Julius Saarikko

WEB-SOVELLUKSEN KEHITTÄMINEN

Tieto- ja viestintätekniikan koulutusohjelma

2019

Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

WEB-SOVELLUKSEN KEHITTÄMINEN

Saarikko, Julius
Satakunnan ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutusohjelma
Maaliskuu 2019
Ohjaaja: Nuutinen, Petri
Sivumäärä: 44
Liitteitä: 0

Asiasanat: Verkko-ohjelmointi, JavaScript, React

Tämän opinnäytetyön tavoitteena oli kehittää Enersensen E-Sense web-sovellukseen viestintäosion ulkonäöllinen osio ja suunnitella valmiiksi serveripuolen koodausta. Viestintäosioon sisältyy yksityisviestit, ryhmäviestit ja projektitiedotteet.

Työn teoriaosuudessa selvitettiin työssä käytettyjä teknologioita ja tekniikoita. Perehdyttiin mitä on web-kehitys ja mitä vaatii, että pystyy olemaan web-kehittäjä. Kävin lävitse erilaisia web-kehittäjien töitä. Front-end-kehittäjän, back-end-kehittäjän ja full stack-kehittäjän eroja kävin myös läpi.

Tämän jälkeen selvitin ketterää kehitystä ohjelmointialalla ja siihen perustuvaa Scrum tekniikkaa. Siihen liittyvät asiat olivat suuressa osassa, kun ohjelmoin sovellusta. Kävin myös lävitse ketterään kehitykseen liittyvää versionhallintaa. Tässä työssä versionhallintatyökaluna oli Git. Lisäksi kerroin muutamia tärkeitä seikkoja hyvistä ohjelmointikäytännöistä.

Kävin lävitse web-kehitykseen vaaditut ohjelmointitaidot. Työssä esitettiin myös aiheita HTML ja CSS. JavaScriptiä kävin läpi sekä normaalin JavaScriptiin liittyen että yhteen JavaScript kirjastoon Reactiin liittyen.

Käytännön osiossa esittelin, miten ratkaisin viestintäosion tuomia ongelmia. Aiheisiin kuului muun muassa front-end suunnittelu ja kehitys. Back-end puolelta loin alustavan suunnitelman toteutukselle. Esittelin työn tuloksia kuvilla ja kerroin jatkokehitysmahdollisuuksista.

WEB APPLICATION DEVELOPMENT

Saarikko, Julius
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information and Communications Technology
March 2019
Supervisor: Nuutinen, Petri
Number of pages: 44
Appendices: 0

Keywords: Web development, JavaScript, React

The purpose of this thesis was to develop messaging section for web application E-Sense made by Enersense. Interface planning and development were involved in this. Also plans for server-side coding was made. Messaging section includes private messages, group messages and project bulletins.

Theoretical part of the thesis explored the technologies and techniques used in the work. What web development is and what it requires to be a web developer. I went through various web developer jobs. The differences between front-end, back-end developer and full stack developer.

After that, I investigated agile development in the field of programming and the Scrum technology based on it. Things based on Scrum were mostly involved when I was programming the application. I also went through things about version control. Mostly I told about version control tool Git. In addition, I told some important facts about good programming practices.

I went through the programming skills required for web development. A quick overview of topics in HTML and CSS. I went through JavaScript for both normal JavaScript and a JavaScript library React.

In the practical section I introduced how I solved the problems about creating messaging section. Topics included front-end design and development. From back-end side I created the initial plan for implementation. I presented the results of the work with pictures and told something about the possibilities of further development.

SISÄLLYS

LYHENTEET

API	Ohjelmointirajapinta (Application Programming Interface)
CI	Jatkuva integraatio (Continous Integration)
CSS	HTML-kielen ulkoasumääritys tekniikka (Cascading Style Sheets)
HTML	Avoimesti standardoitu kuvauskieli (HyperText Markup Language)
IDE	Ohjelmistoympäristö (Integrated Development Environment)
MUI	Material-UI
PhpStorm	Yksi ohjelmoinnin IDE
PR	Pull Request
REST	HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen (Representational State Transfer)
UML	Graafinen mallinnuskieli (Unified Modeling Language)
VCS	Versionhallintajärjestelmä (Version Control System)

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli kartoittaa, mitä vaatii olla web-kehittäjä ja toteuttaa E-Senseen viestintäominaisuuden ulkoasullinen osio. E-Sense on Enersense Solutions Oy:n kehittämä web-sovellus. Viestinnällä E-Sensessä tarkoitetaan tämän opinnäytetyön laajuudessa yksityis- ja ryhmäviestejä sekä projektitiedotteita.

Viestintäominaisuus kehitettiin käyttämällä käyttöliittymässä Reactin versiota 15.6.2 ja valmiita MUI:n versioiden 0.18.1 ja 1.2.0 komponentteja. API puolen kehityksessä käytettiin PHP-kieltä ja suunnitelmia tehtiin UML:n avulla. Sekä käyttöliittymän että API puolen jutut tehtiin PhpStorm koodieditorilla.

Opinnäytetyössä kehitetty viestintäominaisuus on käyttöliittymällisesti valmis, mutta API puolelta on tehty vain suunnitelmat. Suunnitelmat sisältävät kuinka API toteutetaan ja alustavia valmisteluja. Ominaisuuden tuotteistaminen vaatisi API puolen rakentamisen ja automaattisten testien rakentamisen sekä manuaalista testausta.

Työssä käyn läpi mitä kaikkea web-kehitys on, kuinka monimuotoinen se on ja mitä vaatii osatakseen tehdä kokonaisuudessaan websovellus. Osaamiseni asiaan oli suhteellisen hyvä jo, saadessani Satakunnan ammattikorkeakoululta pohjia ohjelmointiin ja paljon käytännön harjoitusta Enersense:llä.

