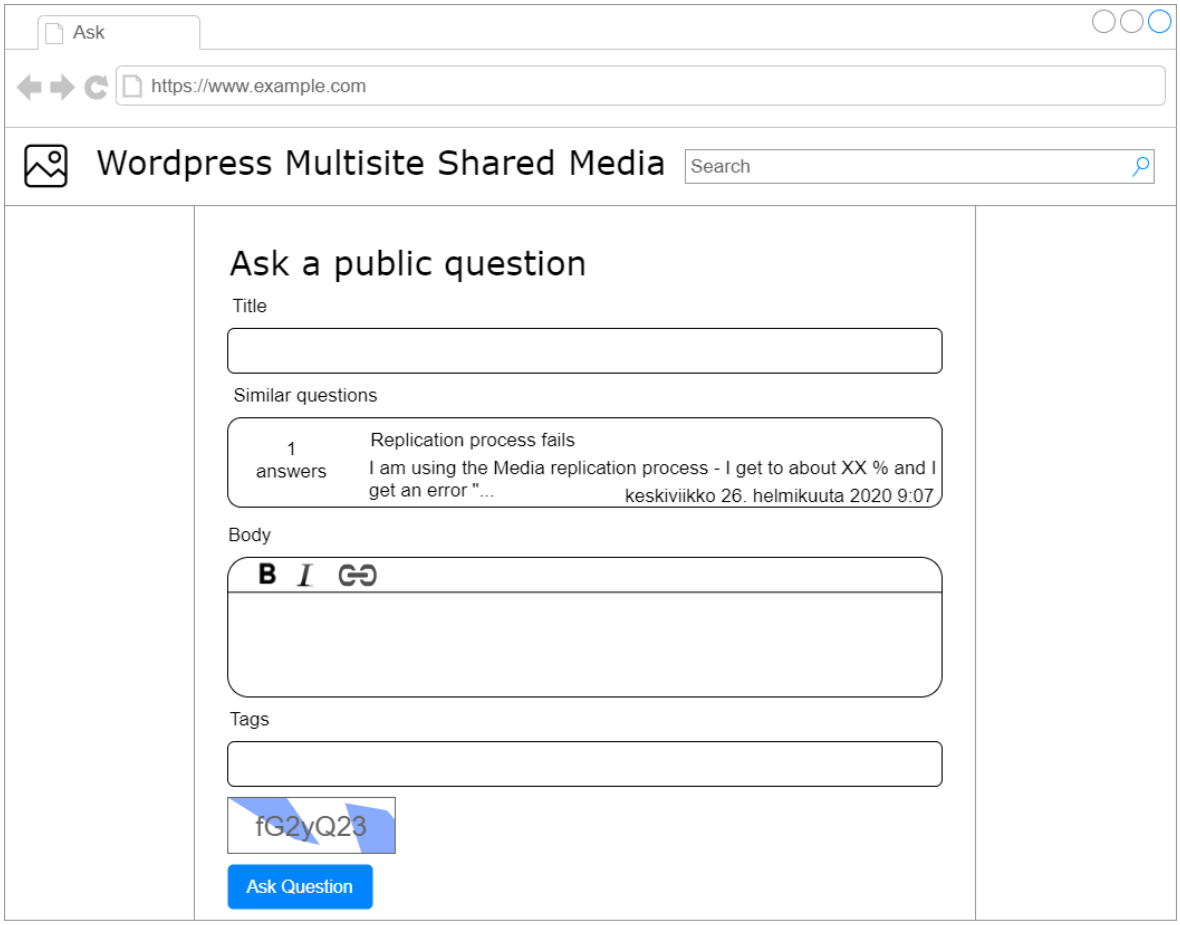
Kuva 11. Valmis YksityiskohtaComponent selaimessa.

4.5 Kysy uusi kysymys -sivu

Sovelluksen viimeinen sivu (kuva 12) tarjoaa toiminnallisuudet uuden kysymyksen kysymiselle. Sivulta lähetetään Wordpressin REST-rajapinnan kautta tietokantaan POST-metodilla olennaiset tiedot ja kysymys tulee näkyviin etusivulle muiden kysymysten joukkoon.

Otsikkopalkin jälkeen ensimmäisenä sivulla on lomakekentät kysymyksen otsikolle ja varsinaiselle sisällölle on oma rich text editor -tekstikenttä, joka sekin toteutetaan TinyMCE-ohjelmalla. Erikoisuutena sivulla on Stack Overflow -sivustolta tuttu lisäominaisuus, jossa kun käyttäjä alkaa kirjoittaa uudelle kysymykselle otsikkoa käynnistetään haku Wordpressin tietokantaan ja etsitään samankaltaisia jo olemassa olevia kysymyksiä. Tämän idea on auttaa käyttäjä löytämään jo olemassa olevia kysymyksiä, mikäli uuden kysymyksen ja vanhojen kysymysten otsikoista löydetään samoja piirteitä. Idea on se, että vältyttäisiin mahdollisilta kopioilta kysyttäessä uutta kysymystä. Kuten edelliselläkin sivulla, uusi kysymys -sivulla käyttäjä tunnistetaan ihmiseksi CAPTCHA-varmenteella, jotta mahdolliset

roskapostibotit eivät voisi tuottaa sivustolle roskasisältöä. Sivulla alimmaisena on Ask Question -nappi, jota painamalla kysymyksen tiedot lähetetään tietokantaan.



Kuva 12. Kysy uusi kysymys -sivun käyttöliittymäprototyyppi.

Luonnollisesti koska nyt luodaan uusi komponentti, täytyy se reitittää. Tässä kohtaa uusi kysymys -sivun osalta mukaillaan stackoverflow.comin rakennetta ja `UusiKysymysComponent` löytyy osoitteesta `/questions/ask`. Kokonaisuudessaan valmiin `AppRouting`-moduulin reititystaulu näyttää seuraavan laiselta:

```
const reitit: Routes = [
  { path: 'questions/:page/:orderby/:order', component: EtusivuComponent },
  { path: 'search/:term/:page', component: HakuComponent },
  { path: 'detail/:id', component: YksityiskohtaComponent },
  { path: 'questions/ask', component: UusiKysymysComponent },
  { path: '', redirectTo: '/questions/1/date/desc', pathMatch: 'full' }
];
```

Sivun lomakeohjaimet

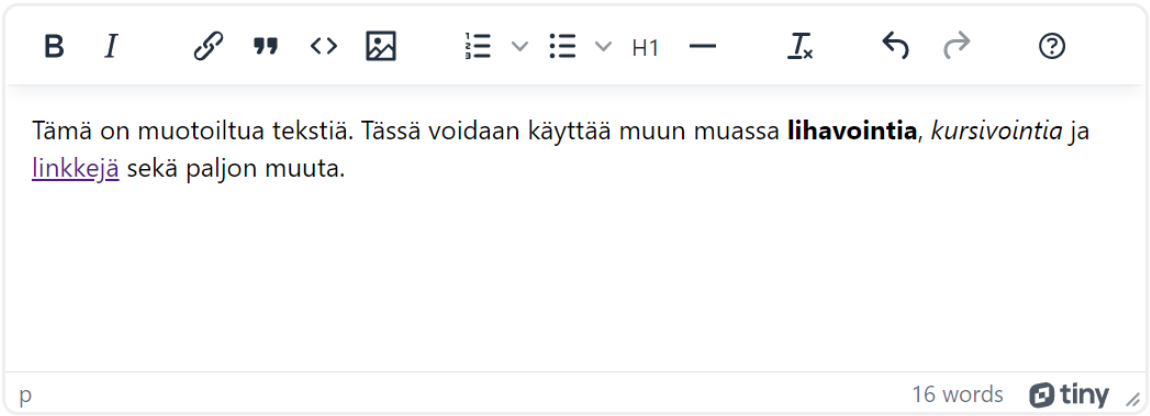
Samaan tapaan kuin edellisessäkin luvussa, uusi kysymys -sivulla on lomake, joka täytyy validoida otsikon osalta ja CAPTCHA:n osalta. Avainsanakenttä on vapaamuotoisempi: käyttäjä voi itse päättää lisätäänkö uuden kysymyksen mukaan avainsanoja, ja jos lisätään, sallitaan enintään kolme kappaletta niitä. Sivun lopussa oleva sininen painonappi aktivoituu, kunhan käyttäjä on täyttänyt lomakkeen vähimmäisvaatimukset eli ainakin 15 merkkiä pitkän otsikon, ei-tyhjän muotoillun runkotekstiosuuden ja En ole robotti -varmennuksen.

`UusiKysymysComponentin` templatessa TinyMCE-kentän lisääminen muotoiltua tekstiä varten on melko yksinkertainen prosessi ja siihen löytyy ohjelmiston kehittäjän opas osoitteesta: https://www.tiny.cloud/docs/integrations/angular/. On huomionarvoista, että Angular 11 -sovelluksen ollessa kyseessä, ohje ei suoraan toimi, vaan TinyMCE-versio täytyi pudottaa uusimmasta versiosta 6.0.1 aina versioon 4.2.4:ään. Tämä on helppo tehdä `package.json`-tiedostossa muokkaamalla riviä `"@tinymce/tinymce-angular": "4.2.4"` Muokkauksen käyttöönottaminen on helppoa ajamalla uudestaan `npm install` -komento, joka päivittää työtilan ohjelmistoriippuvuudet halutun kaltaisiksi.

Kuten jo aiemmin mainittiin, TinyMCE valikoitui useista muotoiltua tekstiä tukevista lomakekenttäohjelmistoista siksi, koska sillä on olemassa vahva integraatio Angular-ohjelmistokehyksen kanssa, mikä tekee käyttöönotosta helppoa. Opinnäytetyön artefaktin kanssa pyrittiin mahdollisimman tarkasti jäljittelemään stackoverflow.com-sivustolla olevaa muotoillun tekstin kenttää, jotta käyttäjän olisi helppo muotoilla tekstiä ja käyttäminen olisi sujuvaa ja intuitiivista. Templaten HTML-koodi näyttää seuraavanlaiselta:

```
<editor [(ngModel)]="html" [init]="{
  base_url: '/tinymce',
  suffix: '.min',
  height: 250,
  menubar: false,
  plugins: [
    'link', 'image', 'code', 'help', 'wordcount', 'advlist', 'lists'
  ],
  toolbar:
    'formatselect | bold italic | \
    link blockquote code image | \
    numlist bullist h1 hr | removeformat | undo redo | help'
}"></editor>
```

Tämä koodinpätkä saa aikaan sen, että selainpäässä (kuva 13) näkyviin tulee seuraavan kaltainen tekstikenttä muotoillulle tekstille:



Kuva 13. TinyMCE-tekstikenttä selaimessa.

TinyMCE-kentän sisällön nappaaminen talteen Angularissa on helppoa. Riittää, että `[(ngModeli)]`:lla kentän arvo sidotaan kaksisuuntaisesti templaten ja komponetin luokkatiedoston välillä, jolloin `html`-muuttuja komponentissa pitää aina sisällään ajantasaisen tekstisisällön.

Avainsanojen osalta päädyttiin lähdekoodin kannalta mahdollisimman yksinkertaiseen ratkaisuun. Käyttäjää pyydetään antamaan enintään kolme vapaavalintaista avainsanaa ja erottamaan niiden nimet toisistaan pilkulla. Jää siis oikeastaan käyttäjän vastuulle, että avainsanat on kirjattu oikein lomakkeeseen, sillä taustalla pyörivä lähdekoodi ei tee korjauksia avainsanalistaan, mikäli käyttäjä teki jonkinlaisen virheen siinä. Tallennusnappia painettaessa templatesta komponenttiin saapuu merkkijonomuotoinen pätkä avainsanoja pilkuilla eroteltuna, esimerkiksi `”avainsanan-nimi-1,avainsanan-nimi-2”`.

Seuraavassa vuokaaviossa (kuva 14) osoitetaan, kuinka uusi kysymys -sivun komponentti käsittelee ja tallentaa dataa servicen kautta. Wordpressin REST-rajapinta on tiukka sen suhteen, minkä muotoista JSON-dataa se hyväksyy luotaessa uusia avainsanoja. `POST`-muotoisen palvelinkutsun runko täytyy olla seuraavanlainen:

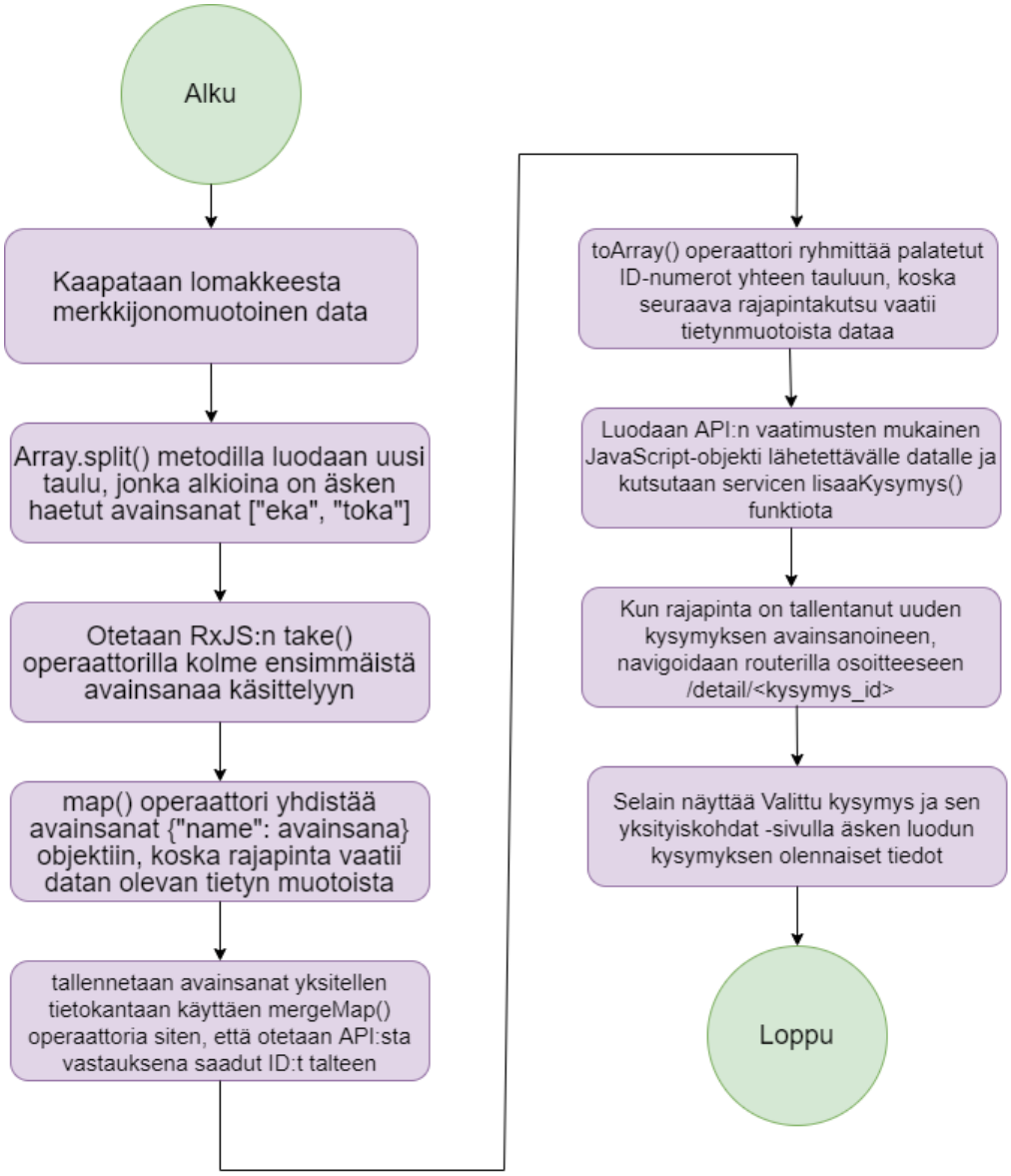
```
{
    "name": "avainsananNimi"
}
```

Lisäksi kuka tahansa ei voi Wordpressin REST-rajapinnalle luonnollisesti lähettää avainsanoja, vaan jokainen `POST`-muotoinen palvelinkutsu täytyy todentaa, mutta siitä lisää hieman myöhemmin.

Vaatimukset uuden kysymyksen luomiselle ovat myös tiukat. Käyttäjä tulee tunnistaa jollakin menetelmällä, minkä lisäksi `POST`-pyynnön rungossa täytyy olla seuraavan mallin mukaista informaatiota:

```
{
    "date": "2022-06-01T16:00:00",
    "status": "publish",
    "title": "Esimerkkiotsikko",
    "content": "Sisältöä havainnollistamisen vuoksi...",
    "tags": ["45", "46", "47"]
}
```

On tärkeää huomata, että luotaessa uutta kysymystä rajapinnan kautta tietokantaan, kysymysdatan mukana ei toimitetakaan avainsanojen ihmisen-luettavia nimiä, vaan vain ID:t. Nuo ID:t saatiin, kun ensin oli `mergeMap()`-operaattorilla tallennettu kukin avainsana palvelimelle, joka vastauksena tallennuspyyntöön palautti niiden tunnistenumerot. Uutta kysymystä luotaessa `POST`-pyynnön rungon `tags`-avain saa arvokseen ID-numeroita sisältävän taulun, ei siis merkkijonoa, kuten edellä on esitetty.



Kuvio 14. Vuokaavio uusi kysymys -sivun datan tallennusprosessista

Käyttäjän todentaminen kehitysympäristössä

Wordpressin (2022d) mukaan sillä välin, kun evästepohjainen todentaminen on ainoa todentamismekanismi, joka Wordpressissa on luonnostaan saatavilla, liitännäisiä voidaan lisätä tukemaan vaihtoehtoisia todentamismenetelmiä, jotka toimivat etäsovellusten kanssa. Esimerkkiliitännäisiä ovat OAuth 1.0a Server, Application Passwords ja JSON Web Tokens. Sitten on myös Basic Authentication -liitännäinen, jota opinnäytetyön artefakti tässä vaiheessa käyttää. Sen GitHub-sivut löytyvät osoitteesta https://github.com/WP-API/Basic-Auth. On huomionarvoista, että tämä liitännäinen asettaa vaatimuksen lähettää käyttäjänimi ja salasana jokaisessa pyynnössä, joten liitännäistä pitäisi käyttää vain ja ainoastaan kehitys- ja testausympäristöissä, ei siis tuotannossa.

Tietoturvallisempi ratkaisu käyttäjän todentamiseen on esimerkiksi äsken mainittu JSON Web Tokens (JWT). Sen kotisivut löytyvät osoitteesta https://jwt.io/ ja sen osoite Wordpress-liitännäisen lataamiselle on https://wordpress.org/plugins/jwt-authentication-for-wp-rest-api/. Internetistä osoitteesta https://blog.angular-university.io/angular-jwt-authentication/ löytyy opas JWT:n käyttöönottamisesta Angular-ympäristössä. Ei voida tarpeeksi painottaa tietoturvallisen käyttöliittymä-palvelin-kommunikoinnin tärkeyttä. Opinnäytetyön artefaktin kohdalla Basic Auth -todentamisen käyttö on tuotantokäytössä tietoturvariski, ja ratkaisuun päädyttiin puhtaasti aikataulullisista syistä: Basic Auth -todennus oli suoraviivainen ja ajankäytöllisesti tehokas ratkaisu, muuten sitä ei voi suositella kuin artefaktin demonstroimistarkoituksiin.

Kun Wordpressin graafisessa selainkäyttöliittymässä on ensin luotu uusi käyttäjä ”admin”, jonka salasana on myös ”admin” ja Basic Auth -liitännäinen on aktivoitu, voidaan serviceen lisätä todentamis-header http-kutsuja varten. Tuotantokäytössä moista käyttäjää ei kannata luonnollisestikaan tehdä tietoturvasyistä.

```
httpValinnatAuth = {
  headers: new HttpHeaders({
    'Authorization': `Basic ${btoa("admin:admin")}`,
    'Content-Type': 'application/json'
  })
}
```

Edellä on esitetty Basic Authilla toteutettu http-headeri, joka lähetetään palvelimelle jokaisen http-kutsun mukana. JavaScriptin `btoa()`-funktio käytännössä Base64-enkoodaa käyttäjän tiedot merkkijonoksi. Sisällön tyypiksi asetetaan `application/json`.

```
/** Lisää uusi kysymys */
lisaaKysymys(uusikysymys: UusiKysymys): Observable<UusiKysymys> {
```

```
    return this.http.post<UusiKysymys>(`${this.kysymyksetUrl}/posts`, uusi-
kysymys,
      this.httpValinnatAuth);
  }

  /** Lisää uusi avainsana */
  lisaaTagi(avainsana: Avainsana): Observable<any> {
    return this.http.post<Avainsana>(`${this.kysymyksetUrl}/tags`, avain-
sana,
      this.httpValinnatAuth)
  }
```

Itse servicen funktiot ovat yksinkertaisia. `HttpClient.post()`-metodin avulla rajapintaa kutsutaan halutussa URL-osoitteessa (`/posts` kysymykselle ja `/tags` avainsanoille) ja mukana toimitetaan varsinainen työkuorma sekä edellä esitetty todennus-header.