2 WEB-KEHITYS

2.1 Johdanto web-kehitykseen

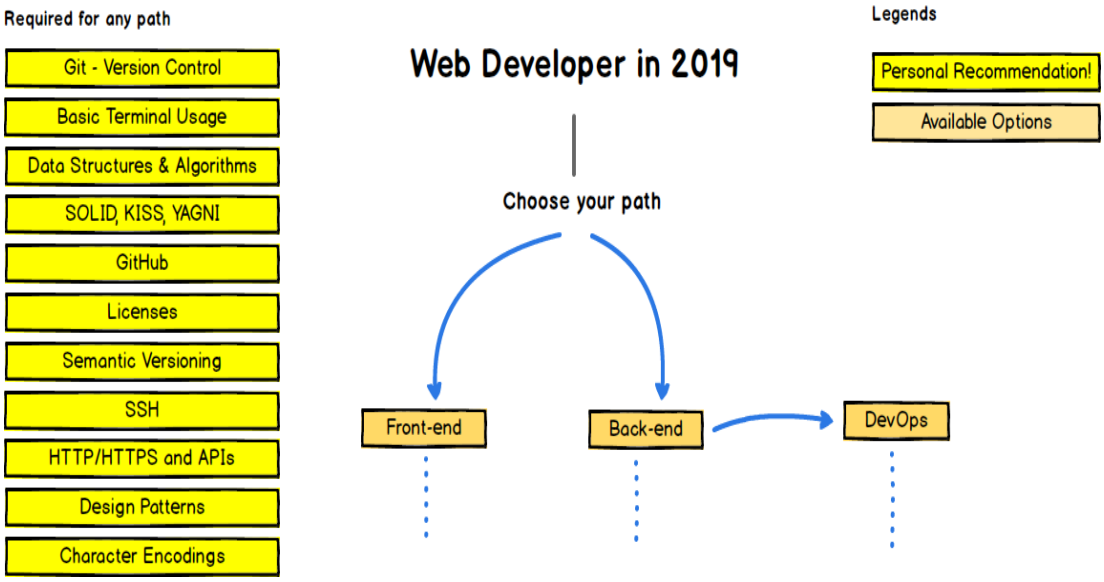
Aluksi kerron, mitä web-kehittäminen on. Web-kehittäminen tunnetaan myös nimillä web-ohjelmointi ja nettisivujen kehittäminen. Laajasti tutkailtuna web-kehittäminen on internetissä tai intranetissä näytettävien sivujen kehittämistä. Intranet on esimerkiksi yrityksen sisäinen verkko, johon ei pääse muualta käsiksi. Web-kehittämiseen kuuluu ulkoasun suunnittelu, sisällön kehittäminen, selainohjelmointi(front-end/client), serveriohjelmointi(back-end) ja turvallisuusasetusten konfigurointi. Web-kehityksen hierarkia on selaimessa näkyvien asioiden koodaus, back-end koodaus ja tietokanta. On mahdollista tehdä nettisivuja ilman, että siihen liitettäisiin back-endin eli serveripuolen koodausta. Tällaiset pelkästään selainpuolen ohjelmointia käyttävät sivut sisältävät vain informaatiota. Niillä ei siis tallennu mitään tietoa tietokantaan. (Techopedia A 2019; Code Conquest 2019.)

Web-kehittäminen on dynaamisten web-sovellusten luomista. Web-sovellusten esimerkkejä ovat muun muassa verkkokauppojen sivustot kuten Amazon tai sosiaalisen verkostoitumisen sivustot kuten Facebook. Monet väittävät, että aloittelevalle kehittäjälle web-sovellusten kehittäminen on paras tapa aloittaa ohjelmointi. Ohjelmointiympäristö on helppo asentaa, saat välittömiä tuloksia työstä ja netistä löytyy paljon online-koulutuksia. (Code Conquest 2019.)

2.2 Web-kehittäjä

Web-kehittäjä tai web-ohjelmoija on henkilö, joka ottaa web-suunnitelman ja muuttaa sen nettisivuksi. Web-suunnitelmat luodaan joko suunnittelutiimin tai asiakkaan toimesta. Tämä tehdään kirjoittamalla paljon koodirivejä eri ohjelmointikielillä. Web-kehittäjillä on melko vaikea työ kääntäessään ihmisen ymmärtämää kieltä koneen ymmärtämäksi kieleksi. Web-kehittäjä käyttää esimerkiksi englannin kieltä ja kääntää sen Pythoniksi tai HTML:ksi. (King 2018.)

Web-kehittäjiä on kolmea päätyyppiä: front-end, back-end ja full-stack. Front-end-kehittäjät ovat vastuussa siitä, mitä käyttäjä näkee ja voi tehdä nettisivulla. Back-end-kehittäjät ovat vastuussa siitä, mitä ’verhon’ takana tapahtuu. Tälläisiä tapahtumia ovat esimerkiksi miten nettisivu latautuu ja pyörii. Full stack -kehittäjät tekevät molempia. Kuvassa 1 näkyy web-kehittäjän mahdollinen polku. Voit aloittaa front-end-puolella tai back-end-puolella, josta voi tehdä myös DevOps-töitä. DevOps koostuu sanoista Development ja Operations. Se on erään tyyppinen ketterän kehityksen muoto, jossa pyritään kehittämään kommunikaatiota ja yhteistyötä kehitystiimin ja IT-operaatioiden välillä. IT-operaatioita ovat IT-osaston kaikki muut tehtävät, paitsi ohjelmistokehitys. (King 2018; Beal 2019; Rouse 2019.)



Kuva 1. Web-kehittäjän suuntautuminen (Ahmed 2018.)

2.2.1 Front-end

Front-end-kehittäjä ottaa suunnittelutiimin nettisivun ulkonäkösuunnitelman ja kirjoittaa siihen vaaditun ohjelmointikoodin. Jokaisella front-end-kehittäjällä tulee olla perusymmärrys aiheista HTML, CSS ja JavaScript. HTML on sivuston runko. Siihen kuuluvat esimerkiksi otsikot, lauseet ja taulukot. CSS on tyylimääritelmät, esimerkiksi sivun värit, nappuloiden koot ja reunukset. JavaScript tuo mukanaan interaktiivisuuden sivulle, esim. mitä tapahtuu painettaessa nappulaa. (King 2018; w3schools A 2019.)