Stack Overflow -tyyppinen ehdottava hakujärjestelmä

Viimeinen osa-alue ohjelmiston toteuttamisessa on luoda myös Stack Overflow -sivustolta tuttu reaaliaikainen, jo olemassa olevia ja samankaltaisia muita kysymyksiä linkkeineen ehdotta hakujärjestelmä uusi kysymys -sivulle. Angularin dokumentaatiosivuilta osoitteesta https://angular.io/tutorial/toh-pt6#search-by-name löytyy kattava opas tällaisen toiminnallisuuden ohjelmoimiseen. Tämä tekstiosio käy läpi hakujärjestelmän toimintalogiikan yleisellä tasolla.

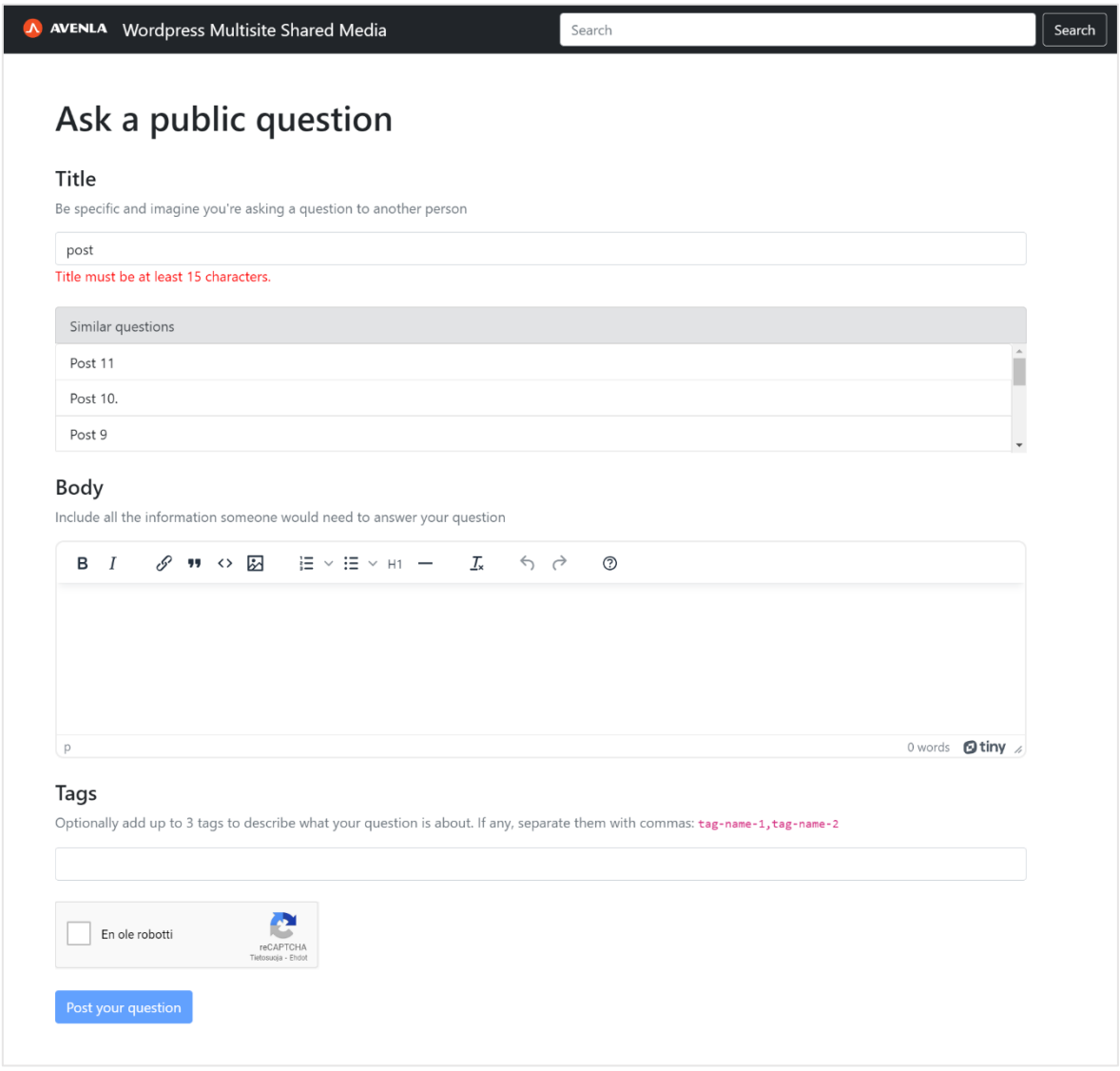
Pohjimmiltaan idea on ketjuttaa `Observable`-operaattoreita yhteen, jotta minimoidaan samankaltaisten http-kutsujen määrä ja verkon kaistainkäyttö pysyisi kohtuullisena. Kun käyttäjä kirjoittaa ensimmäiseen kenttään otsikon, lähetetään toistuvia kutsuja rajapinnalle hakeakseen kysymyksiä, jotka vastaavat annettua nimeä. Tavoite on lähettää vain niin monta kutsua kuin tarvitaan. Joka kerta kun käyttäjä kirjoittaa kenttään, sen `(input)`-tapahtumakytky kutsuu komponentin `hae()`-funktiota kentän merkkijonoarvolla, ”hakutermillä”. (Angular 2022p.)

Uuden hakutermin välittäminen suoraan servicelle joka ikisen käyttäjän näppäinpainalluksen jälkeen loisi kohtuuttoman määrän http-kutsuja, kuormittaen palvelimen resursseja ja kuluttaen rajoitettujen internetliittymien datakattoja. Tämän sijaan `hakuTermit`-Observable vedetään sellaisen RxJS-operaattoriketjun läpi, joka vähentää kutsuja serviceen. `debounceTime(300)`-operaattori odottaa, kunnes näppäinpainallukset taukoavat 300 millisekunnin ajaksi ennen kuin prosessissa edistytään. http-kutsuja ei koskaan tehdä useammin kuin 300 millisekunnin aikavälein. `distinctUntilChanged()`-operaattori varmistaa, että pyyntö palvelimelle lähetetään vain, jos hakutermi on muuttunut. Lopuksi `switchMap()`-operaattori kutsuu serviceä jokaiselle hakutermille, joka läpäisi kahden edellä mainitun operaattorin. Se

peruuttaa ja hylkää edeltävät haku-`Observablet`, palauttaen vain viimeisimmän lähetetyn `Observablen`. (Angular 2022p.)

On tärkeää huomata, että `switchMap()`-operaattorilla jokainen kelvollinen näppäinpainallustapahtuma laukaisee `HttpClient.get()`-metodikutsun. Vaikka käytössä on 300 millisekunnin tauko kutsujen välillä, on mahdollista, että samaan aikaan lennossa on useita http-kutsuja eivätkä ne välttämättä palaa järjestyksessä. `switchMap()` säilyttää alkuperäisen järjestyksen samaan aikaan, kun se palauttaa `Observablen` vain viimeisimmästä http-kutsusta. Vastaukset aiemmista kutsuista peruutetaan ja heitetään roskikseen. (Angular 2022p.)

Uusi kysymys -sivu on nyt valmistunut (kuva 15) ja näinollen myös opinnäytetyön ohjelmistotuotanto-osuus on tullut päätökseen.



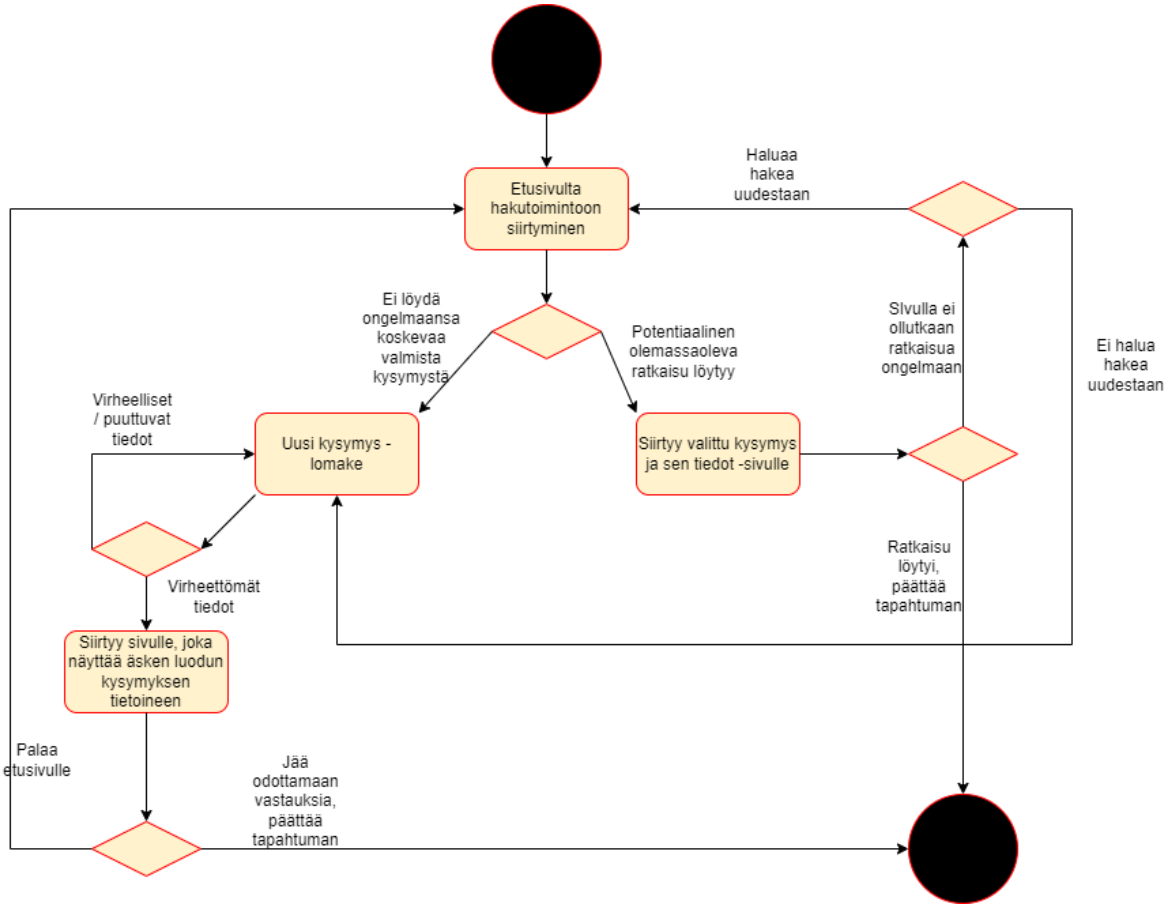
Kuva 15. Valmis uusi kysymys -sivu selaimessa.

Tarvittaessa tukikysymyssivuston kaikki Angular-lähdekoodit löytyvät osoitteesta https://github.com/avaliaho/tukikysymyssivusto.

5 Käyttöliittymän rakenne

Tässä luvussa esitetään kaavioin koko sovelluksen laajuinen käyttöliittymä niin Avenla Oy:n asiakkaan näkökulmasta kuin ylläpitäjän/omistajankin eli Avenla Oy:n web-kehittäjän näkökulmasta. Kaavioiden tarkoitus on selventää sovelluksen toimintalogiikkaa yleisellä tasolla paremmin kuin syvällinen katsaus lähdekoodiin, jota aiemmat luvut käsittelivät. Asiakkaan käyttöliittymän (kuvio 16) kohdalla käy ilmi, että hakutoiminto on keskeisessä roolissa sovelluksen käytön kannalta.

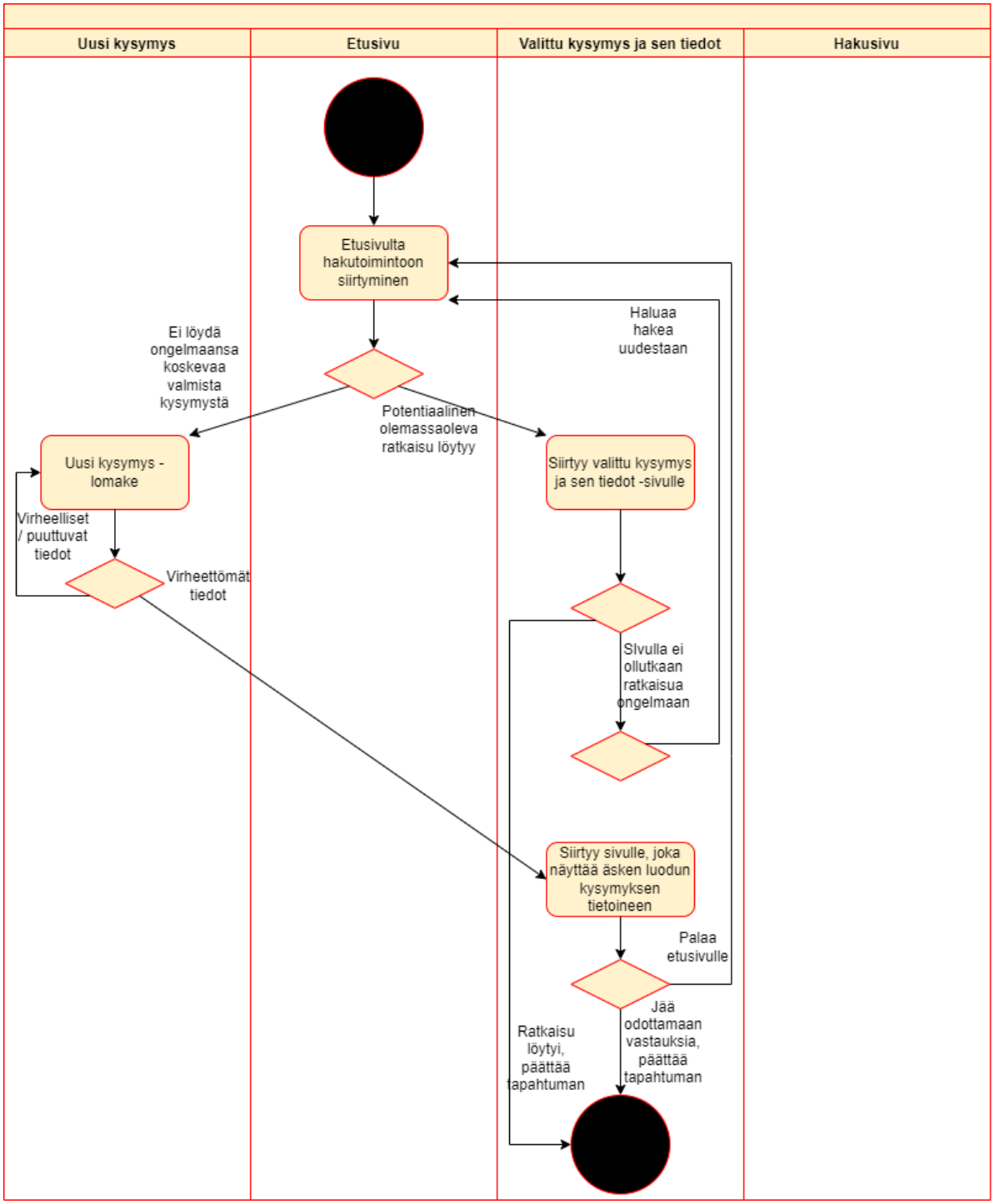
Käyttöliittymäkaavioissa sääntönä on, että sitä, kun käyttäjä jostain päin internetiä saapuu sovelluksen etusivulle ensimmäistä kertaa, ei merkitä omaksi tapahtumakseen, vaan tässä tapauksessa ensimmäinen käyttötapahtuma on jo etusivulta hakutoimintoon siirtyminen.



Kuvio 16. Asiakkaan käyttöliittymä.

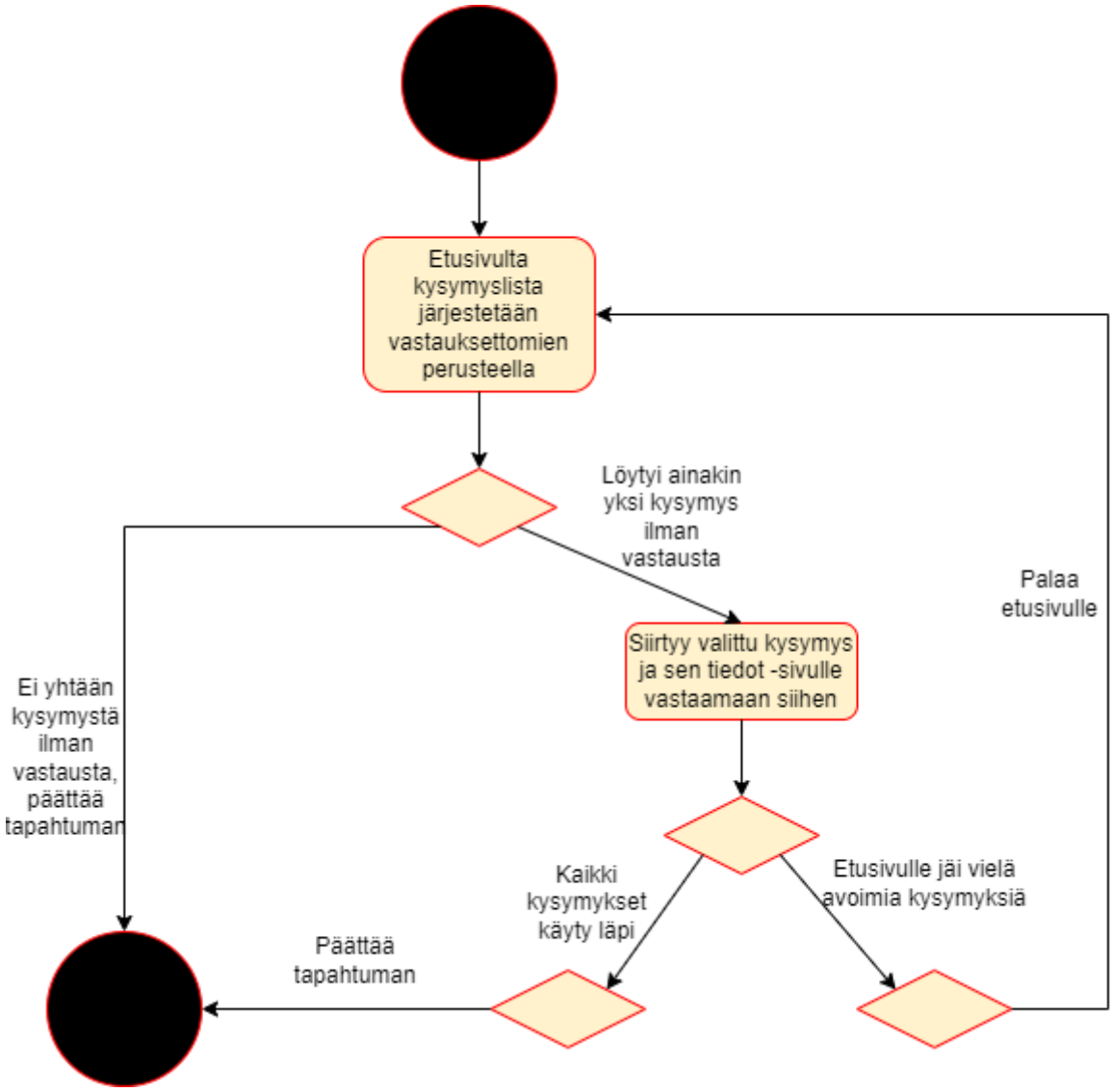
Uimaratakaavio on tietyn tyyppinen vuokaavio, joka kuvaa kuka tekee mitäkin prosessissa. Vertauskuvallisesti se asettaa prosessin vaiheet vaaka- tai pystysuoraan ”uimaratoja” apuna käyttäen. Se näyttää näiden ratojen välisiä yhteyksiä, kommunikaatiota ja vaihtoja, ja se pystyy korostamaan prosessin aikaista haaskausta, tarpeettomuutta tai tehottomuutta. (Lucidcharts 2022.)

Lyhyesti sanottuna, uimaratakaavio tuo esille käyttöliittymän ”työtaakan” vastuunjaon. Asiakkaan uimaradat (kuvio 17) esittävät käyttöliittymän täysin eri valossa kuin aiempi kaavio teki. Asiakkaan uimaratakaaviossa valittu kysymys ja sen tiedot -sivu saa huomattavan tärkeän roolin.



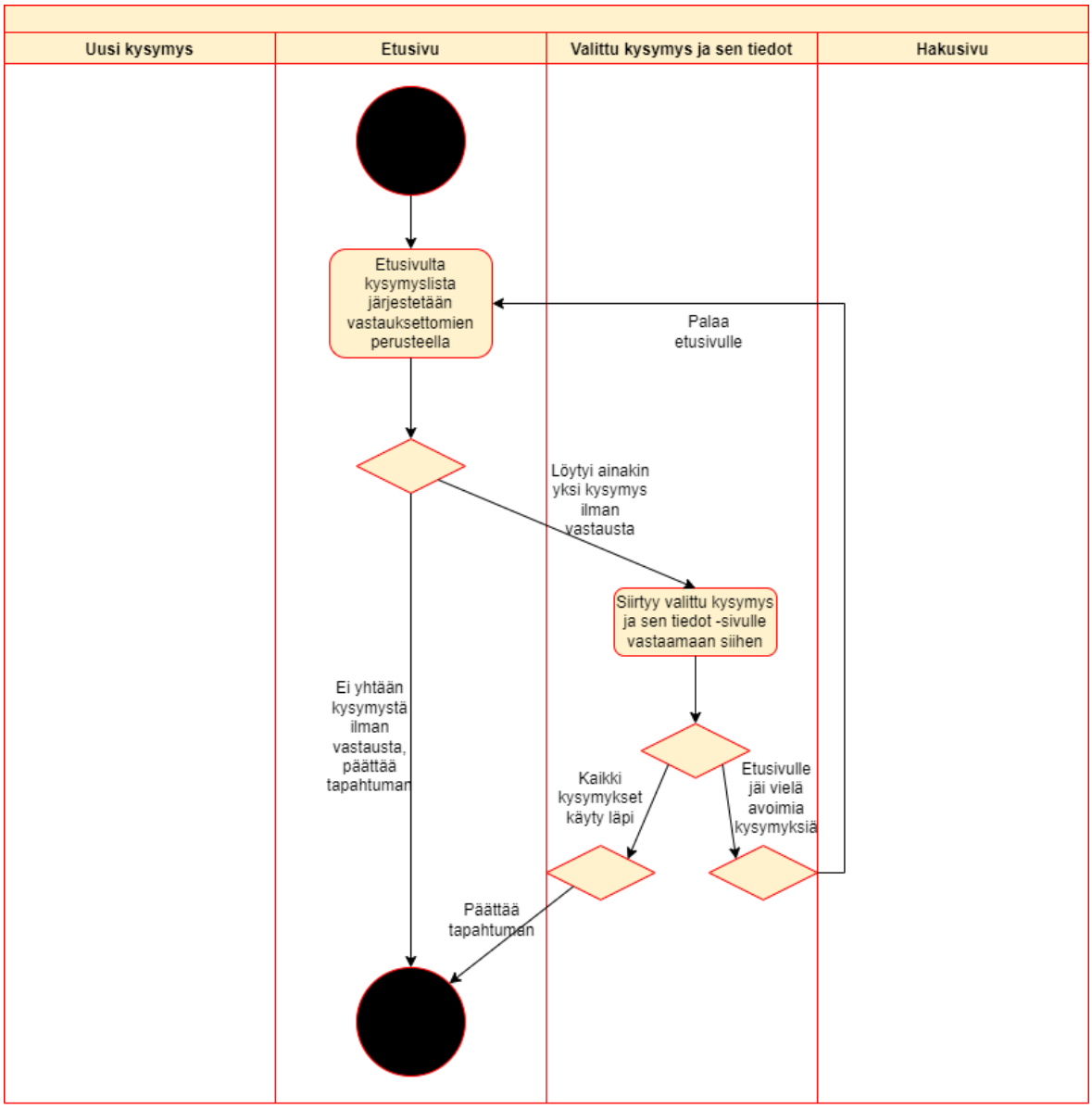
Kuvio 17. Asiakkaan uimaradat.

Ylläpitäjän käyttöliittymä (kuvio 18) osoittaa sen, että hänellä on kaksi päätehtävää. Ensimmäinen niistä on etusivun tarkkailu uusien kysymysten tullessa ja toinen on sitten niihin vastaaminen.



Kuvio 18. Ylläpidon käyttöliittymä.

Ylläpidon uimaradat (kuvio 19) vain vahvistaa edellisen kuvan pohdinnat. Ylläpidolla ei ole mitään tekoa uusi kysymys- ja hakusivujen kanssa, vaan heidän huomionsa keskittyy täysin etusivun ja valittu kysymys -sivun käyttämiseen.



Kuvio 19. Ylläpidon uimaradat.

6 Kvantitatiivinen tutkimus käyttökokemuksesta

6.1 Tietopohja

Käyttäjäkokemussuunnittelu (UX, user experience design) ja käyttöliittymäsuunnittelu (UI, user interface design) ovat toisiinsa lomittuvia käsitteitä ja ratkaisevassa asemassa siinä, miten onnistunut verkkosivun toteutus on. Ohjelmistoprojekteissa kumpaakin asiaa työstetään yhtä aikaa. Käyttäjäkokemukseen vaikuttaa olennaisesti kokemus käyttöliittymästä, joten käyttöliittymäsuunnittelun voidaan lukea kuuluvan käyttäjäkokemussuunnittelun alle. Lopulliseen asiakaskokemukseen vaikuttavat molemmat käsitteet. Vaikka näissä on paljon samaa, kuten samankaltaiset tavoitteet, verkko-ohjelmistoprojekteissa niillä on kuitenkin erilaiset roolit. (Hurja 2021.)

Verkkosivujen käyttäjäkokemuksella tarkoitetaan käyttäjän kokemusta sen käytöstä. Käyttäjäkokemussuunnittelussa käyttäjän matka verkkosivustossa määritellään yksityiskohtaisesti. Lisäksi otetaan huomioon ihmiselle tyypilliset psykologiset toimintatavat ja yrityksen verkkosovellukselle asettamat tavoitteet. Digitaalisten palveluiden lisäksi käyttäjäkokemussuunnittelua voidaan tehdä myös fyysisille tuotteille, sillä käyttäjäkokemus syntyy kaikenlaisten tuotteiden ja palveluiden käytöstä. Käyttökokemuksen muovaavat elementit pyritään ottamaan käyttäjäkokemussuunnittelussa huomioon. (Hurja 2021.)

Yksi verkkosivustojen tärkeimmistä ominaisuuksista on käytettävyys, joka on käyttäjäkokemuksen tärkeä osa-alue. Kun verkkosivustoa käyttämällä saavutetaan haluttu tavoite, on sen osalta saavutettu käytettävyydeltään hyvälaatuinen tuotos. Palvelun toiminnallisuudesta saadaan läpinäkyvä ja ennustettava, kun keskitytään tavoitteen saavuttamiseen. Tärkeää on, että palvelua voi käyttää intuitiivisesti. Käyttäjän on esimerkiksi voitava navigoida ja asioida sivustolla helposti, jotta hän löytää sen, mitä on tullut hakemaan. Käyttäjäkokemussuunnittelu vaatii tarkkaa tietoa tavoitteista ja käyttäjistä. Käyttäjäkokemussuunnittelun taustatutkimuksissa käytetään usein apuna palvelumuotoilua käyttötapausten ja toiminnallisuuksien määrittelyyn sekä prototyyppejä ominaisuuksien ja visuaalisen ilmeen testaamiseen. (Hurja 2021.)

Tyypillinen käyttäjäkokemuksen suunnittelun aikaisessa vaiheessa käytetty menetelmä on rautalankamallit. Sen avulla voidaan määritellä verkkosivuston tietorakenne yksinkertaisena viivapiirroksena, kuten opinnäytetyön ohjelmistotuotantolukujen aluissa on esitetty. Lopputuotteen käytettävyys ja onnistuminen riippuu usein siitä, kuinka hyvin rautalankamalli on tehty. Käyttäjäkokemussuunnitteluun kuuluvat osa-alueet ovat rautalankamallit ja prototyypit, käyttäjätutkimus, skenaariot, interaktiivisuus ja kohderyhmän määrittely. (Hurja 2021.)

Käyttöliittymäsuunnittelu viittaa nimensä mukaisesti siihen, mitä loppukäyttäjä näkee verkkosivuston pinnassa. Käyttöliittymäsuunnittelulla ei siis suunnitella fyysisiä tuotteita, toisin kuin käyttäjäkokemussuunnittelulla. Värein, fontein, kuvin ja muodoin, eli visuaalisin keinoin, voidaan vaikuttaa siihen, miten käyttäjä on vuorovaikutuksessa verkkosivuston kanssa. Käyttöliittymäsuunnittelu on vaihe, jossa verkkosivuston visuaalinen ilme muotoillaan yrityksen brändin mukaiseksi sekä vaikutetaan siihen, kuinka helppoa on tiedon löytäminen ja kuinka saavutettavissa se on. Saavutettavuuteen vaikuttaa esimerkiksi tekstin ja taustan värien kontrastisuhde. Käyttöliittymäsuunnittelussa tavoitteena on, että käyttöliittymää on mahdollisimman yksinkertaista käyttää ja se palvelee myös lopullisia liiketoiminnallisia tavoitteita. Web-sovelluksen visuaalinen ilme on osa käyttökokemusta ja vaikuttaa myös brändiin kohdistuvaan mielikuvaan. Käyttöliittymäsuunnittelun aikana tyypillisesti luodaan niin kutsuttuja mockup-malleja. Mockup-ulkoasusuunnitelman avulla voidaan testata erilaisia näkymiä, web-sovelluksen värimaailmaa, fontteja tai vaikkapa kuvia. Käyttöliittymäsuunnitteluun kuuluvia osa-alueita ovat visuaalinen suunnittelu, värit, mockupit ja asettelut, typografia ja brändi-ilme. Rautalankamallit ja mockupit eivät aina riitä havainnollistamaan koko toteutusta tai etenkään käytettävyyttä ja käyttäjäpolkuja. Silloin suunnittelussa voidaan käyttää apuna lisäksi vuorovaikutteista mockup-mallia ja prototyyppiä. (Hurja 2021.)

Yhteenvetona todettakoon, että käyttäjäkokemussuunnittelussa käyttäjien ongelmien tunnistaminen ja ratkominen on keskiössä. Käyttöliittymäsuunnittelussa taas on kyse visuaalisista keinoista, joilla pyritään rakentamaan intuitiivinen, miellyttävä ja interaktiivinen käyttöliittymä. Käyttäjäkokemussuunnittelulla muotoillaan käyttäjän matka, joka viitoitetaan käyttöliittymäsuunnittelun avulla visuaalisilla elementeillä. (Hurja 2021.)

Käyttäjäkokemus ja miksi sillä on väliä

Garrettin (2011, 6) mukaan käyttäjäkokemus tarkoittaa kokemusta, jonka tuote luo ihmisille, jotka käyttävät sitä tosielämän tilanteissa. Kun tuotetta suunnitellaan, ihmiset kiinnittävät paljon huomiota siihen, mitä se tekee. Käyttäjäkokemus, eli se miten tuote toimii, on toinen usein huomiotta jätetty osa tätä yhtälöä. Tämä voi usein määrittää sen, onko tuote onnistunut vai ei. Käyttäjäkokemuksessa olennaista ei ole tuotteen tai palvelun sisäiset toiminnot, vaan kuinka se toimii ulkopuolelta katsottuna eli kuinka käyttäjä on kontaktissa siihen. Kun joku kysyy, millaista tuotteen tai palvelun käyttö on, he haluavat tietää käyttäjäkokemuksesta. Onko yksinkertaisten asioiden tekeminen vaikeaa? Onko se helposti ymmärrettävissä? Miltä tuntuu olla vuorovaikutuksessa tuotteen kanssa?

Garrett (2011, 8) toteaa, että käyttäjäkokemussuunnittelu usein käsittelee kysymyksiä asiayhteyteen liittyen. Esteettinen suunnittelu varmistaa, että tuotteella on houkutteleva muoto

ja tekstuuri. Funktionaalinen suunnittelu varmistaa, että esimerkiksi painonappi suorittaa halutun toiminnon tuotteella tai palvelulla. Käyttäjäkokemussuunnittelu varmistaa, että tuotteen tai palvelun esteettinen ja funktionaalinen aspekti toimivat muun tuotteen asiayhteydessä. Voidaan kysyä, onko esimerkiksi painonappi liian pieni tärkeää toimintoa varten? Käyttäjäkokemussuunnittelu varmistaa myös, että esimerkin painonappi toimii siinä asiayhteydessä, jossa käyttäjä pyrkii saavuttamaan jonkin tavoitteen. Voidaan kysyä, onko painonappi oikeassa paikassa suhteessa muihin ohjaimiin, joita käyttäjä käyttäisi siinä tilanteessa?

Jos sivusto koostuu pääasiassa siitä, mitä ammattilaiset kutsuvat sisällöksi, eli informaatiosta, silloin sivuston yksi päätavoitteista on kommunikoida se mahdollisimman tehokkaasti. Pelkästään informaation tuuppaaminen sivulle ei riitä. Se on esitettävä sellaisessa muodossa, joka auttaa ihmisiä omaksumaan ja ymmärtämään sen. Muutoin käyttäjä ei välttämättä koskaan saa tietää, että sivusto tarjoaa juuri sitä tuotetta tai palvelua, jota käyttäjä hakee. Ja vaikka käyttäjät onnistuisivat löytämään sen informaation, he luultavasti tekevät johtopäätöksen, että jos sivuston käyttäminen on hankalaa, myös yrityksen kanssa toiminen on hankalaa. (Garrett 2011, 12.)

Käyttäjäkokemuksen elementit

Käyttökokemussuunnittelun keskeisin osa on varmistaa, että mikään ohjelmiston käyttökokemuksen aspekti ei tapahdu ilman tuottajan tietoista, suunniteltua tarkoitusta. Tämä tarkoittaa sitä, että otetaan huomioon jokaisen toimen jokainen mahdollisuus, jota käyttäjä voi sovelluksessa tehdä ja että ymmärretään jokaista käyttäjän odotusarvoa tuon prosessin jokaisella osa-alueella. Se kuulostaa isolta työltä ja tietyllä tapaa se onkin. Paloittelemalla käyttökokemussuunnittelun työ komponenttielementteihin voidaan paremmin ymmärtää tehtävää paremmin kokonaisuutena. (Garrett 2011, 19.)

Pintataso on taso, jossa käyttäjä näkee verkkosivuja, jotka koostuvat kuvista ja tekstistä. Jotkin näistä kuvista ovat klikattavissa, suorittaen jonkin tietyn funktion kuten vieden sinut ostoskoriin. Jotkin kuvat ovat vain havainnollistamista varten, kuten valokuvat tuotteesta, jota myydään tai itse sivuston logo. Luurankotaso on tämän alapuolella sivustolla: painonappien, ohjaimien, kuvien tai tekstikappaleiden sijoittelua varten. Luuranko on suunniteltu optimoimaan näiden elementtien järjestely maksimoidakseen tehokkuutta ja hyötyä, jotta loppukäyttäjä muistaa logon ja löytää vaikkapa tuon ostoskorinappulan, kun sen tarve ilmaantuu. Rakennetaso kuvaa abstraktilla keinolla luurangon konkreettista olemusta. Luuranko saattaa määrittää käyttöliittymäelementtien sijoittelun kassasivulla: rakenne taas määrittää miten loppukäyttäjä pääsevät tuolle sivulle ja minne heidän kuuluisi mennä silloin, kun he valmistuvat sieltä. Luuranko voisi määritellä navigoinnillisten elementtien järjestelyn

sallien käyttäjän selata eri tuotekategorioita: rakenne määrittäisi mitä nuo kategoriat ovat. Rakenne määrittää sen, miten eri ominaisuudet ja toiminnot sivustolla sopivat yhteen. Kattavuustaso pitää kietoo sisäänsä sen, mitä nuo ominaisuudet ja toiminnot ovat. Esimerkiksi jotkin kaupalliset sivustot tarjoavat mahdollisuuden tallentaa hiljattain käytetyn toimitusosoitteen, jotta sitä voidaan käyttää myöhemminkin. Strategia ottaa sisälleen ei ainoastaan sen, mitä sivustoa pyörittävä taho haluaa sivustoltaan vaan myös mitä käyttäjät hakevat sivuston tarkoitukselta. Verkkokauppaesimerkissä tietyt strategiset tavoitteet ovat melko ilmiselviä: käyttäjät haluavat ostaa tuotteita ja ylläpitäjä haluaa myydä niitä. Muut tavoitteet, kuten markkinointi tai käyttäjien tuottama sisältö, eivät välttämättä ole niin helposti sanoitettavissa. (Garrett 2011, 20—21.)

Rakennetaan pohjalta huipulle

Jokaisella tasolla ongelmat, joita täytyy ratkoa, muuttuvat vähän vähemmän abstrakteiksi ja vähän enemmän konkreettisiksi. Matalimmalla tasolla ylläpitäjää ei kiinnosta sivuston tai palvelun lopullinen muoto, vaan ainoa tärkeä asia on se, miten sivusto sopii strategiaan, toki vastaten käyttäjien tarpeisiin. Korkeimmalla tasolla eniten kiinnostaa vain sivuston ulkoasun yksityiskohdat. Taso tasolta valinnat, joita täytyy tehdä muuttuvat vähän tarkemmiksi ja sisällyttävät hienompia yksityiskohtia. (Garrett 2011, 21.)

Jokainen taso on riippuvainen niistä tasoista, jotka ovat sen alapuolella. Pintataso riippuu luurangosta, joka riippuu rakenteesta, joka riippuu laajuudesta, joka riippuu strategiasta. Kun tehdyt päätökset eivät sovi ylä- ja alapuolen kanssa, projektit lähtevät raiteiltaan, aikataulut venyvät ja kulut nousevat samalla kun kehitystiimi yrittää kasata yhteen komponentteja, jotka eivät luonnostaan sovi yhteen. Mikä pahinta, kun sivusto viimein julkaistaan, käyttäjät usein vihaavat sitä, koska se ei tuota tyydyttävää kokemusta. Tämä riippuvuus tarkoittaa sitä, että tehdyt valinnat strategiatasolla omaavat tietynlaisen heijastusvaikutuksen koko ketjun päästä päähän. Käänteisesti ajateltuna, saatavilla olevat valintavaihtoehdot jokaisella tasolla ovat rajoittuneita päätöksiin, joita tehdään tasoilla, jotka ovat sen alapuolella. (Garrett 2011, 22.)

Tämä ei kuitenkaan tarkoita sitä, että jokainen valinta, joka tehtiin alemmilla tasoilla, täytyy tehdä ennen kuin tasoa yläpuolella voidaan tarkastella. Riippuvuudet toimivat molempiin suuntiin, ja valinta, joka tehdään ylemmällä tasolla voi pakottaa uudelleenarvioimaan tilannetta alemmilla tasoilla. Jokaisella tasolla tehdään päätöksiä sen mukaan mitä kilpailijat tekevät, mitä alan käytännöt ovat, mitä tiedetään käyttäjistä ja miten maalaisjärki toimii. Näillä päätöksillä voi olla heijastusvaikutuksia molempiin suuntiin. Paras työskentelytapa on työstää yhtä tasoa siten, että sen oma työ loppuu juuri ennen kuin seuraavan tason työ

loppuu, siis osittain päällekkäin. Ei siis rakenneta talon kattoa ennen kuin tiedetään perustusten muoto. (Garrett 2011, 24.)

Internetin ja sen sivustojen voidaan karkeasti luokitella kuuluvan kahteen leiriin: toiminnallisiin sivustoihin ja tiedonlähdesivustoihin. Tämä seikka voidaan linkittää äsken esitettyihin tasoihin. Strategiatasolla samat strategiset asiat astuvat esiin sekä toiminnallisorientoituneissa sivustoissa että informaatio-orientoituneissa sivustoissa. Käyttäjän tarpeet ovat sivustolle päämääriä, jotka tulevat organisaation ulkopuolelta. Täytyy ymmärtää mitä yleisö tahtoo sivustolta ja miten asiat linkittyvät heidän muihin päämääriin. Ylläpitäjän omat tavoitteet sivuston suhteen tasapainotetaan käyttäjän tarpeisiin nähden. Nämä tuotepäämäärät voivat olla liiketoimintapäämääriä, ”luodaan miljoonan euron edestä myyntiä verkon ylitse tänä vuonna” tai muunlaisia päämääriä, ”tiedotetaan äänestäjiä seuraavien vaalien ehdokkaista”. Kattavuustasolla toiminnallisella puolella strategia kääntyy kattavuudeksi toiminnallisten määritelmien luomisen kautta: yksityiskohtainen kuvaus sivuston ominaisuuspaletista. Informaatiopuolella kattavuus sanelee sisältövaatimukset eli kuvauksen siitä, mitä erilaisia sisältöelementtejä vaaditaan. Kattavuudelle annetaan rakennetta toiminnallisuuspuolella vuorovaikutussuunnittelun kautta, jossa määritellään kuinka järjestelmä reagoi käyttäjän vasteeseen. Tiedonlähteille rakenne on informaatioarkkitehtuuria: sisältöelementtien järjestelyä helpottamaan ihmisymmärrystä. Luurankotaso jakautuu kolmeen komponenttiin: molemmilla puolilla täytyy käsitellä informaatiosuunnittelua: tiedon esittämistä ymmärrettävästi. Toiminnallisuusorientoituneissa tuotteissa luuranko sisältää myös käyttöliittymäsuunnittelua, tai käyttöliittymäelementtien järjestämistä siten, että käyttäjä voi olla vuorovaikutuksessa sivuston toiminnallisuuksien kanssa. Tiedonlähteen käyttöliittymä on sen navigointisuunnittelu: sarja näyttöelementtejä, jotka antavat käyttäjän liikkua tietoarkkitehtuurin läpi. Lopuksi vielä on jäljellä pinta. Riippumatta siitä, onko sivusto toiminnallisorientoitunut vai tietolähde, päähuolenaihe on sama: lopputuotteen aistikokemus. (Garrett 2011, 28—30.)