Mitä front-end-kehittäjät sitten tekevät? He varmistavat, että sivulla on kaikki sisältö mitä tarvitaan. Sivun täytyy olla myös selkeästi esitetty ja kaiken tulee olla oikeilla paikoillaan. Joissain tapauksissa kehittäjällä voi olla myös taidot luoda sisältöä, samalla kun hän luo sivua. Sivun värien määrittely, mm. taustavärit ja otsikoiden värit ovat arkipäiväistä front-end-kehittäjän töitä. Jotkut kehittäjät ovat myös hyviä suunnittelijoita, jolloin he voivat muokata sivua samalla kun ohjelmoivat sitä. He varmistavat myös, että linkit vievät oikeaan paikkaan, nappulat toimivat oikein ja että nettisivu on miellyttävän näköinen ja responsiivinen. Responsiivinen web-kehitys tarkoittaa HTML:n ja CSS:n käyttämistä siten, että ne automaattisesti mukautuvat käyttäjän laitteeseen ja säätöihin. Tähän sisältyy sisällön koon muuttuminen ja tiettyjen asioiden piilottaminen tai näyttäminen. Erilaisia laitteita voivat siis olla tietokoneet, tabletit ja puhelimet. Näistä tabletit ja puhelimet sisältyvät mobiilikehitykseen. Responsiivinen nettisivu mukautuu sitä mukaa, mitä käyttäjä siinä tekee. Yksi suuri mobiilikehityksen huomioon otettavista asioista on ruudun kääntäminen. Mobiilikehitys on usein suuri osa työstä, samalla kun pitää varmistaa, että nettisivu näkyy kaikilla eri selaimilla oikein. Mobiilikehitys voi olla joko erillinen sovellus, joka ladataan laitteelle tai responsiivinen nettisivu. (Smashing Magazine 2011; King 2018; w3schools B 2019.)

2.2.2 Back-end

Back-end-kehittäjä luo nettisivun, ohjelman tai informaatiosysteemin toimintalogiikan. Hän luo komponentteja ja toiminnallisuuksia, joita käyttäjä epäsuorasti käsittelee front-end puolen kautta. Tyypillisesti back-end-kehittäjällä on ammattitaitoa C++:sta, C#:sta, Javasta ja muista korkean tason ohjelmointikielistä. Korkean tason ohjelmointikieli tarkoittaa sellaista kieltä, joka on lähempänä ihmisten puhumaa kieltä kuin tietokoneiden ymmärtämää kieltä. Back-end-kehittäjä vastaa siitä, että front-end puolelta pyydetyt tiedot (data) tai palvelut toimitetaan ohjelmallisesti eteenpäin. Hän luo ja ylläpitää myös koko systeemin back-endiä, joka sisältää sovelluksen logiikan, datan ja sovelluksen yhdistymisen, tietokannat, API:n ja muut prosessit. API on serverin osa, joka vastaanottaa pyynnöt ja lähettää vastaukset selaimelle. Hän myös testaa ja tarvittaessa korjaa back-end-koodin toimivuuden. (Gazarov 2016; Beal 2019; Techopedia B 2019.)

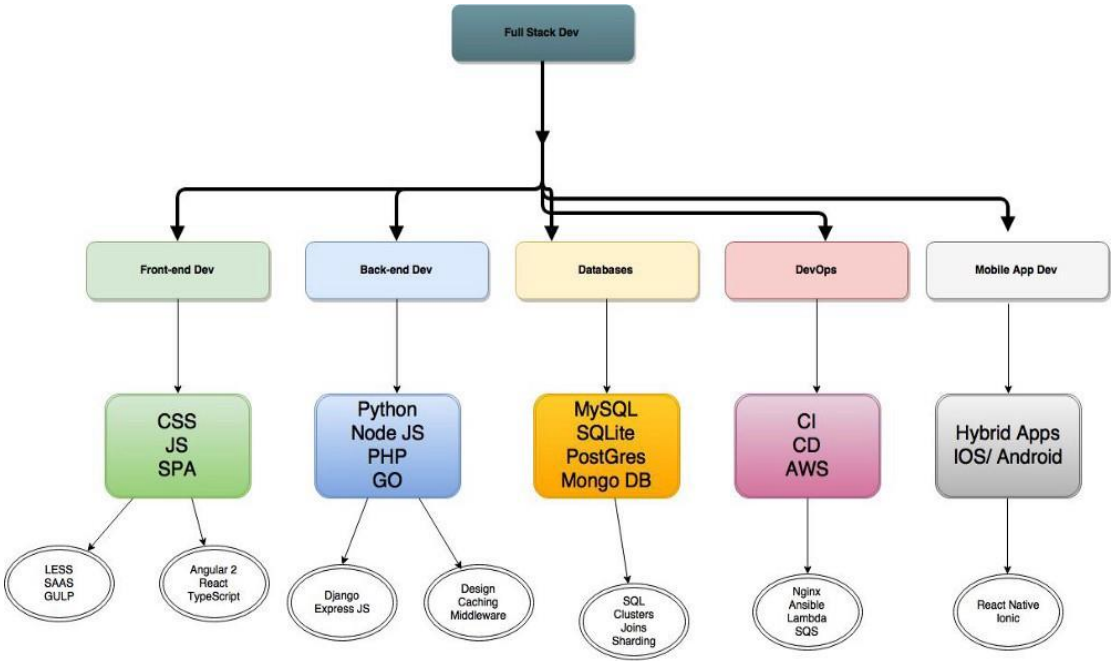
Back-end-kehittäjät työskentelevät käsi kädessä front-end-kehittäjän kanssa tarjoten serveripuolen logiikan, jolla sovellus toimii oikein. Sen lisäksi että back-end-kehittäjä vastaa sovellusten toiminnallisuudesta, he ovat myös vastuussa sovelluksen optimoinnista, nopeudesta ja tehokkuudesta. Sivuston nopeus on suuri kriteeri hakukoneoptimoinnille. Hakukoneoptimointi on sitä, miten esimerkiksi Google näyttää sivustosi hakutuloksissa. Back-end-kehittäjät luovat myös tietokannan. Tietokanta on paikka, johon tallennetaan kaikki data, kuten käyttäjät ja kommentit. Yleisesti käytettyjä tietokantoja ovat MySQL, MongoDB ja PostgreSQL. (Bradford 2018; King 2018.)