Tämä malli jaettuna kivoihin laatikoihin ja tasoihin on miellyttävä tapa ajatella käyttökokemusongelmia. Todellisuudessa voi olla vaikeaa tunnistaa saadaanko jokin erityinen käyttökokemusongelma ratkottua keskittymällä yhteen elementtiin vai johonkin toiseen. Voiko muutos visuaaliseen puoleen tehdä tempun vai täytyykö pinnan alla olevaa navigointia korjata? Jotkin ongelmat vaativat huomiota useilla alueilla ja jotkut taas kulkevat ristiin rastiin tämän mallin rajoilla. (Garrett 2011, 31.)

Tämä luvun loppuosa käsittelee tasoja yksityiskohtaisemmalla tasolla yksi kerrallaan. Tarkastelun kohteena ovat strategiataso, kattavuustaso, rakennetaso, luurankotaso ja pintataso.

Strategiataso

Selkeästi ilmaistu strategia on menestyvän käyttökokemuksen perusta. Tieto siitä, mitä haluamme tuotteella saavutettavan organisaation kannalta ja mitä haluamme sen saavan aikaan käyttäjillemme vaikuttaa päätöksiin, joita meidän on tehtävä käyttökokemuksen kaikilla osa-alueilla. (Garrett 2011, 35.)

Verkkosivuprojektit epäonnistuvat, mikäli heti alusta alkaen ei kysytä itseltä mitä haluamme saavuttaa tuotteen avulla ja mitä käyttäjät haluavat saavuttaa sillä. Vastaamalla ensimmäiseen kysymykseen, kuvataan sivuston tavoitteet, jotka tulevat organisaation sisältä. Toinen kysymys käsittelee käyttäjien tarpeita, tuotteelle määrättäviä tavoitteita, jotka tulevat ulkopuolelta. Yhdessä tuotetavoitteet ja käyttäjätarpeet muodostavat käyttökokemuksen strategiatason, suunnitteluprosessin perustan, kun käyttökokemusta pohditaan. (Garrett 2011, 36.)

Avenla Oy haluaa virtaviivaistaa asiakastukea, säästää työaikaa ja luoda internetiin avoimen tietovaraston, joka kuvaa ratkaisuja alkutuotteen yleisiin ja erityisiin käyttöönoton ongelmatilanteisiin. Asiakas haluaa löytää jo valmiiksi laadittuja ohjeita ongelmatilanteiden korjaamiseksi tai mikäli valmista materiaalia ei ole, mennä itse kysymään uusia kysymyksiä saadakseen vastauksia.

Brändi-identiteetti on paljon muutakin kuin vain logoja, väripaletteja ja typografiaa. Internetsivuston kanssa vuorovaikutuksessa oleminen yhdistyy vääjäämättä käyttäjien mielissä vaikutelmaan organisaatiosta. Täytyy tehdä valinta sen välillä, onko tuon vaikutelman syntyminen sattumankauppaa vai tulos tietoisista päätöksistä, joita tehtiin sivustoa suunnitellessa. Brändäys ei ole vain kaupallisia toimijoita varten, jokainen verkkosivuston omistava organisaatio, ei-kaupallisista laitoksista hallituksen virastoihin ja yksityishenkilöihin, luo käyttökokemuksella vaikutelman itsestään. (Garrett 2011, 38—39.)

Käyttökokemusta on tärkeää myös mitata. Joskus mittarit liittyvät tuotteeseen itseensä ja miten sitä käytetään. Kuinka paljon aikaa keskiverto käyttäjä käyttää sivustolla? Analytiikkatyökalut voivat auttaa mittaamaan tätä. Usein halutaan, että käyttäjät kuluttaisivat enemmän aikaa sivustolla vieraillessaan. Toisaalta taas, jos sivuston idea on tarjota nopeasti informaatiota ja toiminnallisuuksia käyttäjälle ja sitten käyttäjä menee pois, käyttäjän sivustolla kuluttama aika pitäisi saada pienenemään. (Garrett 2011, 39.)

Kaikkien menestysmittareiden ei välttämättä täydy olla johdettuja suoraan sivustosta. Voidaan myös mitata sivuston epäsuoria vaikutuksia. Jos sivuston tarkoitus on tarjota ratkaisuja toisen alkutuotteen ongelmatilanteisiin, sähköpostitse tapahtuvien yhteydenottojen määrän pitäisi alkaa vähentyä. Sivuston käyttökokemus ei voi auttaa paljoakaan itsestään

tuomaan uusia käyttäjiä sivustolle, vaan silloin täytyy luottaa puskaradioon tai markkinointiin. Käyttökokemuksella on kuitenkin paljon vaikutusta siihen, tulevatko käyttäjät takaisin. Takaisin palaamisen mittaaminen voi olla hieno keino arvioida sitä, saavatko käyttäjät tarpeensa tyydytettyä, mutta pitää olla varovainen: joskus käyttäjät eivät palaa, jos vaikka kilpailija on käynnistänyt massiivisen mainostuskampanjan tai organisaatio on saanut julkisuudessa huonoa mainetta. Kaikki mittarit eristettyinä tapauksina voivat olla harhaanjohtavia: asiaa täytyy tarkastella kokonaisvaltaisesti nähdäkseen mitä oikein tosiasiassa tapahtuu. (Garrett 2011, 41.)

Käyttökokemusta suunnitellessa on hyvä pohtia käyttäjiä ryhminä. Käyttäjät voidaan jaotella ryhmiin demografisten piirteiden, kuten vaikkapa siviilisäädyn, iän tai sukupuolen, perusteella tai vaikkapa sen perusteella, kuinka paljon he tietävät sivuston tarkoituksesta. Sivuston, joka palvelee IT-alan ammattilaisia, suunnittelu voi olla hyvin erilaista kuin jos se olisi suunniteltu noviisikäyttäjille. Nämä erot kokemuksessa ja asiantuntemuksessa voivat luoda pohjaa käyttäjien segmentoimiselle. Käyttäjäsegmenttien luominen on vain keino käyttäjien tarpeiden paljastamiseksi. Tarvitaan vain niin monta segmenttiä kuin on käyttäjätarpeita. (Garrett 2011, 44.)

Ymmärtääkseen mitä käyttäjät tarvitsevat, täytyy ensin saada ymmärrystä siitä, keitä he ovat. Käyttäjätutkimuksen ala on omistettu tarvittavan datan keruulle kehittääkseen sitä ymmärrystä. Jotkin tutkimustyökalut, kuten kyselyt tai haastattelut, ovat parhaiten suunnattuja keräämään tietoa käyttäjien yleisistä asenteista ja näkökulmista. Toiset työkalut, kuten käyttäjätestit, soveltuvat paremmin tuottamaan informaatiota asiakkaan käyttäytymisen aspekteista ja sovelluksen kanssa vuorovaikutuksessa olemisesta. (Garrett 2011, 46.)

Verkkopalveluiden suunnittelussa tulee usein törmänneeksi termiin käytettävyys. Jotkut viittaavat tällä käytäntöön testata tuotoksia tyypillisellä esimerkkikäyttäjällä. Toiset taas tarkoittavat sillä erityisten kehitysmetodologioiden omaksumista. Jokainen lähestymistapa käytettävyyteen tähtää sivuston käytön helpottamiseen. Kaikki käytettävyyden määritelmät omaavat saman ydinperiaatteen: käyttäjät tarvitsevat käyttökelpoisia tuotteita. Se on kaikista yleismaailmallisin käyttäjätarve. (Garrett 2011, 48.)

Strategian tulisi olla aloituspiste käyttökokemussuunnitteluprosessille, mutta se ei tarkoita, että strategian tulisi olla kiveen hakattu ennen kuin projekti voi mennä eteenpäin. Vaikka liikkuvaan maaliin osuminen voi olla valtavaa ajan- ja resurssien hukkaa, puhumattakaan valtavasta määrästä sisäistä turhautumista, strategiat voivat ja niiden pitäisikin kehittyä ja tarkentua. Kun sitä tarkistetaan ja korjataan järjestelmällisesti, strategiatyö voi olla jatkuva inspiraation lähde koko käyttökokemukseen liittyvän suunnitteluprosessin läpi. (Garrett 2011, 54.)

Kattavuustaso

Kun on olemassa selkeä käsitys siitä mitä halutaan ja mitä käyttäjät haluavat, ymmärretään, kuinka tyydytetään nuo strategiset tavoitteet. Strategiasta tulee laajuustason asia, kun käyttäjien tarpeet ja tuotetavoitteet muunnetaan erityisiin vaatimuksiin siitä, mitä sisältöä ja toiminnallisuuksia tuote käyttäjille tarjoaa. (Garrett 2011, 57.)

Ihmiset tekevät asioita, koska niiden prosesseissa on arvoa, kuten hölkkäämistä tai pianon soittoa. Toisia asioita tehdään, koska tuotteessa on arvoa, kuten esimerkiksi juustokakun leipominen tai auton korjaaminen. Laajuuden määrittäminen projektille on molempia: arvokas prosessi, joka johtaa arvokkaaseen tuotteeseen. Vaatimusten määrittely poissulkee epäselvyyksiä suunnitteluprosessista. (Garrett 2011, 58.)

Sille, että vaivautuu määrittelemään vaatimukset, on olemassa pääasiassa kaksi syytä. Ensimmäinen syy on se, että tiedetään mitä ollaan rakentamassa. Kun vaatimukset määritellään, projektista tulee jotain konkreettista, jonka parissa kaikki organisaation kaikilla tasoilla, aina johdosta matalan tason kehittäjiin, voivat työskennellä. Ilman tätä eri ihmisillä on rikkinäinen puhelin -leikin tapaan eri käsitys asioista ja jokaisen kuvaus projektista vaihtelee. Tai vielä pahempaa, jokainen olettaa, että joku toinen hoitaa jonkin tuotteelle tärkeän ominaisuuden suunnittelua ja kehitystä, kun itse asiassa kukaan ei hoida. (Garrett 2011, 59.)

Toinen syy vaatimusmäärittelylle on se, että tiedetään, mitä ei olla rakentamassa. Monet ominaisuudet kuulostavat hyviltä ideoilta, mutta ne eivät välttämättä asetu yksiin tuotteen strategisten tavoitteiden kanssa. Vaatimukset, joita ei voi täyttää nykyisessä aikataulussa, voivat muodostaa seuraavan kehityssyklin virstanpylvään. (Garrett 2011, 60—61.)

Laajuustasolla aloitetaan abstraktista kysymyksestä ”miksi tuotetta tehdään?”, jota käsiteltiin strategiatasolla ja sen päälle rakennetaan uudella kysymyksellä: ”mitä aiotaan tehdä?”. Internetin jako alustaksi toiminnallisuuksien tarjoamiseen ja informaatiolähteeksi alkaa vaikuttaa laajuustasolla. Toiminnallisuuspuolella huolenaiheena on se, mitä ajatellaan sivuston ominaisuuspalettina. Informaatiopuolella hoidetaan sisällön asioita, jotka ovat toimituksellisten ja markkinoinnillisten viestintäryhmien ydinaluetta. Mitä tulee laajuuden määritelmään, sisältö ja toiminnallisuudet voidaan ottaa käsittelyyn hyvin samankaltaisilla tavoilla. Läpi tämän aliluvun, käytetään termiä ominaisuus viittaamaan sekä ohjelmistotoiminnallisuuksiin että sisältöön. (Garrett 2011, 61—62.)

Tämän aliluvun kieli on enimmäkseen ohjelmistokehityksen kieltä, vaikka tässä esitellyt konseptit pätevät myös sisältöön. Sisällönkehitys sisältää usein vähän muodollisia vaatimusmäärittelyjä kuin ohjelmisto, mutta taustalla olevat perusperiaatteet ovat samat. (Garrett 2011, 63.)

Minkä tahansa teknologiatuotteen toiminnallisilla vaatimuksilla on sisältöimplikaatioita. Tuleeko tuotteessa olemaan virheilmoitusviestejä? Joka kerta, kun verkkosivuilla näkyy virheilmoitus ”Null input field exception”, tiedetään, että jonkun ohjelmoijan tilapäinen viesti löysi tiensä lopulliseen tuotteeseen, koska kukaan ei tehnyt tuosta virheilmoituksesta sisältövaatimusta. Lukuisat väitetysti teknologiset projektit olisivat voineet parantua mittaamattomasti, jos kehittäjät olisivat yksinkertaisesti katsoneet sovellusta pitäen silmällä sisältöä. (Garrett 2011, 64.)

Jotkin vaatimukset pätevät tuotteeseen kokonaan. Brändivaatimukset ovat yksi yleinen esimerkki tästä: tietyt tekniset vaatimukset, kuten tuetut selaimet ovat toinen. Toiset vaatimukset pätevät tiettyyn ominaisuuteen. Usein, kun ajatellaan vaatimuksia, ajatellaan lyhyitä kuvauksia yksittäisestä vaatimuksesta, joka tuotteelta vaaditaan. (Garrett 2011, 65.)

Antoisin vaatimusten lähde tulee aina olemaan käyttäjät itse. Usein kuitenkin vaatimukset tulevat ihmisiltä organisaatiossa, joilla on jotain sanottavaa siitä, mitä tuotteeseen otetaan mukaan. Joka tapauksessa paras keino selvittää, mitä ihmiset haluavat, on kysyä heiltä siitä. Käyttäjätutkimustekniikat, jotka kuvattiin edellisessä aliluvussa, voivat auttaa saamaan paremman ymmärryksen minkälaisia ominaisuuksia käyttäjät haluavat tuotteessa nähdä. (Garrett 2011, 65.)

Huolimatta siitä, kuinka laaja tai monimutkainen projekti on, muutama yleismaailmallinen sääntö pätee minkä tahansa laisen vaatimuksen kirjoittamiseen. Ensimmäinen on ”ole positiivinen”. Sen sijaan, että kuvataan huonoa asiaa, jota järjestelmän ei pitäisi tehdä, kuvataan mitä se tekee estääkseen tuon huonon asian. Esimerkiksi ”järjestelmä ei salli käyttäjän ostaa leijaa ilman narua”. Tämän sijaan kirjoitetaan ”järjestelmä ohjaa käyttäjän leijanarusivulle, jos käyttäjä yrittää ostaa leijan ilman narua”. Toinen sääntö on ”ole täsmällinen”. Jättämällä mahdollisimman vähän tulkinnanvaraa on ainoa keino määrittää, onko vaatimus täytetty. Vertaillaan seuraavia lauseita: ”suosituimmat videot korostetaan” ja ”videot, joilla on eniten katselukertoja kuluneelta viikolta ilmestyvät listalle ensimmäisinä”. Ensimmäinen lause ei vaadi kummoista tarkastelua ennen kuin sen hyvyys alkaa rakoilla. Mikä lasketaan suosituksi? Minkä katsotaan olevan korostamista? Kolmas sääntö on välttää subjektiivista kieltä. Eräs erittäin subjektiivinen vaatimus olisi ”sivustolla tulee olemaan uusimpia virtauksia seuraava, huomiota herättävä tyyli”. Vaatimusten täytyy olla merkittävissä väärennettävissä, tarkoittaen sitä, että on mahdollista demonstroida, onko vaatimus täytetty. Eri ihmisillä on varmasti eri mielipide siitä mitä huomiota herättävä tarkoittaa. Tämä ei tarkoita, että ei voida tehdä vaatimusta huomiota herättävyydestä. Täytyy vaan määritellä kriteeri tälle: ”sivuston täytyy olla huomiota herättävä postivirkailija Veikon mielestä”. Tätäkin parempi

vaatimus voisi olla ”sivuston ulkoasun täytyy mukailla yrityksen asiakirjaa brändäysohjeistuksesta”. (Garrett 2011, 69—71.)

Ei ole vaikeaa kerätä ideoita mahdollisia vaatimuksia varten. Lähes jokaisella, joka säännöllisesti on tekemisissä tuotteen kanssa, olivatpa he sitten organisaation sisällä tai ulkopuolella, on ainakin yksi idea ominaisuudesta, jonka voisi lisätä. Vaikea osuus tässä on lajitella ominaisuudet, jotka pitäisi ottaa laajuuteen mukaan. On itse asiassa aika harvinaista, että eteen sattuu yksi-yhteen-korrelaatio strategisten tavoitteiden ja vaatimusten välillä. Joskus yksi vaatimus voi päteä moneen strategiseen tavoitteeseen. Samalla tavalla yksi tavoite voi korreloida usean erilaisen vaatimuksen kanssa. (Garrett 2011, 75.)

Koska laajuus on rakennettu strategian päälle, täytyy arvioida mahdollisia vaatimuksia sen perusteella täyttävätkö ne strategisia tavoitteita (sekä tuotetavoitteita että käyttäjätarpeita). Näiden kahden seikan lisäksi myös kolmas tulee esiin: kuinka toteutettavissa olevaa tämän tekeminen on? (Garrett 2011, 75.)

Kannattaa pitää silmällä ominaisuusehdotuksia, jotka näyttävät mahdollisia siirtymiä strategiassa ja jotka eivät olleet ilmeisiä visiodokumentin kehitysvaiheessa. Mikä tahansa ominaisuusehdotus, joka ei ole linjassa projektin strategian kanssa on määritelmältään laajuuden ulkopuolella. Jos huomataan, että yhä uudelleen palataan muokkaamaan strategiaa, on vaatimusten määrittely aloitettu liian aikaisin. (Garrett 2011, 76—77.)

Rakennetaso

Rakennetason ydinsisältöä ovat vuorovaikutussuunnittelu ja informaatioarkkitehtuuri. Kun vaatimukset ovat määritelty ja priorisoitu, syntyy selkeä kuva mitä lopulliseen tuotteeseen sisällytetään mukaan. Vaatimukset eivät kuitenkaan kuvaa miten palaset sopivat yhteen muodostaakseen yhtenäisen kokonaisuuden. Tämä on seuraava askel ylöspäin laajuudesta: käsitteellisen rakenteen kehittäminen sivustolle. (Garrett 2011, 78—79.)

Rakenteen maailma on kolmas viidestä tasosta, ja asianmukaisesti se on virstanpylväs sille, että käsiteltävät huolenaiheet muuttuvat strategian ja laajuuden abstrakteista asioista enemmän konkreettisempiin tekijöihin, jotka määrittävät mitä käyttäjät lopulta kokevat. Rajanveto abstraktin ja konkreettisen välillä voi olla kuitenkin häilyvä, vaikka paljon sillä, mitä tässä käsitellään, tulee olemaan huomattavaa, käsinkosketeltavaa vaikutusta lopputuotteeseen, tehdyt päätökset itsessään edelleen pitävät sisällään suurelta osin käsitteellisiä asioita. (Garrett 2011, 80.)

Perinteisessä ohjelmistokehityksessä rakenteellisen kokemuksen luomisen tiedonala on nimeltään vuorovaikutussuunnittelu. Sisältökehityksessä käyttökokemuksen rakenteen suunnittelussa on kyse informaatioarkkitehtuurista. Vuorovaikutussuunnittelu ja

informaatioarkkitehtuuri jakavat painotuksen määrittää kaavoja ja ketjuja, joissa käyttäjälle esitetään valintoja. Vuorovaikutussuunnittelu liittyy valintoihin, jotka sisältyvät tehtävien suorittamiseen ja valmiiksi saattamiseen. Informaatioarkkitehtuuri liittyy valintoihin, jotka koskevat tiedon välittämistä käyttäjälle. (Garrett 2011, 80—81.)

Ohjelmistokehittäjät ovat perinteisesti kallistuneet luomaan teknisesti tehokkaita tuotteita ilman huomioon ottamista käyttäjän tarpeista, erityisesti vanhoina aikoina, kun laskentateho oli rajallinen resurssi. Lähestymistapa, joka toimii teknologialle parhaiten ei lähes koskaan ole lähestymistapa, joka toimii parhaiten käyttäjälle. Tämän vuoksi ohjelmistot ovat olleet sellaisessa maineessa, että ne ovat monimutkaisia, sekavia ja vaikeita käyttää. Syntyi uusi tiedonala paremmin vastaamaan käyttäjien tarpeisiin. Se on nimeltään vuorovaikutussuunnittelu. (Garrett 2011, 82—83.)

Käyttäjien vaikutelmat siitä, miten vuorovaikutteiset komponentit, joita luodaan tulevat käyttäytymään, tunnetaan nimellä käsitteelliset mallit. Sivuston ominaisuudet voivat asioita, joita käyttäjä kuluttaa, paikkoja, joissa vieraillaan tai objekteja, joita käyttäjä hankkii. Esimerkiksi käsitteellinen malli tyypillisen verkkokaupan ostoskorille on säiliö. Säiliö pitää sisällään objekteja: tuloksena ”koriin laitetaan asioita” ja ”korista otetaan asioita”, ja järjestelmän täytyy tarjota funktiot näiden tehtävien saavuttamiseksi. Ostoskorimalli on verkkokauppaliiketoiminnassa nykyään niin yleinen asia, että se on omaksunut vakiintuneen käytännön roolin. Kun käytetään käsitteellisiä malleja, joita ihmiset tuntevat jo entuudestaan, syntyy tilanne, jossa vieraan verkkosivuston käyttö tuntuu tutulta. (Garrett 2011, 83.)

Informaatioarkkitehtuuri on uusi idea, mutta vanha tapa. Itse asiassa voisi sanoa, että se on yhtä vanha kuin ihmiskommunikointi itsessään. Niin pitkään kuin ihmisillä on ollut välitettävää informaatiota, on täytynyt tehdä päätöksiä siitä, miten se jäsennellään, jotta muut ymmärtävät sitä. Koska informaatioarkkitehtuuri huolehtii siitä, miten ihmiset kognitiivisesti käsittelevät informaatiota, informaatioarkkitehtuuripäätökset nousevat pintaan minkä tahansa tuotteen kanssa, joka vaatii käyttäjien ymmärtävän esitettyä informaatiota. (Garrett 2011, 88.)

Sisältöpainotteisilla sivustoilla informaatioarkkitehtuuri koskee sellaisten organisaatiollisten ja navigoinnillisten järjestelmien luomista, jotka sallivat liikkua sivustolla tehokkaasti. Informaatioarkkitehtuuri internetissä liittyy läheisesti tiedonhaun alaan: sellaisten järjestelmien suunnitteluun, jotka sallivat käyttäjän löytää tietoa helposti. Usein verkkosivustojen on tarkoitus tehdä paljon muutakin kuin vain auttaa ihmisiä löytämään asioita: monessa tapauksessa käyttäjiä täytyy kouluttaa, tiedottaa tai suostutella. (Garrett 2011, 89.)

Usein informaatioarkkitehtuuriongelmat vaativat sellaisten kategorisointijärjestelmien luomista, jotka vastaavat sivuston omien tavoitteiden, tyydytettävien käyttäjätarpeiden ja

sivustolle lisättävän sisällön kanssa. Sellaisten kategorisointijärjestelmien luominen voidaan hoitaa kahdella tavalla: ylhäältä alas tai alhaalta ylös katsoen. Informaatioarkkitehtuurin ylhäältä alas -lähestymistapa pitää sisällään arkkitehtuurin luomisen suoraan strategiatason seikkojen kautta: tuotetavoitteet ja käyttäjätarpeet. Aloittaen näiden strategisten tavoitteiden saavuttamiseksi tarvittavien mahdollisten sisällön ja toimintojen laajimmista luokista, jaamme sitten luokat loogisiin alaosiin. Tämä luokkien ja aliluokkien arvojärjestys palvelee tyhjänä kuorena, johon sisältö ja toiminnot sujautetaan. (Garrett 2011, 89.)

Informaatioarkkitehtuurin alhaalta ylös -lähestymistapa on myös johdettu luokista ja aliluokista, mutta se tekee sen pohjautuen analyysiin sisällöstä ja toiminnoista. Aloittaen olemassa olevasta lähdemateriaalista, asiat ryhmitetään yhteen matalan tason luokkiin ja sitten ryhmitetään nuo korkeamman tason luokkiin, kasaten kohti rakennetta, joka heijastelee tuotetavoitteita ja käyttäjätarpeita. Ylhäältä päin katsova arkkitehtuurillinen lähestymistapa toimii strategiatasosta katsoen, kun taas alhaalta ylöspäin katsova lähestymistapa toimii laajuustasosta katsoen. Asiaa lähestyminen ylhäältä alas voi joskus johtaa siihen, että tärkeät yksityiskohdat sisällöstä jättäytyvät huomaamatta. Toisaalta taas alhaalta ylöspäin -tapa voi joskus johtaa arkkitehtuuriin, joka on niin tarkkaan hiottu ja sovitettu olemassa olevaan sisältöön, että se ei ole joustava omaksumaan muutoksia ja lisäyksiä. On tärkeää löytää tasapaino. (Garrett 2011, 90.)

Ei ole tarpeen takertua tiettyyn lukumäärään luokkia millään arkkitehtuurin tasolla. Luokkien täytyy vain olla oikeita käyttäjien tarpeita ajatellen. Laadukkuuden tärkein merkki, ei ole se, kuinka monta vaihetta prosessi vei, vaan kävivätkö vaiheet järkeen käyttäjän kannalta. Käyttäjät poikkeuksetta puoltavat selkeästi määriteltyä seitsenvaiheista prosessia sekavan kolmeen vaiheeseen puristetun prosessin yli. (Garrett 2011, 91.)

Informaatiorakenteen perusyksikkö on solmukohta. Solmukohta voi liittyä mihin tahansa tiedonpalaseen tai tietoryhmään. Se voi olla niinkin pieni kuin yksittäinen numero, vaikkapa tuotteen hinta, tai niinkin laaja kuin kokonainen kirjasto. Tekemällä töitä solmukohtien parissa sivujen, dokumenttien tai komponenttien sijaan, voidaan ottaa käyttöön yhteinen kieli ja yleinen rakenteellisten konseptien kattaus monenlaisten ongelmien ratkaisuun. (Garrett 2011, 92.)

Solmukohdat informaatiorakenteessa laitetaan järjestykseen järjestäytymisperiaatteiden mukaan. Se on kriteeristö, jonka mukaan määritellään mitkä solmukohdat ryhmitetään ja mitkä pidetään erillään. Esimerkiksi yritysinformaatiosivustolla hierarkkisessa rakenteessa, jota joskus myös puuksi nimitetään, voisi olla sen huipulla luokkia kuten ”Kuluttaja”, ”Liiketoiminta” ja ”Sijoittaja”. Tällä tasolla järjestäytymisperiaate on kohdeyleisö, jolle sisältö on suunnattu. Toisen sivuston yläpuolen luokat voisivat olla ”Pohjois-Amerikka”, ”Eurooppa” ja

”Afrikka”. Maantiedon käyttäminen järjestäytymisperiaatteena on yksi lähestymistapa vastaamaan maailmanlaajuisen yleisön tarpeisiin. (Garrett 2011, 96.)

Käyttäjät eivät tule löytämään tietänsä eteenpäin organisaation suunnittelemassa arkkitehtuurissa, jos he eivät ymmärrä sivuston terminologiaa. On tärkeää käyttää loppukäyttäjien kieltä ja tehdä sitä johdonmukaisesti. Käyttäjätutkimus auttaa oikean, käyttäjille luonnollisen terminologian löytämisessä. (Garrett 2011, 98—99.)

Luurankotaso

Luurankotaso käsittää käyttöliittymä-, navigointi- ja informaatiosuunnittelun työtehtävät. Käsitteellinen rakenne alkaa muotoutua strategisista tavoitteistamme nouseville vaatimuksille. Luurankotasolla edelleen hiotaan tuota rakennetta tunnistaen tiettyjä käyttöliittymän, navigoinnin ja aineettomasta rakenteesta konkreettiseksi tekevän informaatiosuunnittelun näkökohtia. (Garrett 2011, 106—107.)

Rakennetaso edellisessä aliluvussa määrittää kuinka sivusto toimii; luurankotaso määrittää minkälaisen muodon tuo toiminnallisuus ottaa. Sen lisäksi, että käsitellään konkreettisempia esittelykysymyksiä, luurankotaso hoitaa asioita, jotka pitävät sisällään hienojakoisempia yksityiskohtia. Rakennetasolla tarkasteltiin isomman skaalan arkkitehtuuri- ja vuorovaikutusasioita; luurankotasolla huolenaiheita ovat pienemmän skaalan yksilölliset komponentit ja niiden suhteet toisiinsa. Jakolinjan toiminnallisuuspuolella luuranko määritetään käyttöliittymäsuunnittelun; nappien, kenttien ja muiden käyttöliittymäkomponenttien kautta. Informaatiotuotteilla on tyystin omat ongelmansa. Navigaatiosuunnittelu on käyttöliittymäsuunnittelun erikoistunut muoto, joka on räätälöity informaation esittämiseen. Lopuksi molemmat puolet kattaen, on olemassa informaatiosuunnittelu tiedon esittämiseen tehokasta kommunikaatiota varten. (Garrett 2011, 108.)

Nämä elementit ovat läheisesti toisiinsa kytkettyjä, enemmän kuin mikään muu elementti, jota tässä luvussa on esitelty. Ei ole epätavallista kohdata navigointisuunnittelun ongelmia, jotka alkavat hämärtyä informaatiosuunnittelun ongelmiksi tai törmätä kysymyksiin informaatiosuunnittelusta, jotka ovatkin käyttöliittymäsuunnittelun asioita. Vaikka rajanvedot voivat joskus hämärtyä, näiden tunnistaminen erillisinä asioina auttaa paremmin arvioimaan onko päädytty sopivaan ratkaisuun. Hyvä navigointisuunnittelu ei voi korjata huonoa informaatiosuunnittelua. Jos ei pystytä erottamaan ongelmatyyppejä, ei voida sanoa, että ne ovat todella tulleet ratkaistuiksi. Jos kyseessä on se, että tarjotaan käyttäjille mahdollisuus tehdä asioita, puhutaan käyttöliittymäsuunnittelusta. Jos kyseessä on se, että tarjotaan käyttäjille mahdollisuus mennä paikkoihin, puhutaan navigointisuunnittelusta. Jos kyseessä on se, että käyttäjälle viestitään ideoita, puhutaan informaatiosuunnittelusta. (Garrett 2011, 109.)

Vakiintuneet käytännöt ovat ihmisille tärkeitä, oli kyseessä sitten fyysinen tuote tai aineettomampi asia, kuten verkkosivusto. Vastauksen jokaiseen käyttöliittymäongelmaan ei täydy olla orjallisesti sitoutunut vakiintuneisiin käytäntöihin, mutta pitää muistaa olla varovainen siltä polulta poikkeamisessa ja poiketa käytännöistä vain, kun sillä on selkeä hyöty. Käyttöliittymästä kannattaa tehdä yhdenmukainen käyttäjille jo tuttujen muiden asioiden kanssa, mutta vielä tärkeämpää on tehdä käyttöliittymästä johdonmukainen itsensä kanssa. (Garrett 2011, 110—111.)

Jos sivustolla on kaksi ominaisuutta, joilla on samat käsitteelliset mallit, niillä on luultavasti samankaltaiset käyttöliittymävaatimukset. Käyttämällä samoja käytäntöjä molemmissa paikoissa sallii käyttäjän, joka on sinut toisen kanssa, nopeasti sopeutua toiseenkin. Joissain tapauksissa saatetaan olla taipuvaisia kaavoittaa jonkin erityisen toiminnon käyttöliittymäsuunnittelu oikean maailman käyttöliittymän mukaan. Kuitenkin useimmat käyttöliittymät ja navigoinnilliset laitteet ovat oikean elämän rajoitteiden tuotteita: fysiikat, ainemateriaalit ja niin edelleen. Näyttölaitepohjaiset verkkosivustot ja muut ohjelmistot sisältävät vähemmän samoja rajoitteita. (Garrett 2011, 112.)

Vastaavuuksien vetäminen sivuston ominaisuuksien ja käyttäjien oikean elämän kokemuksien välille voi vaikuttaa hyvältä tavalta antaa käyttäjille tietoa siitä, mistä nuo ominaisuudet kertovat. Kuitenkin tällainen lähestymistapa yleensä sumentaa ominaisuuden luonteen sen sijaan, että se tulisi esille. Mitä tuo pieni puhelimen kuvake sivustolla tarkoittaa? Salliiko se soittaa puhelun? Salliiko se tarkistaa ääniviestit? Voiko sen kautta maksaa puhelinlaskun? (Garrett 2011, 113.)

Käyttöliittymäsuunnittelun pääaiheita ovat oikeiden käyttöliittymäelementtien valitseminen käyttäjän tehtävien suorittamista varten sekä niiden järjestäminen näytöllä siten, että käyttäjä vaivatta ymmärtää ja helposti käyttää niitä. Tehtävät usein venyvät läpi useiden sivujen, kukin sisältäen eri käyttöliittymäelementit, joita käyttäjä käyttäjää. Rakennetason vuorovaikutussuunnittelussa on kyse siitä, mitkä toiminnot päätyvät millekin sivulle: käyttöliittymäsuunnittelun osa-aluetta on se, kuinka nuo toiminnot toteutetaan näytöllä. (Garrett 2011, 114.)

Onnistuneet käyttöliittymät ovat niitä, joissa käyttäjä huomaa heti tärkeät asiat. Merkityksettömät asiat taas jäävät huomaamatta tai niitä ei edes ole. Täytyy pohtia, mitä asioita käyttäjän ei tarvitse nähdä ja vähentää niiden näkyvyyttä. Tämä on suuri haaste. Ohjelmoijat ovat tottuneet ottamaan huomioon kaikista epätodennäköisimmät skenaariot, ohjelmistokehittäjien jargonissa ”reunatapaukset”. Ohjelmoijat pyrkivät luomaan sovelluksia, jotka eivät mene rikki. Tästä syystä ohjelmoijat ovat saaneet oppinsa kohdella jokaista tapausta samanarvoisesti, edusti se sitten yhtä tai tuhatta käyttäjää. Tämä lähestymistapa ei toimi

käyttöliittymäsuunnittelussa. Käyttöliittymä, joka antaa pienelle määrälle äärimmäisiä tapauksia saman arvon kuin valtaosan käyttäjien tarpeille on huono tekemään kummankaan yleisön tyytyväiseksi. Hyvin suunniteltu käyttöliittymä tunnistaa ne toimintatavat, joita käyttäjät luultavimmin tekevät, ja tekee niistä käyttöliittymäelementeistä helpoimpia käyttää. Jos ymmärrys käyttäjien tarpeista ja tavoitteista johtaa siihen ajatukseen, että useimmat heistä haluavat todella yksityiskohtaisia hakutuloksia, on silloin ”Näytä minulle enemmän tietoja” -valintaruudun jättäminen oletusarvoisesti päälle jotain, mikä johtaa tyytyväisempiin käyttäjiin riippumatta siitä lukivatko käyttäjät valintaruudun tekstin ja tekivätkö käyttäjät valinnan itse. (Garrett 2011, 114—115.)

Käyttöliittymäelementtejä on useita erilaisia. Valintaruudut sallivat käyttäjän tehdä valintoja, jotka ovat riippumattomia toisistaan. Radiopainikkeet sallivat käyttäjän valita yhden vaihtoehdon useista toisensa pois sulkevista vaihtoehdoista. Tekstikentät luonnollisesti sallivat käyttäjän syöttää tekstiä. Pudotusvalikot tarjoavat saman toiminnallisuuden kuin radiopainikkeet, mutta tekevät sen kompaktimmassa tilassa sallien monien vaihtoehtojen esittämisen tehokkaasti. Listalaatikot tarjoavat saman toiminnallisuuden kuin valintaruudut, mutta jälleen kompaktimmassa kokoluokassa, koska niitä voi vierittää ylös ja alas. Ne tukevat suuria määriä vaihtoehtoja. Painonappulat voivat tehdä paljon erilaisia asioita. Yleensä ne toimivat siten, että kaikki muu tieto, jota käyttäjä on sivulla syöttänyt, otetaan haltuun ja sitten sillä tehdään jokin toiminto. (Garrett 2011, 117.)

Navigointisuunnittelun täytyy saavuttaa kolme samanaikaista tavoitetta: ensin täytyy tarjota käyttäjälle keinot päästä yhdestä pisteestä toiseen sivustolla. Koska on yleensä epäkäytännöllistä linkittää jokainen sivu jokaiseen toiseen sivuun, navigointielementtien valinta täytyy tehdä sen perusteella, mikä helpottaa asiakkaan todellista käytöstä. Toiseksi navigointisuunnittelun täytyy viestiä siihen sisältyvien elementtien suhteista. Ei riitä, että vain tarjotaan linkkilista. Mitä yhteistä noilla linkeillä on? Ovatko toiset niistä tärkeämpiä kuin toiset? Mitkä ovat niiden olennaisimmat eroavaisuudet? Tämä viestintä on tärkeää, jotta käyttäjä ymmärtää mitä vaihtoehtoja hänellä on. Kolmanneksi navigointisuunnittelun täytyy viestiä sisältönsä ja käyttäjän nykyisin vierailemansa sivun suhteesta. Mitä tekemistä tällä on sen kanssa, mitä katson juuri nyt? Tämän viestiminen auttaa käyttäjää ymmärtämään, mikä saatavilla olevista vaihtoehdoista voisi parhaiten tukea tehtävää, jota he haluavat suorittaa. (Garrett 2011, 118—119.)

Useimmat sivustot tarjoavat monia navigointijärjestelmiä, kukin täyttäen erityisen roolin, jotta käyttäjä voi onnistuneesti navigoida sivustolla erilaisissa olosuhteissa. Globaali navigointi tarkoittaa käytännössä navigointipalkkia, joka usein näkyy jokaisella sivulla. Mihin tahansa halutaan mennä, sinne lopulta päätyy globaalin navigoinnin avulla. Paikallinen

navigointi tarjoaa käyttäjille pääsyn siihen, mitä heidän ”lähellään” arkkitehtuurissa on. Paikallinen navigointi voi tarjota käyttäjälle keinon siirtyä sivua ylemmälle tasolle, rinnakkaistasolle tai alitason sivulle. Täydentävä navigointi tarjoaa oikotiet aiheeseen liittyvään sisältöön, joka ei välttämättä ole helposti saavutettavissa edellä mainituilla navigointijärjestelmillä. Tämän tyyppinen järjestelmä sallii käyttäjän muuttaa fokusta sisällön tutkimisesta ilman, että tarvitsee aloittaa alusta. Kontekstillinen navigointi on sulautettuna sivun sisältöön itseensä. Tämä voi esimerkiksi olla hyperlinkki sivun tekstin joukossa. Kun käyttäjä lukee tekstiä, silloin on usein se tilanne, että he tarvitsevat lisätietoja. Mukavuusnavigointi tarjoaa pääsyn kohteisiin, joita käyttäjät eivät tarvitse säännöllisesti, mutta joita tavanomaisesti tarjotaan mukavuussyistä. Linkit yhteystietoihin, palautelomakkeisiin ja tietosuojakäytänneinformaatioon löytyvät usein kategorisoituna mukavuusnavigointiin. Jotkin navigointijärjestelmät eivät ole sulautettuna sivurakenteeseen, vaan toimivat itsenäisesti. Näitä kutsutaan etänavigoinniksi. Käyttäjät kääntyvät näiden puoleen, kun he turhautuvat muihin navigointijärjestelmiin, joita tarjotaan tai kun he ovat vilkaisseet niitä muita, mutta tulleet tulokseen, että heidän on parempi olla edes yrittämättä järkeillä niiden toimintaa. Sivustokartta on yleinen etänavigointikeino. Indeksi on aakkosjärjestyksessä toimiva toinen keino tehdä etänavigointia, jos sivusto käsittelee laaja-alaisesti eri aihealueita. Useissa tapauksissa sivukartta ja hyvin suunniteltu arkkitehtuuri riittävät. (Garrett 2011, 120—123.)

Informaatiosuunnittelu voi olla vaikea hahmottaa. Usein se toimii liimana, joka pitää muita suunnittelun komponentteja yhdessä. Joka tapauksessa informaatiosuunnittelu perustuu päätösten tekemiseen siitä, kuinka esittää informaatiota, jotta ihmiset voivat käyttää sitä ja ymmärtää sitä helpommin. Joskus informaatiosuunnittelu on visuaalista. Onko piirakkakaavio paras keino esittää dataa, vai olisiko pylväskaavio parempi käyttäjille? Välittääkö kiikarikuvake riittävästi sivustolta tiedon hakemisen konseptia, vai olisiko suurennuslasin kuvake paremmin ymmärretty? (Garrett 2011, 124.)

Informaatiosuunnittelulla on osansa käyttöliittymäsuunnittelun haasteissa, koska käyttöliittymän ei tarvitse ainoastaan kerätä informaatiota käyttäjältä, vaan myös viestiä informaatiota käyttäjälle. Virheilmoitukset ovat klassinen informaatiosuunnittelun haaste onnistuneiden käyttöliittymien suunnittelussa. Milloin vain järjestelmän täytyy antaa käyttäjälle informaatiota, jotta he voivat onnistuneesti käyttää käyttöliittymää, olipa syy sitten se, että he tekivät virheen tai vasta aloittivat käyttämisen, kyseessä on informaatiosuunnittelun haaste. (Garrett 2011, 126.)

Yksi tärkeä toiminto, jossa tietosuunnittelu ja navigointisuunnittelu toimivat yhdessä, on reitin etsimisen tukeminen. Siinä autetaan ihmistä ymmärtämään missä he ovat ja minne he voivat mennä. Reitinhaku on fyysisestä elämästä otettu idea, jota monet julkiset paikat

käyttävät. Verkkosivustoilla reitinhaku tyypillisesti sisältää sekä navigointisuunnittelua että informaatiosuunnittelua. Sivuston navigointijärjestelmien ei ainoastaan täydy tarjota pääsyä sivuston eri osiin vaan myös täytyy viestiä noita valintoja selkeästi. Hyvä reitinhaku sallii käyttäjän nopeasti luoda ajatuskartan siitä, missä he ovat, minne he voivat mennä ja mitkä valinnat auttavat heitä pääsemään tavoitteisiinsa. (Garrett 2011, 127.)