Back-end-kehittäjän tulee omata ne kielet, joita yrityksessään käytetään. Hänen tulee myös ymmärtää front-end-kehityksen kielet, HTML, CSS ja JavaScript. Tämä johtuu siitä, että back-end-kehittäjät kommunikoivat myös front-end-kehittäjien kanssa. Tärkeää on myös omata tiedot, kuinka tehdä sovelluksesta esteetön ja tietoturvallinen. Esteettömyydellä tarkoitetaan helppokäyttöisyyttä. Kaikkien tulee osata käyttää sovellusta edes jossain määrin. Olennaista on myös osata jokin versionhallintaohjelmisto, esimerkiksi Git, jotta virheen sattuessa kykenee palaamaan edelliseen versioon. Yksi taidoista on myös osata julkaista nettisivusto. Nettisivua tulee osata ylläpitää ja päivittää. (Bradford 2018.)

2.2.3 Full stack

Full stack -kehittäjä on vähintään tutustunut jokaiseen ohjelmoinnin tasoon ja osaa hyödyntää niitä työssään. Hän myös omaa aidon kiinnostuksen kaikkeen sovelluskehitykseen. Hyvä kehittäjä on sellainen, joka tuntee jokaisen tason ’stackistä’, ja osaa helpottaa muiden elämää ympärillään. Jos viisaat ihmiset lyödään yhteen käyttämään päätään ja sydäntään, parempi tuote kehittyy pienemmässä ajassa. (Gellert 2012.)

’Stack’ tarkoittaa niitä sovelluksen kehittämiseen liittyviä osaamisia, joilla projekti saadaan vietyä kokonaisuudessaan alusta loppuun. Nämä eivät rajoitu vain selainpuolen ja serveripuolen ohjelmointiin. Lisäksi pitää myös hallita back-endiin liittyviä asioita. Näitä ovat esimerkiksi tietokannat, pilvipalvelut kuten AWS (Amazon Web Services), testaaminen CI-tyyppisesti (Continuous Integration) sekä mobiiliapplikaatioiden tekeminen (katso kuva 2). (Liu 2017.)



Kuva 2. Full stack -kehittäjän osaamisalueet (Liu 2017.)

Full stack -kehittäjän tulee ohjelmointikielien lisäksi hallita useita muita asioita. Näitä asioita ovat ohjelmointikielen kielioppi ja kuinka sovelluksen rakenne, suunnittelu, toteutus ja testaus tehdään valittujen ohjelmointikielien perusteella. On olemassa sanonta “Jack of all trades, but master of none”. Tällä on hieman eri merkitys suomen kielessä, sanonnan käännös on “jokapaikanhöylä”. Tällä tarkoitetaan sitä, että full stack -kehittäjän ”stack” on iso. Tämä johtaa siihen, että vaikka tietää useasta asiasta jotain, ei pysty hallitsemaan kaikkia asioita täydellisesti. (Liu 2017.)

3 OHJELMISTON KETTERÄ KEHITTÄMINEN

Ohjelmiston ketterä kehittäminen (agile software development) on prosessi, joka auttaa ohjelmointitiimejä tarjoamaan nopeasti vastauksia heille tulleesta palautteesta. Ketterä kehittäminen luo mahdollisuuksia arvioida projektin kulkua kehityssyklin aikana. Kehitys tapahtuu asiakkailta saaduista kehitystiketeistä, joita kutsutaan käyttäjätarinoiksi. Näissä tiketeissä asiakas kertoo kirjallisesti, millaisen ominaisuuden tarvitsee sovellukseen. Tämän jälkeen voidaan pitää vielä täydentävä kokous, jossa

keskustellaan asiasta ja kuinka se toteutetaan. Koska asiakas on aktiivisesti mukana koko projektissa, on kaikkien osapuolien välillä jatkuvaa yhteistyötä. Tämä helpottaa kehitystiimiä ymmärtämään paremmin asiakkaan toiveet ja visio. Kehitystiimin tarjotessa jatkuvasti toimivaa ja laadukasta ohjelmistoa, asiakkaan ja tiimin välinen sitoutuminen paranee. Ennen ketterää kehittämistä, sovelluksia kehitettiin vesiputousmallin avulla. Siinä ennen kuin kehitys alkoi, oli tehtävä suuri määrä dokumentaatiota. Dokumentaatioissa kerrottiin ensin vaatimusmäärittely. Dokumentit olivat pitkiä ja niissä kerrottiin yksityiskohtaisesti kaikki. Näitä asioita olivat mm. strategian kattavat toiminnalliset määritelmät ja visuaalisen käyttöliittymän suunnittelut. (Sacolick 2018; Goncalves 2019.)

3.1.1 Scrum

On olemassa useita eri metodeja, jotka toimivat ketterän kehityksen ehdoilla. Näitä ovat muun muassa Scrum, XP (extreme programming) ja Crystal. Näistä keskityn tähän opinnäytetyöhön liittyvään Scrummiin. Scrum on projektinhallinnan viitekehys, jolla on kauaskantoisia kykyjä hallita lisäyksiä kaikissa projektityypeissä. Scrummia voidaan käyttää muiden ketterää kehitystä käyttävien viitekehysten kanssa erilaisissa tekniikoissa. Scrum on kasvattanut suosiotaan ketterän kehityksen viitekehyksenä, koska se on yksinkertainen ja sillä on todistettu tuottavuusaste. (Goncalves 2019.)

Scrum on kevyt ja helppo ymmärtää, mutta vaikea hallita. Scrummin avulla ihmiset voivat käsitellä monimutkaisia ja mukautuvia ongelmia, samalla kun he tuottavat tehokkaasti tuotteita, joilla on mahdollisimman suuri arvo. Scrum toteuttaa empirismin tieteellistä menetelmää. Scrum-tiimi koostuu tuotteen omistajasta, kehitystiimistä sekä Scrum-mestarista. (Scrum A 2018; Goncalves 2019.)

Tuotteen omistaja valvoo projektin liiketoimintaolosuhteita varmistaakseen, että tuote on rakennettu ja oikeassa järjestyksessä. Hyvä tuotteen omistaja tasapainottaa kilpailevat prioriteetit, on tiimin käytettävissä ja tekee päätöksiä projektista. Scrum-mestari on joukkueen valmentaja, hän auttaa tiimiä työskentelemään tehokkaasti yhdessä. Scrum-mestari huolehtii esteistä, jotka hidastavat kehitystä. Hän myös tehostaa kokouksia ja keskusteluryhmiä sekä suorittaa muita projektinhallintatehtäviä.

Kehitystiimi työskentelee yhdessä määrittääkseen parhaan tavan saavuttaa tuotteen omistajan haluamat tuotetavoitteet. Tiimi päättää keskenään, kuka hoitaa mitäkin tehtävää ja määrittelee tekniset käytännöt, joita tarvitaan haluttujen tavoitteiden saavuttamiseen. (Goncalves 2019; Scrum C 2019.)

Tiimit ovat itseorganisoituvia ja monitoiminallisia. Tiimit valitsevat itse, kuinka toteuttaa työnsä, sen sijaan että heitä ohjailtaisiin tiimin ulkopuolelta. Monitoiminnallisella ryhmällä on kaikki tehtävät, jotka tarvitaan työn suorittamiseksi. Tehtävät eivät riipu muista henkilöistä, jotka eivät kuulu tiimiin. Tiimimalli on suunniteltu optimoimaan luovuus, joustavuus ja tuottavuus. Sprintit ovat jaettu viiteen eri tapahtumaan. Nämä tapahtumat ovat Sprint kokonaisuudessaan, suunnittelu (sprint planning), päivittäinen Scrum (daily Scrum), tarkastelu (sprint review) sekä taaksepäin katsova osio (sprint retrospective). (Scrum A 2019.)

3.1.2 Sprint

Tiimit arvioivat projektia säännöllisesti ”sprinteiksi” kutsutuissa kokouksissa. Sprintit ovat kiinteältä kestoltaan 1-4 viikkoa. Kesto sovitaan tiimin kesken ja keston määrittelemisen avulla on helppoa arvioida ja rajoittaa kustannuksia. Sprintit helpottavat myös uusien ominaisuuksien saapumisen ennakointia. Tämä auttaa myös kehitystiimiä testaamaan tuotetta paremmin. Jos haluttu ominaisuus valmistuu etuajassa, voidaan se myös julkaista käyttöön aiemmin kuin on suunniteltu. Rajoitus neljään viikkoon johtuu yksinkertaistamisesta. Sprintin kestäessä kauemmin, rakennettavan asian määritys voi muuttua, monimutkaisuus ja riskit voivat kasvaa. Rajoituksella mahdollistetaan ennustettavuuden varmistaminen. Edistystä tarkastetaan vähintään kuukausittain, jolloin myös kustannusten riski on sidottuna yhteen kuukauteen. Seuraava sprint alkaa välittömästi edellisen sprintin loputtua. (Goncalves 2019; Scrum B 2019.)

Sprintin aikana ei tehdä muutoksia, jotka voisivat vaikuttaa ja vaarantaa Sprintin tavoitetta. Laatutavoitteet eivät saa laskea mahdollisten muutosten myötä. Sprintin sisältöä voidaan tarvittaessa selventää ja muokata tuotteen omistajan ja kehitystiimin välillä, kun asiasta on opittu enemmän. Sprint suunnittelussa sovitaan, mitä kehitetään seuraavan sprintin aikana. Työt valitaan kehitettävien ominaisuuksien listalta (product

backlogilta) ja viedään sprintin backlogille. Backlog on eräänlainen tilauskanta, siellä säilötään kaikki ominaisuudet, jotka tulee kehittää tuotteelle. Sprintin backlog ei ole sitoumus työlle, se on ennuste mitä onnistutaan kehittämään sprintin aikana. Ainoa ”säiliö” sprintille on sen kesto. (Scrum B 2019; Scrum C 2019; Scrum G 2019.)

Päivittäinen Scrum kestää 15 minuuttia. Päivittäisen Scrummin aikana kehitystiimi keskustelee viimeisen päivän tekemisistä ja suunnittelee seuraavan 24 tunnin ohjelman. Keskustelun avulla optimoidaan tiimityötä ja suorituskykyä. Keskustelu pidetään samaan aikaan ja samassa paikassa päivittäin monimutkaisuuden välttämiseksi. Sprintin tarkastelu tehdään aina sprintin lopussa. Tarkastelussa käydään asiat läpi, jotka tuli tehtyä ja joita ei saatu tehtyä. Tarkasteluun osallistuu Scrum tiimin lisäksi asiakkaat, joiden haluamia ominaisuuksia kehitettiin sprintin aikana. Sprintin taaksepäin katsovassa osiossa (Retrospective), tarkastellaan mikä meni sprintissä hyvin, mitä voitaisiin parantaa ja miten tiimi kykenisi parantamaan seuraavaan sprinttiin. (Scrum D 2019; Scrum E 2019; Scrum F 2019.)