Sivuston asettelu on se, missä informaatio-, käyttöliittymä- ja navigointisuunnittelu yhdessä muodostavat yhtenäisen, koossapysyvän luurangon. Siinä on paljon tasapainoteltavaa. Siksi sivuasettelu katetaan yksityiskohtaisesti rautalankamalleissa. Rautalankamalli on karu kuvaus kaikista sivun komponenteista ja siitä, kuinka ne sopivat toistensa kanssa yhteen. (Garrett 2011, 128.)

Opinnäytetyön raportin luvun kolme aliluvuissa on havainnollistettu sovelluksen rautalankamalleja. Mallit sisältävät muun muassa tekstiä, painonappeja, hyperlinkkejä ja tekstikenttiä.

Rautalankamallit ovat välttämätön ensiaskel prosessissa, jossa muodollisesti näytetään toteen sivuston visuaalinen suunnittelu, mutta kaikki kehitysprosessiin kuuluvat tulevat käyttämään niitä ainakin jossain vaiheessa. Strategiasta, laajuudesta ja rakenteesta vastuussa olevat ihmiset voivat referoida rautalankamalleja vahvistaakseen lopputuotteen vastaavan heidän odotuksiaan. Varsinaisesti sivuston rakentamisesta vastuussa olevat ihmiset voivat referoida rautalankamalleja vastatakseen kysymykseen siitä, miten sivuston pitäisi toimia. (Garrett 2011, 129—130.)

Rautalankamallien arvo on siinä, miten ne integroivat rakennetason kaikki kolme elementtiä: käyttöliittymäsuunnittelun, käyttöliittymäelementtien valikoiman ja järjestelyn kautta; navigointisuunnittelun, navigoinnillisten järjestelmien tunnistuksen ja määrittelyn kautta; ja informaatiosuunnittelun, tietokomponenttien sijoittelun ja arvojärjestyksen kautta. Tuomalla kaikki kolme elementtiä yhteen yhdeksi dokumentiksi, rautalankamalli voi määrittää luurangon, joka rakentaa taustalla olevia käsitteellisiä rakenteita samalla osoittaen tietä eteenpäin kohti pintatason suunnittelua. (Garrett 2011, 131.)

Pintataso

Pintataso käsittelee aistisuunnittelua. Viisitasoisen mallin huipulla huomio siirtyy niihin tuotteen seikkoihin, joita käyttäjät ensin havaitsevat. Tässä kohtaa sisältö, toiminnallisuudet ja estetiikka tulevat yhteen tuottaakseen lopullisen kuosin, joka miellyttää aisteja ja samalla täyttää muiden neljän tason tavoitteet. (Garrett 2011, 132—133.)

Luurankotasolla käsiteltiin pääasiassa asioiden järjestyksen kanssa. Käyttöliittymäsuunnittelu hoitaa elementtien järjestykseen asettamista salliakseen sovelluksen kanssa tapahtuvan vuorovaikutuksen: navigointisuunnittelun eli elementtien järjestely sen sallimiseen, että

tuotteen läpi voidaan liikkua, ja informaatiosuunnittelun eli elementtien järjestely sen sallimiseen, että tietoa voidaan viestiä käyttäjälle. Kun siirrytään pintatasolle, aletaan olla tekemisissä aistisuunnittelun ja tuotteen luurangon tekevien loogisten järjestelyjen esittämisen kanssa. Esimerkiksi informaatiosuunnittelussa määritetään miten verkkosivun informaatioelementit pitäisi ryhmitellä ja järjestää. Visuaalisessa suunnittelussa määritetään miten tuo järjestys pitäisi esittää visuaalisesti. (Garrett 2011, 134.)

Aluksi voisi ajatella, että visuaalinen suunnittelu on puhtaasti estetiikkakysymys. Kaikilla on erilainen maku ja idea siitä, mikä käsittää visuaalisesti houkuttelevan suunnittelun, joten voisi luulla, että kaikki pohjautuu henkilökohtaisiin mieltymyksiin. Tämä ei kuitenkaan tarkoita, että suunnittelupäätösten pitäisi pohjautua siihen, mikä näyttää kaikkien mielestä hienolta. Huomion pitäisikin kiinnittyä siihen, kuinka hyvin visuaalinen suunnittelu toimii. Kuinka tehokkaasti suunnitelma tukee alemmilla tasoilla määritettyjä tavoitteita? Tekeekö tuotteen ulkonäkö esimerkiksi eroja arkkitehtuurin osien välillä epäselväksi, rakennetta heikentäen? Vai selventääkö visuaalinen suunnittelu käyttäjille tarjottavia vaihtoehtoja vahvistaen rakennetta? Brändi-identiteetin viestiminen esimerkiksi on yleinen verkkosivustojen strateginen tavoite. Brändi-identiteetti välittyy monella tapaa, käytetyssä kielessä tai sivuston toiminnallisuuden vuorovaikutussuunnittelussa, mutta pääasiallinen keino viestiä brändi-identiteettiä on visuaalinen suunnittelu. Jos identiteetti, jota halutaan viestiä, on tekninen ja arvovaltainen, sarjakuvamaisten fonttien ja kirkkaiden pastellisävyjen käyttö ei luultavasti ole oikea valinta. Kyseessä ei ole vain esteettinen valinta, vaan strateginen sellainen. Visuaalista suunnittelua voidaan arvioida kysymällä, minne silmä menee ensin? Mikä elementti ensin vetää puoleensa käyttäjän huomion? Kiinnittyikö huomio johonkin, joka on tuotteen strategisten tavoitteiden perusteella tärkeää? Vai onko ensimmäinen objekti, johon huomio kiinnittyi, häiriötekijä loppukäyttäjän tai organisaation tavoitteiden kannalta? (Garrett 2011, 136—137.)

Jos sivuston visuaalinen suunnittelu on menestyksekäs, kaava, jota käyttäjän silmät seuraavat, pitää sisällään kaksi tärkeää piirrettä: ensiksi silmät seuraavat sulavaa virtaa. Kun ihmiset sanovat, että suunnittelu on ”hektinen” tai ”sotkuinen”, he oikeasti reagoivat siihen, että visuaalinen suunnittelu ei johdattele heitä sulavasti sivuilla. Tämän sijaan heidän silmänsä pomppivat edestakaisin elementtien välillä, kun ne elementit kilpailevat käyttäjän huomiosta. Toiseksi visuaalinen suunnittelu antaa käyttäjille tietynlaisen opastetun kiertueen heille tarjottavien valintojen halki ilman, että ylikuormitetaan käyttäjä yksityiskohdilla. Kuten aina, noiden valintojen pitäisi tukea käyttäjien tavoitteita ja tehtäviä. Kenties vielä tärkeämpää on, että nuo vaihtoehdot eivät harhauta käyttäjää saamasta informaatiota tai funktioita, joita käyttäjät tarvitsevat noiden tavoitteiden täyttämiseen. (Garrett 2011, 138—139.)

Visuaalisessa suunnittelussa pääkeino käyttäjän huomion saamiseen on kontrasti. Suunnittelu ilman kontrastia nähdään harmaana, muodottomana massana, joka johtaa käyttäjän katseen ajelehtimiseen päämäärättömästi sivuilla. Kontrasti vetää huomiota puoleensa, korostaa navigointielementtejä sekä ensisijaisesti viestii käsitteellisiä ryhmiä informaatiosuunnittelussa. (Garrett 2011, 139.)

Tasaisuuden ylläpitäminen suunnittelussa on tärkeä asia siinä, että suunnittelu viestitään tehokkaasti ilman, että hämmennetään tai ylikuormitetaan käyttäjä. Elementtien, kuten vaikkapa painonappien, kokojen pitäminen yhdenmukaisina voi tehdä niiden uudelleen yhdistämisestä uuteen suunnitelmaan helpompaa. Ruudukkopohjainen sivuasettelu on yksi printtimedian tekniikka, joka siirtyy kätevästi myös internetin puolelle. Tämä lähestymistapa varmistaa suunnittelun tasaisuuden, kun käytetään mallia, joka sallii luoda asetteluvariaatioita. (Garrett 2011, 141.)

Koska verkkosivustot ovat usein rakennettu eristyksissä organisaation muusta suunnittelutyöstä, niitä on riivannut visuaalisen suunnittelun johdonmukaisuusongelmat. Nämä ongelmat ilmenevät kahdella tapaa: on olemassa sisäisen johdonmukaisuuden ongelmia, joissa tuotteen eri osat heijastelevat eri suunnittelulähestymistapoja. Sitten on myös olemassa ulkoisen johdonmukaisuuden ongelmat, joissa tuote ei heijastele samaa suunnitteluperiaatetta, joita käytetään saman organisaation muissa tuotteissa. (Garrett 2011, 143.)

Värit voivat olla yksi tehokkaimmista keinoista viestiä brändi-identiteettiä. Jotkut brändit ovat niin läheisesti yhdistetty väreihin, että on vaikea kuvitella niitä ilman värejä, jotka automaattisesti tulevat mieleen. Esimerkkejä voisivat olla vaikkapa Coca-Cola ja UPS. Nämä yritykset ovat käyttäneet tiettyjä värejä johdonmukaisesti vuosien varrella luodakseen vahvemman käsityksen identiteetistään yleisön mielissä. Useimmiten kirkkaammat värit ovat hyviä etualalla oleviin, ponnahtaviin, huomionhakuisiin elementteihin, kun taas taka-alalla olevat objektit usein käyttävät vaimeampia värejä edukseen. (Garrett 2011, 145.)

Joillekin yrityksille typografia, eli fonttien ja kirjasimien, käyttö erityisen visuaalisen tyylin saavuttamiseen on niin tärkeää heidän brändi-identiteetilleen, että he ovat ottaneet toimeksiannoksi käyttää tiettyjä kirjasimia erikseen heidän omaan käyttöönsä. Yritykset, kuten Apple ja Volkswagen, käyttävät mukautettuja typografioita luodakseen vahvemman vaikutelman brändi-identiteetistä viestinnässään. Usein ei tarvita kuin pari fonttia kaikkien viestintätarpeiden tyydyttämiseen. (Garrett 2011, 147.)

Elementit käytännössä

Käyttökokemuksen suunnittelu on vain erittäin suuri kokoelma erittäin pieniä ratkaistavia ongelmia. Ero menestyksekkään ja tuhoon tuomitun lähestymistavan välillä koostuu

kahdesta perusideasta: ensin täytyy ymmärtää mitä ongelmaa ratkotaan. Todetaan esimerkiksi, että iso, violetti nappula etusivulla on ongelma. Täytyykö nappulan kirkkaus ja violetti väri muuttua (pinta)? Vai onko nappula väärällä paikalla sivua (luuranko) vai onko funktio, jota nappi edustaa, jotakin ihan muuta kuin mitä käyttäjä odottaa (rakenne)? Toiseksi täytyy ymmärtää ongelman ratkaisun seuraamukset. Täytyy muistaa, että jokaisella tehdyllä päätöksellä on heijastusvaikutuksia läpi koko käyttökokemuselementtipinon. Navigointisuunnittelu, joka toimii hienosti yhdessä osassa ohjelmistoa, ei välttämättä vastaa toisten arkkitehtuuriosioiden vaatimuksiin. Joidenkin ohjelmiston osien vuorovaikutussuunnittelu voi olla innovatiivinen lähestymistapa, mutta täyttääkö se teknofobisten käyttäjien vaatimukset toisaalta? (Garrett 2011, 154.)

Voi tulla yllätyksenä, kuinka moni pikkuisista käyttökokemuksen määrittävistä päätöksistä ei ole ollenkaan tietoisia. Useimmiten käyttökokemuksesta tehdyt valinnat asettuvat yhteen seuraavista skenaarioista: ensimmäinen on oletusarvoinen suunnittelu. Se tapahtuu, kun käyttökokemuksen rakenne seuraava taustalla olevan teknologian tai organisaation rakennetta. Asiakkaiden tilaushistorian ja laskutustietojen pitäminen tietokannoissa erillään voi olla hyvä ratkaisu teknologisten järjestelmien kannalta, mutta se ei tarkoita sitä, että näiden asioiden pitäminen erillään on hyvä ratkaisu myös käyttökokemuksen kannalta. Voi olla, että yrityksen eri osastojen tuottama sisältö olisi hyvä pitää yhdistettynä. Toinen skenaario on jäljittelysuunnittelu. Se tapahtuu, kun käyttökokemus perääntyy takaisin muiden tuotteiden, julkaisujen tai ohjelmistojen tuttuihin käytäntöihin riippumatta siitä, kuinka sopivia nuo käytännöt saattavat olla käyttäjille (tai edes internetille itsessään). Kolmas skenaario on valtuutussuunnittelu. Se tapahtuu, kun käyttäjien tarpeiden sijaan henkilökohtaiset mieltymykset ajavat käyttökokemuspäätöksiä. Jos vaikka oranssi dominoi väripalettia, koska vanhempi varajohtaja pitää siitä, tai jos kaikki navigoinnilliset elementit ovat pudotusvalikoita, koska johtava teknikko tykkää niistä, on kadotettu näkyvistä strategiset tavoitteet ja käyttäjien näkemykset, joiden pitäisi olla syinä tehtyjen päätösten taustalla. (Garrett 2011, 156.)

Kannattaa vastustaa houkutusta sivuuttaa projektin perusteelliset käyttökokemusasiat ajan tai rahan säästämisen nimissä. Joskus on nähty käyttökokemuksen tulleen liimatuksi projektin päälle aivan projektin loppuvaiheilla, kauan aikaa sen jälkeen, kun aika näiden seikkojen ratkaisuun on jo loppunut. Juoksukilpailu sovelluksen julkaisuun ajattelematta voi vaikuttaa hyvältä idealta, kun alun perin julkaisupäivä oli lyöty lukkoon, mutta lopputulema luultavasti on se, että tuote kattaa tekniset vaatimukset, muttei vastaa käyttäjien tarpeita. Vielä pahempaa on se, että liimaamalla käyttökokemussuunnittelun toimet projektin päälle ihan lopuksi saattaa johtaa sellaisen tuotteen julkaisuun, jonka tiedetään olevan rikki, mutta jonka korjaamiseen ei ole enää tilaisuutta (tai rahaa). (Garrett 2011, 157—158.)

Kun katsotaan tuotetta ulkopuolelta, tai kun tullaan kehitysprosessiin mukaan ensimmäistä kertaa, on helppo keskittyä ilmiselvempiin elementteihin viisitasoisen mallin huipulla sen kustannuksella, että pohjimmaiset tasot jäävät huomiotta. Ironisesti kuitenkin elementit, jotka ovat vaikeimpia nähdä, strategia, laajuus ja rakenne, ovat ylipäätään suurimmassa roolissa käyttökokemuksen menestyksen tai epäonnistumisen kannalta. Usein epäonnistumiset ylemmillä tasoilla voi naamioida menestyksen alemmilla tasoilla. Ongelmat visuaalisessa suunnittelussa, liian sotkuinen sivuasettelu tai epäjohdonmukaiset tai ristiriidassa olevat värit, voivat olla käyttäjille niin epämiellyttäviä, että he eivät koskaan tulee kohtaamaan kaikkia nerokkaita valintoja, joita tehtiin navigointi- tai vuorovaikutussuunnittelussa. Huonosti laaditut lähestymistavat navigointisuunnittelussa voivat tehdä kaiken järkevän, joustavan informaatioarkkitehtuurin parissa vietetyn työn vaikuttamaan ajanhukalta. Vastaavasti kaikkien oikeiden päätösten tekeminen ylemmillä tasoilla ei merkitse mitään, jos nuo päätökset perustuvat huonoihin valintoihin alemmilla tasoilla. (Garrett 2011, 162.)

6.2 Tutkimuksen suorittaminen ja tulosten esittely ja pohdinta

Ohjelmisto-opinnäytetyön yhteydessä tehdään kvantitatiivinen tutkimus, joka liittyy ratkaisun onnistumisen arviointiin. Tutkimuksessa selvitetään toteutetun sovelluksen käyttökokemusta. Opinnäytetyön aliluku 6.2 koostuu seuraavista asioista:

- tutkimuksen suorittaminen kuvattuna tarkasti - menetelmät ja analyysin kuvaus tulosten kanssa
- lomakkeen kysymykset ja muuttujat perustellaan esitetyn tietopohjan avulla
- tutkimustulosten esittely kuvioina ja taulukoina
- tulosten esittelyn loppuun pohdinta siitä, mitä tuloksista tulisi ymmärtää

Tutkimusongelma on lähtökohta, johon tutkimus perustuu. Se on esimerkiksi kysymys, johon tutkimuksella haetaan vastauksia. Tutkimusongelman selvittämistä autetaan ongelmanasettelulla. Asettelu tarkoittaa ongelman rajaamista, hahmottelua ja muotoilua, joiden avulla ongelma tullaan ratkaisemaan. Tutkimusongelmaa pohtiessa on myös tärkeää selvittää, mitä tutkiessa on tavoitteena tuottaa. Se, mihin halutaan pyrkiä auttaa tutkimuskysymysten muodostamisessa. Oikein muotoillut kysymykset auttavat rajaamaan saatavaa tietoa. (Jyväskylän yliopisto 2021.) Tutkimusongelma on ”miten toteutetaan tehokas tukikysymyssivusto Avenla Oy:lle?”.

Tutkimusongelman määrittämisen jälkeen etsitään lähteitä apuna käyttäen tietoa aiheesta. Tutkimuksen tietopohja/kirjallisuuskatsaus laadittiin jo raportin edellisessä luvussa. Tämä

sallii tutustua asioihin, joita toiset tutkijat ovat nostaneet esille. Näin saadaan ajantasaista ja olennaista tietoa aiheesta.

Kohdeilmiö tarkoittaa konkreettista ilmiötä, mitä tutkimus tarkastelee. Kohdeilmiö on ”arvon tuottaminen yritysmaailmaan laadukkaan ohjelmistotuotteen kautta”.

Hypoteesilla tarkoitetaan ennakoitua ratkaisua tai selitystä tutkittavaan ongelmaan, jotka yleensä esitetään väitteen muodossa. Perusteet hypoteesien asettamiselle löytyvät mahdollisesti teoreettisista malleista, teoriasta tai aihetta käsittelevistä aiemmista tutkimuksista. Isossa osassa on myös tutkijan tai tutkijoiden oma mielikuvitus, sekä keskustelut muiden kanssa. Tutkimuksen hypoteesi on ”Garrettin viisiportaiseen malliin nojaten opinnäytetyön artefakti menestyksekkäästi vastaa Avenla Oy:n tarpeisiin saada hyödyllinen verkkosivusto”.

Tutkimuksen ainoa taustamuuttuja on Avenla Oy:n ohjelmistokehittäjän työkokemus alalla vuosina. Tutkimuksen aiheesta ja luonteesta johtuen muiden taustamuuttujien käyttäminen ei ole mielekästä. Keräämällä tietoa työkokemuksesta vuosina voidaan tuloksia analysoida siten, että nähdään kuinka kyselyn vastaukset eroavat mahdollisten junior- ja senior-tason ohjelmistokehittäjien välillä. Junior-tason ohjelmoijan käsitykset ohjelmistojen käyttökokemuksen laadukkuudesta yleisesti voivat olla hyvinkin erilaiset verrattuna kokeneemman kollegansa näkemyksiin, jotka ovat hioutuneet vuosien työkokemuksen saatossa.

Tutkimuksen mitattavia muuttujia ovat

- Avenla Oy:n erityisvaatimukset sovellukselle
- Sovelluksen käytettävyys
- Suunnitteluprosessin laadukkuus erityisesti rautalankamallien osalta
- Käyttökokemuksen rakennetason asiat
- Käyttökokemuksen luurankotason asiat
- Käyttökokemuksen pintatason asiat

Mitattava muuttuja ”Avenla Oy:n erityisvaatimukset artefaktille” antaa tilaisuuden selvittää, toteutuivatko kaikista keskeisimmät perusvaatimukset artefaktissa ja näin ollen voidaanko artefaktia Avenla Oy:n mielestä pitää menestyksekkäänä. ”Sovelluksen käytettävyys” on käyttökokemusta arvioitaessa tärkeä mittauskeino, sillä se kertoo saavuttaako loppukäyttäjä tavoitteensa ollessaan tekemisissä artefaktin kanssa. ”Suunnitteluprosessin laadukkuus rautalankamallien osalta” -muuttuja on olennainen osa ammattimaista ohjelmistokehitystä, sillä huolellisesti suunniteltu ohjelmisto varmistaa, että artefakti toimii ja näyttää

hyvältä. Kolme viimeistä mitattavaa muuttujaa liittyvät vahvasti J. J. Garrettin viisiportaiseen käyttökokemusmalliin. Garrettin kirja on alansa arvostetuimpia teoksia.