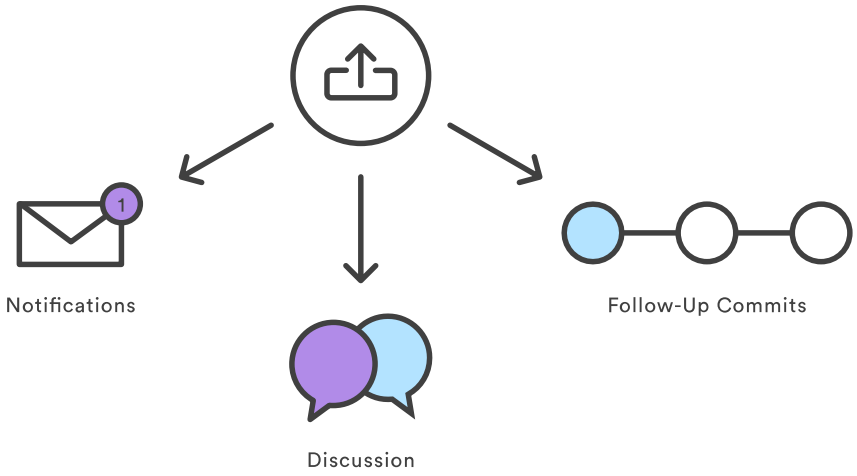
3.2 Versionhallinta

Mitä on versionhallinta ja miksi sitä kannattaisi käyttää? Versionhallinta on järjestelmä, joka tallentaa tehdyt muutokset tiedostoon tai tiedostojen ryhmään ajan mittaan. Tämä helpottaa asioiden muistamista ja tiettyihin versioihin voi palata myöhemmin. Tässä opinnäytetyössä otan kantaa koodin hallintaan, vaikka versionhallintaa voidaan käyttää lähes minkä tahansa tiedostotyypin kanssa. Jos olet web-suunnittelija tai graafikko ja haluat säilyttää jokaisen ulkoasuversion tai kuvan, VCS (versionhallintajärjestelmä) on viisas valinta. VCS:n avulla voit palauttaa valitut tiedostosi takaisin edellisiin tiloihin, nähdä kuka on viimeksi muokannut tiedostoa, jos vaikka uusimmassa versiossa on ongelmia. Voit myös verrata muutoksia versioissa ja jos vahingossa poistat tiedostoja tai muuten tuhoat niitä, voit helposti palauttaa ne edelliseen tilaan. Lisäksi tämä kaikki tapahtuu pienellä vaivalla. Versionhallintajärjestelmien merkitystä muutosten seurannassa ei voida yliarvioida, ja yritysten tulisi valita se järjestelmä, joka parhaiten vastaa heidän tarpeitaan. (Git A 2019; Kadivar 2018.)

VCS:n suurimpia etuja ovat kehittämisprosessin virtaviivaistaminen, koodin hallinta useille projekteille ja koodihistoriassa kaikkien muutosten pitäminen. VCS voidaan integroida useisiin ohjelmistokehitystyökaluihin, kuten integroituihin kehitysympäristöihin (IDE). On olemassa useita eri palveluita, jotka tarjoavat säilytysalustan koodeille. Näitä ovat mm. GitHub, Gitlab ja Bitbucket. Osan alustoista voi asentaa paikalliselle serverille, palveluntarjoajan datakeskukseen tai pilveen. (Kadivar 2018.) Tässä opinnäytetyössä on käytettynä apuna Atlassianin Bitbuckettia.

Bitbucket tarjoaa ilmaisen alustan pienelle tiimille (5 henkilöä) tai maksullisia vaihtoehtoja suuremmille tiimeille. Bitbucket hallitsee sovelluksen versioita. Hyvänä ominaisuutena on integroitu Pipelines CI. Muitakin CI-työkaluja voi halutessaan käyttää, kuten Jenkins CI:tä, josta kerron osiossa 3.2.2 jatkuva integraatio. (Bitbucket 2019.)

PR:n luominen Bitbucketissa on myös helppoa. PR eli pull request on ominaisuus, jolla kehittäjä voi ilmaista muulle kehitystiimille saaneensa työn alla olleen ominaisuuden valmiiksi. Kehittäjä luo PR:n ja liittää siihen haluamansa henkilöt tarkistajiksi. Hyvä ominaisuus on myös mahdollisuus kommentoida PR:n koodia. Muut kehitystiimin jäsenet voivat tarkastaa koodin ja kommentoida sitä, hyväksyä tai hylätä sen. Kuvassa 3 näemme, kuinka kehitystiimi käsittelee pull requestia. Ylhäällä oleva nuoli on PR, siitä lähtee ilmoitukset muulle kehitystiimille. Kehitystiimi sitten voi keskustella PR:stä ja tarvittaessa laittaa korjauksia tai lisäyksiä koodiin. (Bitbucket 2019; Atlassian A 2019.)



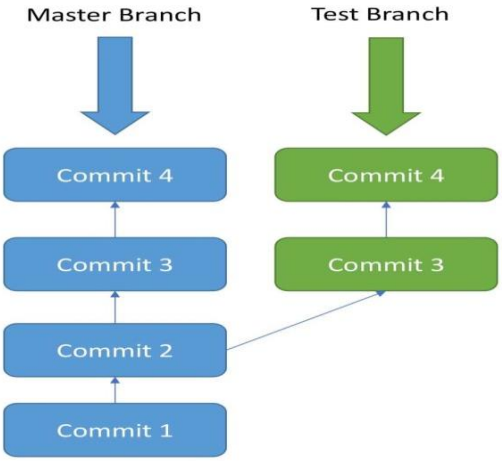
Kuva 3. Pull requestin käsittely (Atlassian A 2019.)

Bitbucketin saa yhdistettyä Atlassianin muihin tuotteisiin, kuten Jiraan. Jirassa on projektiin liittyvät tarinat (backlogit) ja käyttäjätarinat (user storyt). Sprintin backlogit ja suunnittelu voidaan myös toteuttaa Jiran kautta. (Bitbucket 2019; Atlassian B 2019.)

3.2.1 Git-versionhallintaohjelmisto

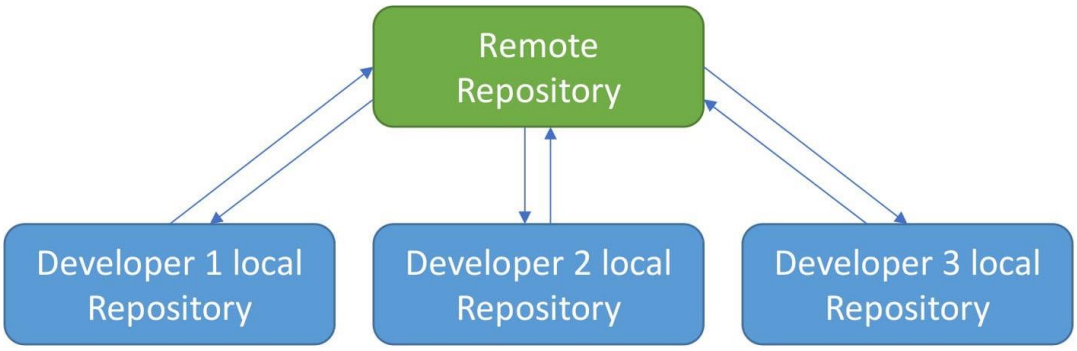
Git on avoimen lähdekoodin hajautettu versionhallintajärjestelmä. Käytetään enimmäkseen koodin tallentamiseen. Gitissä oleva koodi muuttuu sitä mukaa kun koodia lisätään. Tosielämän projekteissa on yleensä useita kehittäjiä, jotka toimivat samanaikaisesti. Monet kehittäjät voivat lisätä samaan haaraan (branch) koodiaan. Tästä syystä on kehitetty Git. Git varmistaa, ettei kehittäjien koodin välillä ole ristiriitoja. Hajautettu järjestelmä tarkoittaa sitä, että koodia ei tallenneta vain keskusserverille. Koodit löytyvät myös kokonaisuudessaan kehittäjien omissa koneissa. Aina kun koodia tallennetaan, Git tallentaa sen tietynlaisena kuvana. Kuvassa on tieto, miltä kaikki tiedostot näyttävät sillä hetkellä. Jos jokin tiedosto ei ole muuttunut, Git ei tallenna siitä uutta versiota. Muuttumattomasta tiedostosta tallennetaan vain linkki edelliseen samalaiseen tiedostoon. (Sridhar 2018; Git B 2019.)

Projektien vaatimusten muuttuessa voidaan käyttää hyväksi Gitin kaltaista ohjelmistoa, jolla voidaan palauttaa edellinen tai joku edellisistä versioista. Lisäksi toisinaan on useita hankkeita, joilla on sama koodipohja. Tässäkin Git on hyödyllinen. Gitin avulla voidaan luoda yksi päähaara, josta kehittäjät ottavat pohjan itselleen. Tähän päähaaraan ei välttämättä lisätä mitään. Kehittäjät voivat toteuttaa eri hankkeita samalla koodipohjalla ja siirtää koodiaan eri haaroihin, kuten näemme kuvassa 4. (Sridhar 2018.)



Kuva 4. Git haarat (Sridhar 2018.)

Kun käytetään Gittiä, ensin luodaan paikallinen kansio (repository), joka toimii säilytyspaikkana kaikelle koodille. Repository luodaan komennolla ”git init”. Seuraavaksi voidaan luoda sisältöä kyseiseen kansioon. Voit lisätä esimerkiksi demo.txt-nimisen tiedoston komennolla ”git add demo.txt”. Tämän jälkeen tulee kyseinen muutos lähettää eteenpäin ”commit” komennolla ”git commit -m”. Commit-komennon perään voidaan vielä lisätä heittomerkkien sisälle kommentti liittyen kyseiseen committiin. Tässä vaiheessa ei olla vielä luotu omaa haaraa (branch), vaan se luodaan komennolla ”git branch ’haaran nimi’”. Tämä ei kuitenkaan vielä siirrä Gittiä käsittelemään sitä haaraa. Siihen pitää vielä erikseen siirtyä komennolla ”git checkout ’haaran nimi’”. Kun saadaan haluttu ominaisuus valmiiksi, voidaan se yhdistää (merge) päähaaran (master branch) kanssa. Yhdistämisen kanssa voi välillä tulla konflikteja, jotka pitää ratkaista. Konfliktien ratkaiseminen voi alkuunsa tuntua hankalalta, mutta kokemuksen kertyessä sen ymmärtää paremmin. Tässä vaiheessa on käsitelty vain paikallisia tiedostoja. Siirrettäessä serverille tiedostot, Gitin avulla ”työnnetään” (push) tiedostot serverille. Kaksi muuta hyödyllistä komentoa ovat vielä ”vetäminen” (pull) ja kloonaus (clone). Pull-komennolla voidaan ottaa omalle tietokoneelle serverillä oleva haara tai sen viimeisimmät päivitykset. Kloonaamalla voidaan ottaa kokonainen repository omalle tietokoneelle ja alkaa käsittelemään sitä. Kuvassa 5 näkyy, kuinka useampi kehittäjä voi keskustella saman repositoryn kanssa omalta koneeltaan. Useat kehittäjät voivat samanaikaisesti kehittää samaa repositorya. Näissä tilanteissa voi syntyä konflikteja, jotka jonkun kehittäjistä tulee ratkaista. (Sridhar 2018.)



Kuva 5. Git repository (Sridhar 2018.)

3.2.2 Jatkuva integraatio

CI eli jatkuva integraatio on kehityskäytäntö, joka vaatii kehittäjiä integroimaan koodinsa jaettuun repositoryyn useita kertoja päivässä. Jokainen integrointi tarkistetaan automaattisilla testeillä, jolloin kehitystiimi voi havaita ongelmat aikaisessa vaiheessa. Integroimalla säännöllisesti virheet löytyvät nopeasti ja helposti. Integroimalla usein, on vähemmän taaksepäin tutkimista missä asiat menivät pieleen, joten voit käyttää enemmän aikaa kehittämään ominaisuuksia. CI on halpa käytäntö verrattuna siihen, että koodia ei integroitaisi useasti. Integroidessa koodia harvemmin, virheiden löytäminen ja varsinkin korjaaminen on eksponentiaalisesti vaikeampaa. Tällaiset viivästykset voivat helposti suistaa projektin pois aikataulustaan tai aiheuttaa sen epäonnistumisen. (ThoughtWorks 2019.)