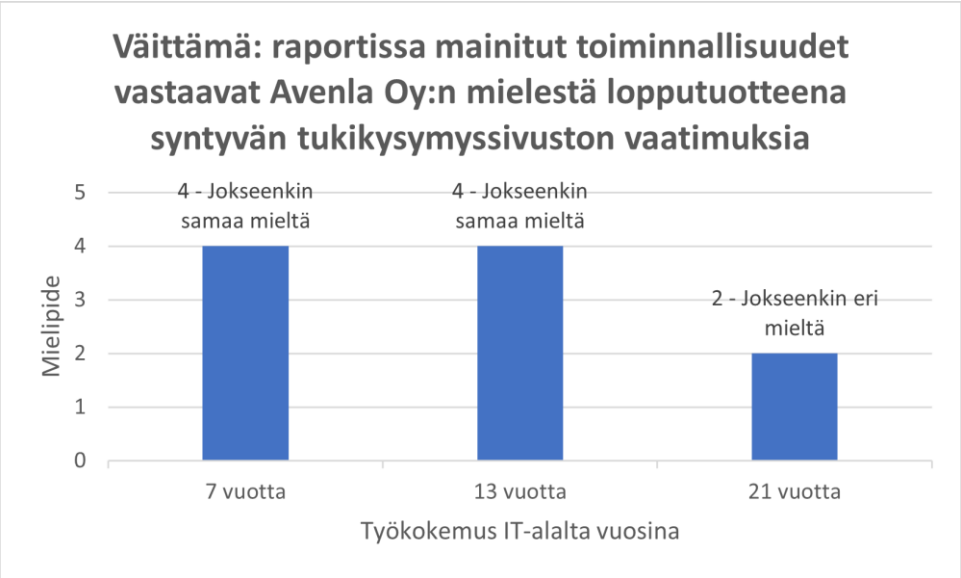
Aiemmin määritettyjen taustamuuttujien ja mitattavien muuttujien arvoja mitataan asteikollisilla kysymyksillä eli niin kutsutuilla Likert-asteikoilla. Niissä on joukko asenneväittämiä, jotka ilmaisevat sekä kielteistä että myönteistä suhtautumista asiaan. Vastausvaihtoehdot ovat:

1. Täysin eri mieltä
2. Jokseenkin eri mieltä
3. Ei samaa eikä eri mieltä
4. Jokseenkin samaa mieltä
5. Täysin samaa mieltä

Näiden lisäksi asteikossa käytetään vaihtoehtoa ”en osaa sanoa” (EOS). Se ei ole keskimäinen, neutraali vastausvaihtoehto, vaan sillä vastaaja voi ohittaa kysymyksen ottamatta siihen kantaa esimerkiksi siinä tapauksessa, jossa vastaaja kokee, ettei tiedä asiasta riittävästi. EOS sijoitetaan Likert-asteikon viimeiseksi vaihtoehdoksi. Kysymyksiin ei voi vastata "kyllä" tai "ei". Tämän seurauksena kysymykset eivät sisällä sanoja "onko", "ovatko", "tekeekö" tai "tekevätkö". Kysymystyypit määritellään oppilaitoksen Webropol-ohjelmalla tarkoituksenmukaisesti. Kyselyssä käytetään myös avoimia kysymyksiä lisätietojen saamiseksi. Kyselyyn muotoillaan avoin kysymys, johon vastaajat voivat kirjoittaa vastauksensa vapaasti. Kysely määritellään anonyymiksi.

Tutkimus tulee olemaan kokonaistutkimus, jolla saadaan käsitys koko perusjoukosta. Aikataulusyistä kysely toteutetaan vain niille Avenla Oy:n ohjelmistokehittäjille, jotka tulevat olemaan jatkossa tekemisissä opinnäytetyön artefaktin kanssa. Avenla Oy:n kansainvälisten asiakkaiden näkemyksiä toteutetun artefaktin tehokkuudesta ei opinnäytetyön aikana tulla mittaamaan aikataulusyistä.

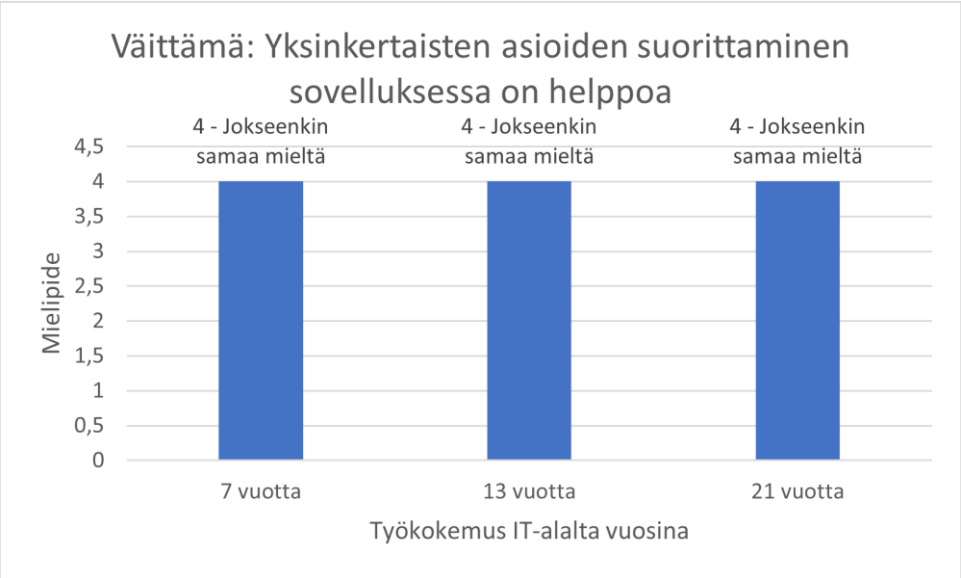
Myös tulosten esittely käydään läpi. Kuvioita ja taulukoita käytetään tarkoituksenmukaisesti niin, että verrataan tuloksia taustamuuttujan (ristiintaulukointi) suhteen esimerkiksi vastausten jakautumisesta vastaajan työkokemuksen määrän suhteen. Analyysissa käytetään myös erilaisia tunnuslukuja, joiden pohjalta tehdään johtopäätöksiä ratkaisun menestyksen arvioimisessa. Tunnusluvut antavat tietoa tietojen keskipainotteisuudesta sekä vaihtelevuudesta. Myös avoimet vastaukset analysoidaan.



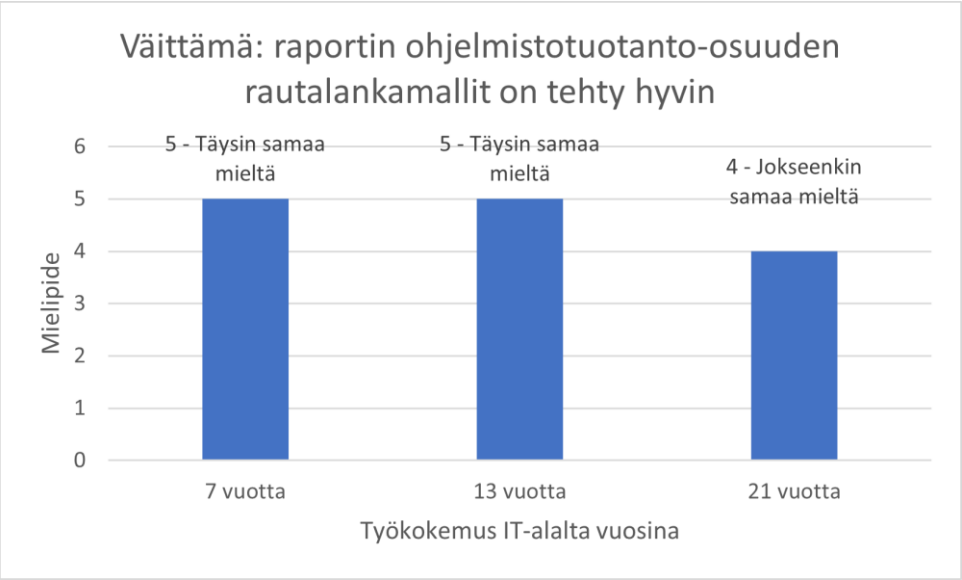
Kuvio 20. Kyselyn tulokset raportin ja artefaktin suhteesta toisiinsa.

Kyselyn toinen kysymys tarjosi vastaajille antaa vapaamuotoisen kirjoitelman, kun heiltä kysyttiin näkemystä siihen, miten tukikysymyssivusto auttaa Avenla Oy:tä ja alaa yleensäkin? Eräs vastaus kuului näin:

> *Tukikysymyssivusto voisi viimeisteltynä olla toimiva työkalu Avenlan asiakaspalveluprosessien tehostamiseen. Haluttaessa sivuston sisältö voitaisiin julkaista täysin avoimesti internetiin, jolloin sen kautta generoituva sisältö voitaisiin valjastaa myös SEO tarkoituksiin. Yleisemmin alalle koituvia hyötyjä on vaikea nähdä, koska tukikysymyssivusto koskisi tässä tapauksessa Avenlan omaa tuotetta, eikä tieto ole kovin hyödyllistä muille kuin tuotteen käyttäjille.*

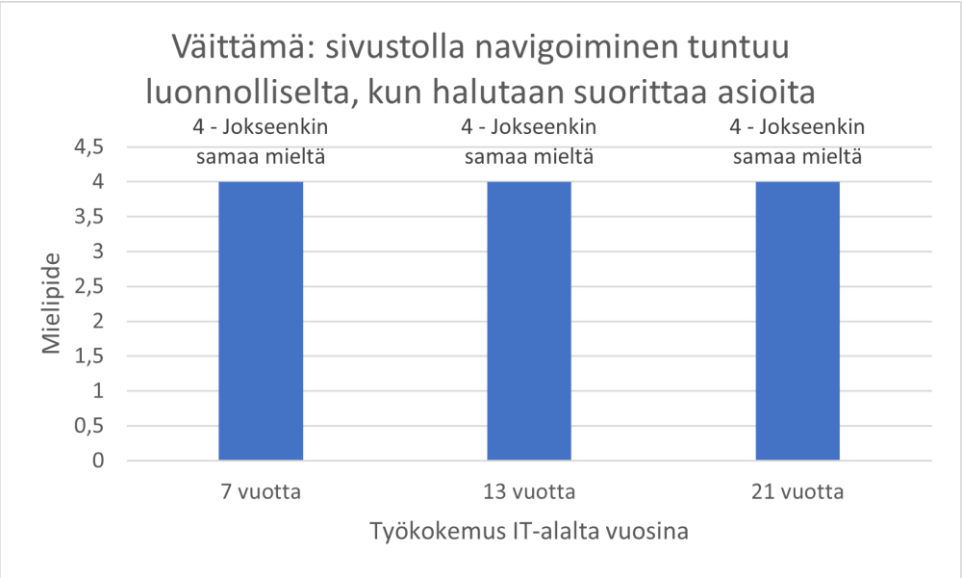


Kuvio 21. Kyselyn tulokset artefaktin käytön helppoudesta.

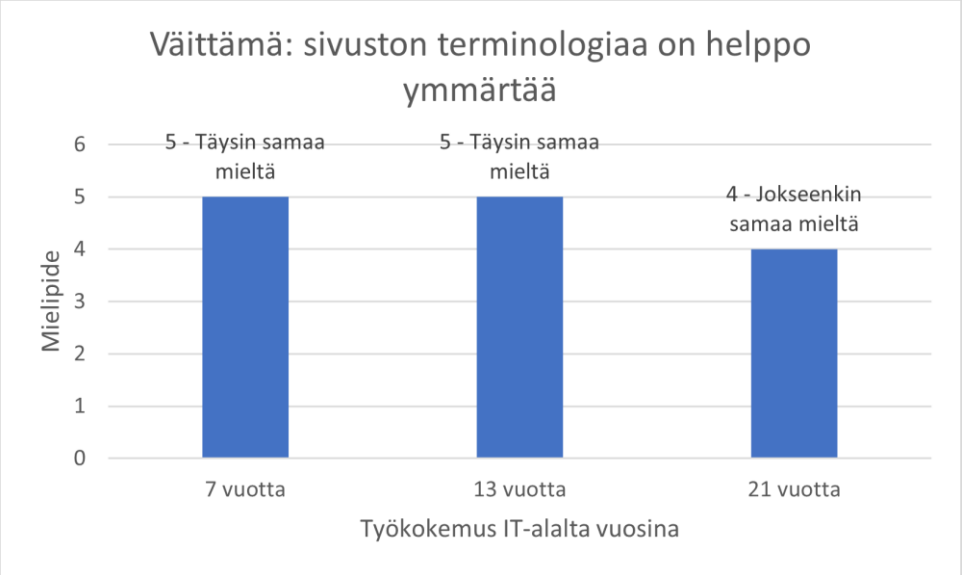


Kuvio 22. Kyselyn tulokset raportin rautalankmallien laadukkuudesta.

Kuvio 23. Kyselyn tulokset painonappien sijainnista artefaktissa.

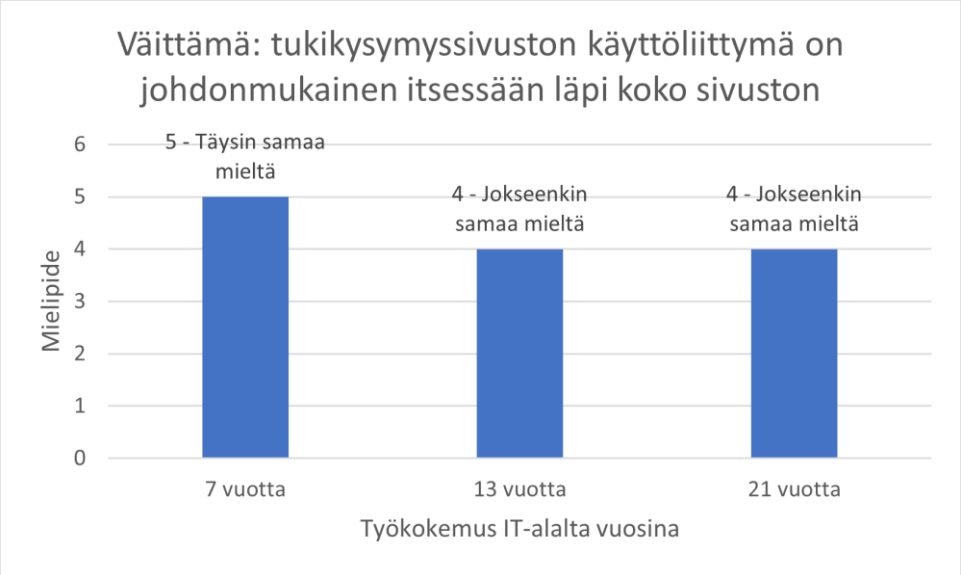


Kuvio 24. Kyselyn tulokset navigoinnin sujuvuudesta.

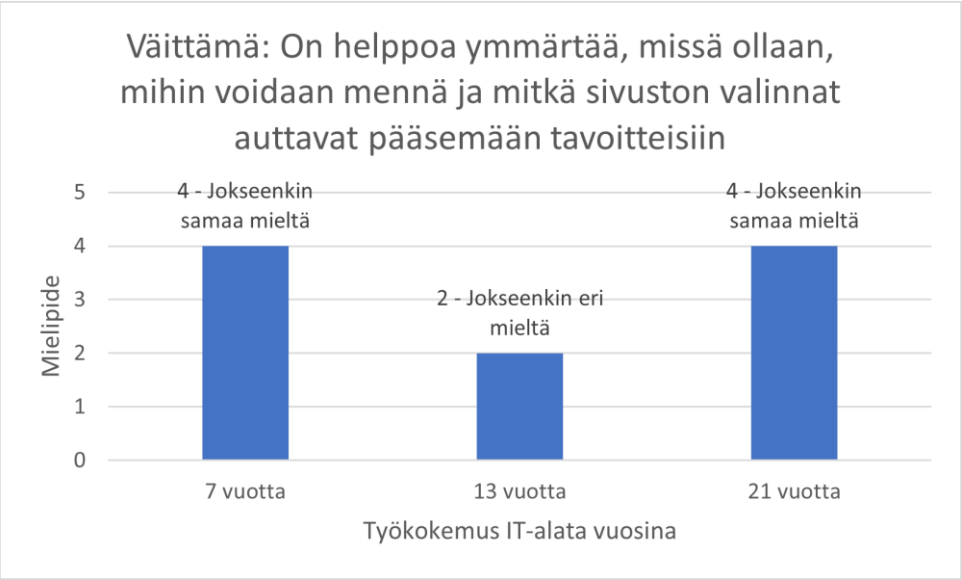


Kuvio 25. Kyselyn tulokset sivuston terminologian käytön osalta.

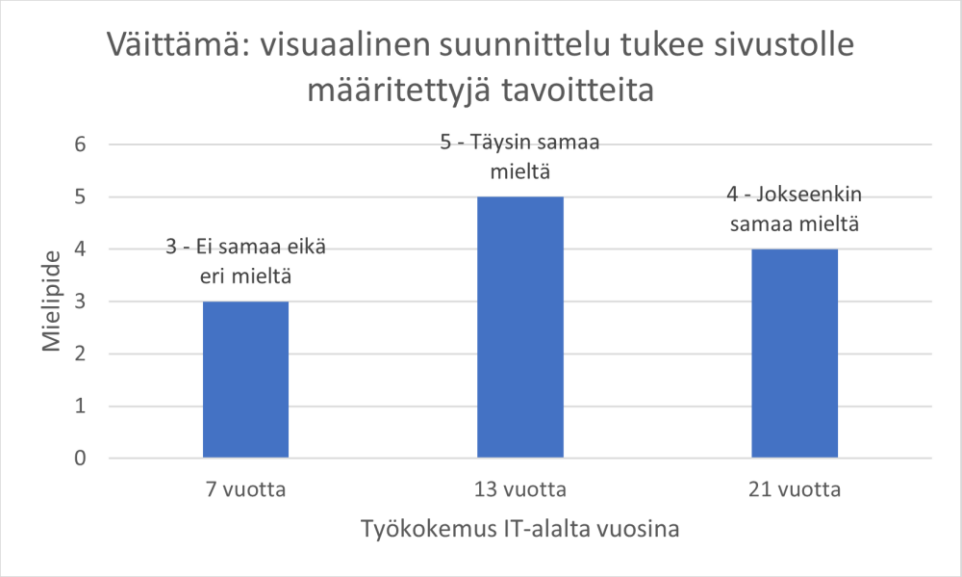
Kuvio 26. Kyselyn tulokset käyttöliittymästä muihin sivustoihin verrattuna.



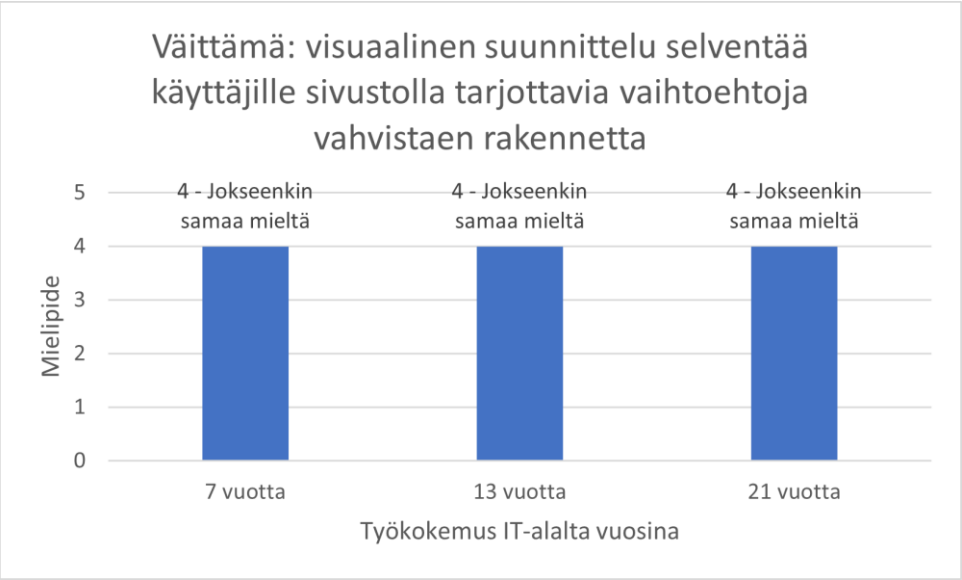
Kuvio 27. Kyselyn tulokset käyttöliittymästä sovelluksen sisäisesti.



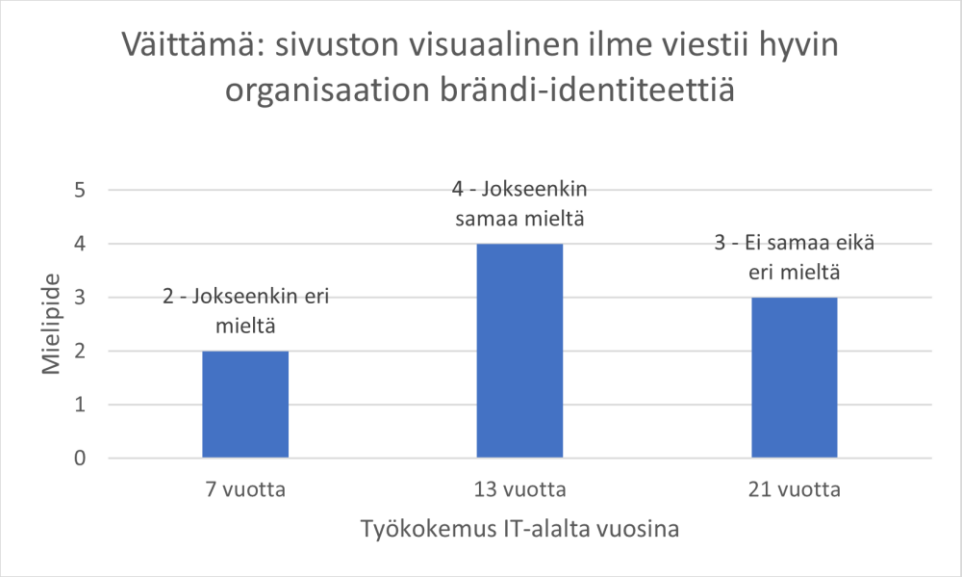
Kuvio 28. Kyselyn tulokset sivuston peruslogiikan hahmottamisesta.



Kuvio 29. Kyselyn tulokset visuaalisen suunnittelun ja tavoitteiden suhteen.



Kuvio 30. Kyselyn tulokset visuaalisen suunnittelun selkeydestä.



Kuvio 31. Kyselyn tulokset sivuston brändi-identiteetin suhteen.

Tunnusluvut	
Keskiarvo	3,9
Keskivirhe	0,1
Mediaani	4
Moodi	4
Keskihajonta	0,8
Otosvarianssi	0,7
Kurtosis	1,1
Vinous	-1,1
Alue	3
Minimi	2
Maksimi	5
Summa	141
Lukumäärä	36

Alkuperäinen toive kvantitatiiviselle tutkimukselle oli saada noin kymmenkunta tai enemmän vastauksia Webropol-kyselyyn. Näin olisi saatu melko kattavasti dataa siitä, minkälaisen käyttökokemuksen artefakti tuotti Avenla Oy:lle. Toimeksiantaja ei kuitenkaan ole suuryritys ja on lisäksi jakautunut eri kehitysalustoilla, kuten PHP:lla ja .NET:lla työskenteleviin tiimeihin, joten opinnäytetyön tutkimuksen tapauksessa vastauksien lukumäärä oli melko rajallinen kolme kappaletta. Tästä huolimatta tutkimuksella saatiin kohtuullisen hyvin mitattua artefaktin menestystä käytettävyyden osa-alueella, vaikka lisädataa olisi suotavaa kerätä esimerkiksi altistamalla kysely Avenla Oy:n kansainvälisille asiakkaillekin.

Huomionarvoinen seikka lienee se, että kyselyssä Likert-kysymysten ”En osaa sanoa” -vastausvaihtoehdon kohdalla ei ollut yhtään vastausta. Tästä voinee päätellä, että kysymyksenasettelu tutkimuksessa oli siinä määrin olennainen, että kaikkien kysymysten kohdalla vastaajat osasivat arvioida mielipiteitään, eikä kenenkään tarvinnut jättää mielipidettään ilmaisematta.

Avoin vastaus -tyyppinen kysymys alleviivasi sivuston tarvitsevan vielä viimeistelytyötä, mutta vastaaja näki sovelluksessa potentiaalia tehostaa Avenla Oy:n asiakaspalveluprosessia. Sovelluksen hyötyjä IT-alalle yleensäkin vastaaja ei tunnistanut, mikä melko lailla pitää paikkansa, sillä kyseinen artefakti palvelee käytännössä vain suljettua ryhmää ihmisiä: Avenla Oy:tä ja sen asiakkaita. Vastaaja pohdiskeli myös sovelluksen mahdollista näkyvyyttää hakukoneissa, mikä toisi kaikille osapuolille huomattavia etuja.

Kysely toi esiin, että käytettävyysmittauksissa eniten Avenla Oy:n vastaajat pitivät raportissa aiemmin esitellyistä sovelluksen käyttöliittymien rautalankamalleista. Myös se, että sivuston terminologiaa oli helppo ymmärtää, keräsi kannatusta taakseen. Puolestaan erimielisyyttä kyselyssä nähtiin siinä, miten hyvin sivuston visuaalinen ilme viesti Avenla Oy:n brändi-identiteettiä. Tämä olisi eräs kehittämisen kohde. Keskimäärin tuloksista on

havaittavissa eroja vastauksissa sen mukaan, kuinka kokeneelta alan ammattilaiselta vastauksia kerättiin. Kyselyn vastaaja, jolla oli työvuosissa mitattuna vähiten kokemusta, keskimäärin arvioi sovelluksen käytettävyyttä parempilaatuiseksi kuin kokeneemmat kollegansa. Luultavasti pitkäaikainen työura IT-alalla tuo mukanaan kriittisemmän näkökulman opinnäytetyön kaltaisiin sovelluksiin, ja ylipäätään työote on vaativampi laadun ja viimeistelyn suhteen.

Tunnuslukujen keskiarvo kertoo, että vastaanotto opinnäytetyön artefaktille oli varovaisen positiivinen. Mediaanikin oli aavistuksen keskiarvoa korkeampi eli sinänsä käytettävyys kokemuksena vaikuttaa olleen pääosin myönteinen. Lisäksi on positiivista, että mikään kyselyssä käytettävyyttä mitannut kysymysosa-alue ei tuottanut vastaukseksi keneltäkään täysin eri mieltä -vastausvaihtoehtoa. ”Täysin samaa mieltä” eli parasta vastausvaihtoehtoa tarjottiin kyselyn tuloksissa sen sijaan useampaankin otteeseen eri mittauskohdissa.

7 Yhteenveto

Opinnäytetyön tarkoituksena tiivistetysti oli virtaviivaistaa toimeksiantajan ohjelmistoasiakkaille annettavaa käytön tukea tukikysymyssivuston kautta. Sovelluksen perusperiaate on tarjota kootusti yhdessä paikassa asiakkaille mahdollisuuden kysyä tukikysymyksiä ostamiensa ohjelmistojen käyttöön liittyen ja saada niihin vastauksia. Toimeksiantajan asiakastuesta aiheutuva kuormitus helpottuu, kun yhteen paikkaan koottu tieto Wordpress Multisite Shared Media -ohjelmiston käytöstä ja asentamisesta vähentää tarvetta vastailla toistuvasti asiakkaille yksitellen sähköpostitse tukitilanteissa. Opinnäytetyössä käytiin läpi ohjelmistotuotannon eri vaiheet perustuksista lähtien valmiiseen sovellukseen saakka. Kvantitatiivisen tutkimuksen teoriapohja luodaan Jesse James Garrettin teokseen The elements of user experience viitaten. Sen rakentamassa porrastetussa mallissa käydään läpi käyttökokemuksen rakentamista painottaen web-sivustojen roolia esimerkkinä pohdinnalle. Garrettin malliin nojaten opinnäytetyössä suoritettiin kvantitatiivinen tutkimus artefaktin menestyksestä käytettävyyden osa-alueella. Kyselyn tulosten valossa artefaktia voisi varovaisesti nimittää menestykseksi. Sovellus sai keskimäärin vastauksista kerättyä varovaisen positiivista palautetta, ja vaikka työsarkaa sovelluksen hiomisessa huippuunsa riittää, melko hyvät perustukset on jo luotu.

Tutkimuksen pohjalta voitaneen sanoa, että tulokset olivat melko hyvin linjassaan odotusten kanssa. Missään nimessä kyseessä ei ole ääripäätapaus, vaan opinnäytetyön artefakti menestyi käytettävyysmittauksissa keskimäärin varovaisen positiivisesti. Sovellusta on mahdollista parantaa useilla eri tavoilla. Alla on listattuna joitakin olennaisimpia poimintoja kerätyn palautteen pohjalta:

- Ehdottavan hakujärjestelmän pitäisi etsiä samankaltaisia, olemassaolevia kysymyksiä myös muista annetuista tiedoista kuin vain otsikosta
- Kun jokin ei toimi, käyttäjälle pitäisi näyttää virheilmoitus
- Ei mahdollista lisätä olemassaoleva avainsana (bugi)
- Angularin Observable-bugi ei salli sivun uudelleenlataamista, sivustolla täytyy aina navigoida etusivun kautta
- Hakutuloslistaus ei näytä runkotekstiä, päivämäärää ja vastausten lukumäärää
- Päivämäärät eivät ole muotoiltu suomalaisen standardin mukaisesti
- Sivuasettelu ei toimi mobiililaitteella

Näiden parannusten implementoimisen jälkeen olisi mahdollista suorittaa käytettävyystutkimus uudelleen ja verrata tuloksia keskenään. Tuloksista voisi sitten tehdä johtopäätöksiä sovelluksen valmiudesta siirtyä jopa ihan tuotantokäyttöön ja nähtäisiin, oliko sovelluksessa parannettavaa ja kuinka paljon. Tässä vaiheessa käytettävyystutkimuksen kohderyhmän voisi laajentaa koskemaan suurempaa joukkoa ihmisiä, esimerkiksi ottamalla mukaan Avenla Oy:n kansainvälisiä asiakkaita. Näin saataisiin hienojakoisempaa ja kattavampaa dataa sovelluksen menestyksestä tarkoituksessaan.

Lähteet

Angular. 2021. What is Angular? Google. Viitattu 12.3.2021. Saatavissa: https://angular.io/guide/what-is-angular

Angular. 2022a. Add navigation with routing. Google. Viitattu 4.4.2022. Saatavissa: https://angular.io/tutorial/toh-pt5

Angular. 2022b. Add services – Observable data. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/tutorial/toh-pt4#observable-data

Angular. 2022c. Async pipe. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/observables-in-angular#async-pipe

Angular. 2022d. Building a template-driven form. Google. Viitattu 4.4.2022. Saatavissa: https://angular.io/guide/forms

Angular. 2022e. CLI overview and command reference. Google. Viitattu 4.4.2022. Saatavissa: https://angular.io/cli

Angular. 2022f. Communicating with backend services using HTTP. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/http

Angular. 2022g. Dependency injection in Angular. Google. Viitattu 4.4.2022. Saatavissa: https://angular.io/guide/dependency-injection

Angular. 2022h. Introduction to Angular concepts. Google. Viitattu 10.6.2022. Saatavissa: https://angular.io/guide/architecture

Angular. 2022i. Introduction to services and dependency injection. Google. Viitattu 6.4.2022. Saatavissa: https://angular.io/guide/architecture-services

Angular. 2022j. Listing items with NgFor. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/built-in-directives#listing-items-with-ngfor

Angular. 2022k. Observables compared to other techniques. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/comparing-observables

Angular. 2022l. Property binding. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/property-binding

Angular. 2022m. Requesting data from a server. Google. Viitattu 5.4.2022. Saatavissa: https://angular.io/guide/http#requesting-data-from-a-server

Angular. 2022n. Routes. Google. Viitattu 4.4.2022. Saatavissa: https://angular.io/tutorial/toh-pt5#routes

Angular. 2022o. Search by name. Google. Viitattu 2.6.2022. Saatavissa: https://angular.io/tutorial/toh-pt6#search-by-name

Angular. 2022p. Validating form input. Google. Viitattu 15.5.2022. Saatavissa: https://angular.io/guide/form-validation

Bootstrap. 2022. Grid system. Getbootstrap.com. Viitattu 5.4.2022. Saatavissa: https://getbootstrap.com/docs/5.1/layout/grid/

Cavanaugh, R. 2012. TypeScript's hashtables and keys with a dash. Stack Overflow. Viitattu 5.4.2022. Saatavissa: https://stackoverflow.com/a/13663007/5964318

Cloudflare. 2022. How CAPTCHAs work | What does CAPTCHA mean? Cloudflare, Inc. Viitattu 17.5.2022. Saatavissa: https://www.cloudflare.com/learning/bots/how-captchas-work/

Cuelogic. 2014. How useful are web application frameworks? & How do I know which framework would suit me? Viitattu 12.3.2021. Saatavissa: https://www.cuelogic.com/blog/how-useful-are-web-application-frameworks-how-do-i-know-which-framework-would-suit-me

Freeman, J. 2019. What is JSON? A better format for data exchange. Infoworld. Viitattu 12.3.2021. Saatavissa: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html

Garrett, J. 2011. The Elements of User Experience. 2. uudistettu painos. Kalifornia: New Riders.

Gillis, A. 2020. REST API (RESTful API). SearchApp Architecture. Viitattu 12.3.2021. Saatavissa: https://searchapparchitecture.techtarget.com/definition/RESTful-API

Google. 2021. Architecture overview. Google. Viitattu 4.8.2021. Saatavissa: https://angular.io/guide/architecture

Hurja. 2021. UX- ja UI-suunnittelu – mitä ne ovat ja mikä rooli niillä on verkkosivu- ja ohjelmistoprojektissa? Hurja Solutions Oy. Viitattu 20.6.2022. Saatavissa: https://www.hurja.fi/blogi/ux-ja-ui-suunnittelu-mita-ne-ovat/

Jain, P. 2016. Difference between Constructor and ngOnInit. Stack Overflow. Viitattu 5.4.2022. Saatavissa: https://stackoverflow.com/a/35763811/5964318

Jyväskylän yliopisto. 2021. Tutkimusprosessi. Viitattu 15.12.2022. Saatavissa: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/tutkimusprosessi

Kauppalehti. 2023. Avenla Oy | Yritys ja taloustiedot. Viitattu 28.3.2023. Saatavissa: https://www.kauppalehti.fi/yritykset/yritys/avenla+oy/19771067

Kinsta. 2021. What Is Wordpress? Explained for Beginners. Viitattu 12.3.2021. Saatavissa: https://kinsta.com/knowledgebase/what-is-wordpress/

Lucidchart. 2022. What is a swim lane diagram. Lucid Software Inc. Viitattu 2.6.2022. Saatavissa: https://www.lucidchart.com/pages/tutorial/swimlane-diagram

Mulesoft. 2021. What is an API? (Application Programming Interface). Viitattu 12.3.2021. Saatavissa: https://www.mulesoft.com/resources/api/what-is-an-api

Mozilla Developer Network. 2022. Template literals (Template strings). Mozilla Corporation. Viitattu 5.4.2022. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Ross, T. 2018. WordPress REST API post orderby custom value. Tim Ross Web Development. Viitattu 5.4.2022. Saatavissa: https://www.timrosswebdevelopment.com/wordpress-rest-api-post-order/

TechTerms. 2013. Framework. Sharpened Productions. Viitattu 12.3.2021. Saatavissa: https://techterms.com/definition/framework

Wordpress. 2022a. Authentication. Wordpress.org. Viitattu 1.6.2022. Saatavissa: https://developer.wordpress.org/rest-api/using-the-rest-api/authentication/

Wordpress. 2022b. Comments. Wordpress.org. Viitattu 5.4.2022. Saatavissa: https://developer.wordpress.org/rest-api/reference/comments/

Wordpress. 2022c. Pagination. Wordpress.org. Viitattu 5.4.2022. Saatavissa: https://developer.wordpress.org/rest-api/using-the-rest-api/pagination/

Wordpress. 2022d. Posts. Wordpress.org. Viitattu 5.4.2022. Saatavissa: https://developer.wordpress.org/rest-api/reference/posts/

Wordpress. 2022e. Tags. Wordpress.org. Viitattu 5.4.2022. Saatavissa: https://developer.wordpress.org/rest-api/reference/tags/

Liite 1. Webropol-kyselyn kysymykset ja vastausvaihtoehdot

LAB-opinnäytetyön kysely: tukikysymyssivuston käyttökokemus

1. Kuinka monta vuotta työkokemusta sinulla on IT-alalla?

_____

Mitä erityisiä vaatimuksia Avenla Oy:llä on tukikysymyssivustolle?

2. Arvioi seuraavan väittämän paikkansapitävyyttä: Raportissa mainitut toiminnallisuudet vastaavat Avenla Oy:n mielestä lopputuotteena syntyvän tukikysymyssivuston vaatimuksia.

- ◯ Täysin eri mieltä
- ◯ Jokseenkin eri mieltä
- ◯ Ei samaa eikä eri mieltä
- ◯ Jokseenkin samaa mieltä
- ◯ Täysin samaa mieltä
- ◯ En osaa sanoa

3. Miten tukikysymyssivusto auttaa Avenla Oy:tä ja alaa yleensäkin? Voit kirjoittaa vastauksen vapaamuotoisena tekstikappaleena.

_____

_____

_____

_____

◯

◯

◯

Millainen on sovelluksen käytettävyys?

4. Arvioi seuraavan väittämän paikkansapitävyyttä: Yksinkertaisten asioiden suorittaminen sovelluksessa on helppoa.

Täysin eri mieltä

Jokseenkin eri mieltä

Ei samaa eikä eri mieltä

- ◯ Jokseenkin samaa mieltä
- ◯ Täysin samaa mieltä
- ◯ En osaa sanoa

Lopputuotteen käytettävyys ja onnistuminen riippuu usein siitä, kuinka hyvin rautalankamallit ovat tehty. Kuinka huolellisesti sivusto on suunniteltu?

5. Arvioi seuraavan väittämän paikkansapitävyyttä: Raportin ohjelmistotuotanto-osuuden rautalankamallit on tehty hyvin.

- ◯ Täysin eri mieltä
- ◯ Jokseenkin eri mieltä
- ◯ Ei samaa eikä eri mieltä
- ◯ Jokseenkin samaa mieltä
- ◯ Täysin samaa mieltä
- ◯ En osaa sanoa

Kuinka loogisia ovat käyttökokemuksen rakennetason asiat?

6. Arvioi seuraavan väittämän paikkansapitävyyttä: Painonapit ovat oikeassa paikassa suhteessa muihin ohjaimiin, joita käyttäjä käyttäisi siinä tilanteessa.

◯
◯
◯
◯
◯

- ◯ Täysin eri mieltä
- ◯ Jokseenkin eri mieltä
- ◯ Ei samaa eikä eri mieltä
- ◯ Jokseenkin samaa mieltä
- ◯ Täysin samaa mieltä
- ◯ En osaa sanoa

7. Arvioi seuraavan väittämän paikkansapitävyyttä: Sivustolla navigoiminen tuntuu luonnolliselta, kun halutaan suorittaa asioita.

Täysin eri mieltä

Jokseenkin eri mieltä

Ei samaa eikä eri mieltä

Jokseenkin samaa mieltä

Täysin samaa mieltä

En osaa sanoa

8. Arvioi seuraavan väittämän paikkansapitävyyttä: Sivuston terminologiaa on helppo ymmärtää.

- ◯ Täysin eri mieltä
- ◯ Jokseenkin eri mieltä
- ◯ Ei samaa eikä eri mieltä
- ◯ Jokseenkin samaa mieltä
- ◯
- ◯

Täy-

sin

sa-

maa

mielt

ä En

osaa

sa-

noa

Kuinka johdonmukaisia ovat käyttökokemuksen luurankotason asiat?

9. Arvioi seuraavan väittämän paikkaansapitävyyttä: Tukikysymyssivuston käyttöliittymä on johdonmukainen verrattuna tuttuihin, muihin samankaltaisiin sivustoihin.

- Täysin eri mieltä
- Jokseenkin eri mieltä
- Ei samaa eikä eri mieltä
- Jokseenkin samaa mieltä
- Täysin samaa mieltä
- En osaa sanoa

10. Arvioi seuraavan väittämän paikkansapitävyyttä: Tukikysymyssivuston käyttöliittymä on johdonmukainen itsessään läpi koko sivuston.

- Täysin eri mieltä
- Jokseenkin eri mieltä
- Ei samaa eikä eri mieltä
- Jokseenkin samaa mieltä
- Täysin samaa mieltä
- En osaa sanoa

11. Arvioi seuraavan väittämän paikkansapitävyyttä: On helppoa ymmärtää, missä ollaan, mihin voidaan mennä ja mitkä sivuston valinnat auttavat pääsemään tavoitteisiin.

Täysin eri mieltä

Jokseenkin eri mieltä

Ei samaa eikä eri mieltä

Jokseenkin samaa mieltä

Täysin samaa mieltä

En osaa sanoa

Kuinka loogisia ovat käyttökokemuksen pintatason asiat?

12. Arvioi seuraavan väittämän paikkansapitävyyttä: Visuaalinen suunnittelu tukee sivustolle määritettyjä tavoitteita.

- Täysin eri mieltä
- Jokseenkin eri mieltä
- Ei samaa eikä eri mieltä
- Jokseenkin samaa mieltä
- Täysin samaa mieltä
- En osaa sanoa

13. Arvioi seuraavan väittämän paikkansapitävyyttä: Visuaalinen suunnittelu selventää käyttäjille sivustolla tarjottavia vaihtoehtoja vahvistaen rakennetta.

- Täysin eri mieltä
- Jokseenkin eri mieltä
- Ei samaa eikä eri mieltä
- Jokseenkin samaa mieltä
- Täysin samaa mieltä
- En osaa sanoa

14. Arvioi seuraavan väittämän paikkansapitävyyttä: Sivuston visuaalinen ilme viestii hyvin organisaation brändi-identiteettiä.

Täysin eri mieltä

Jokseenkin eri mieltä

Ei samaa eikä eri mieltä

Jokseenkin samaa mieltä

Täysin samaa mieltä

En osaa sanoa