CI toimii periaatteella, jossa kehittäjät työntävät koodin repositoryyn. CI serverissä oleva putkilinja (Pipeline) tutkii repositorya löytääkseen muuttuneita tiedostoja. Putkilinja on se osio CI serveristä, jossa kaikki testit ajetaan. Tutkimisen jälkeen putkilinja kääntää ja rakentaa (build) koodin, jonka jälkeen serveri ajaa yksikkö- ja integraatiotestit kyseiselle repositorylle. CI serveri määrittää ”buildille” versiomerkinnän ja kertoo kehitystiimille onnistuneesta ”buildista”. Testien epäonnistuessa CI serveri ilmoittaa kehitystiimille, joka korjaa ongelmat. Kehitystiimillä on muutamia vastuita, jotta CI toimisi halutulla tavalla. Näitä ovat esimerkiksi se, että koodia pitää työntää usein eteenpäin ja että ei saa laittaa rikkinäistä tai manuaalisesti testaamatonta koodia eteenpäin. (Power 2016; ThoughtWorks 2019.)

Jenkins CI on johtava avoimen lähdekoodin automaatiopalvelin, jossa on noin 1400 kehitystehtävien automatisointia tukevaa laajennusta. Nämä laajennukset kattavat viisi eri aluetta, käyttöliittymän, hallinnan, alustat (platform), lähdekoodin hallinnan ja edellä mainitun rakentamisen (buildin). Jenkins tarjoaa yksinkertaisen tavan luoda jatkuvaa integrointia lähes kaikkiin ohjelmointikielien ja lähdekoodi-repositoryjen yhdistelmiin putkilinjojen avulla. Lisäksi Jenkins automatisoi muita rutiininomaisia kehittämistehtäviä. Jenkins ei poista tarvetta luoda yksittäisiin vaiheisiin skriptejä. Jenkins antaa nopeamman ja vankemman tavan integroida rakentamis-, testaus- ja käyttöönottotyökalujen ketju, kuin mitä itse rakentaisit. (Heller 2017.)

3.3 Hyviä ohjelmointikäytäntöjä

Aloitteleva kehittäjän ei välttämättä tule miettineeksi hyviä ohjelmointikäytäntöjä. On helppoa kirjoittaa koodista toimivaa. Koodin ymmärrettävyys on suurin ongelma. Voit kirjoittaa nopeasti jonkin koodinpätkän ja todeta, että se toimii. Myöhemmin kun palaat omaan koodiisi, voit joutua miettimään, mitä ihmettä olet kirjoittanut. Tästä syystä tulisi noudattaa hyviä ohjelmointikäytäntöjä ja yrityksen sisällä kannattaisi kaikkien toimia samoilla käytännöillä. Niiden avulla saadaan työskentelystä nopeampaa ja helpompaa. Hyvät ohjelmointikäytännöt auttavat myös uusia työntekijöitä sopeutumaan yrityksen koodaustottumuksiin. Ideaalitapauksessa koodin tulisi itse kommentoida itseään. Eli koodin tulee olla niin ymmärrettävää, että koodikommentit olisivat vain saman asian kertomista uudelleen.

Kehittäjän tulee aina kirjoittaa koodia, jota on helppo lukea ja helppo kehittäjille ymmärtää, koska aikaa ja resursseja kuluu enemmän vaikeaselkoisen koodin kanssa. Mitä yksinkertaisempaa koodia tehdään, sitä todennäköisempää on, että siinä on vähemmän virheitä ja siten resursseja ei tarvitse käyttää niin paljoa virheiden korjaamiseen. Koodin tulisi sisältää vain ne asiat, joita oikeasti käytetään. Tärkeää on ajatella ensiksi, mitä tekee ja miten. Koodin kirjoittaminen ilman sen arkkitehtuurin ajattelua on yhtä hyödytöntä kuin uneksia haaveistasi ilman, että aiot tehdä niiden eteen mitään. Aikaa arkkitehtuurin suunnitteluun tulee varata riittävästi. (Nikishaev 2017.)

Koodin tulisi olla ymmärrettävää ilman että siihen kirjoitetaan kauhea määrä kommentteja kertomaan, miten se toimii. Lyhyt ja ytimekäs kommentti riittää. Kommentin tulee kertoa mitä ja miten metodi toimii. Tämä auttaa uusia ohjelmoijia ymmärtämään nopeammin koodia. Koodin tarkistus muiden kehittäjien toimesta voi olla hyvä tai huono asia. Monet ajattelevat, että koodin tarkistus on hyvä tapa opettaa uusia tiiminjäseniä. Tarkistuksen päätarkoitus on ylläpitää koodin laatua eikä opettaminen. Mieti tilanne, jossa oppinut kehittäjä on tehnyt erittäin monimutkaisen koodin tarkistettavaksi uudelle tyypille, joka ei vielä tiedä asiasta paljoakaan. Monet asiat voivat mennä tässä pieleen. Koodia ei tulisi myöskään kirjoittaa väsyneenä tai huonolla tuulella. Tämä lisää virheiden määrää verrattuna siihen, että koodia kirjoitettaisiin täynnä energiaa. Ajatustyö ei kuitenkaan ole sama asia kuin käyttäisit hauistasi. (Nikishaev 2017.)

4 KÄYTETYT TEKNIIKAT

Tässä osiossa käyn opinnäytetyöhöni tarvittavia tekniikoita. Käytin kaikkia seuraavaksi mainittuja tekniikoita työssäni, jossa rakensin viestintäosiota web-sovellukseen. HTML, CSS ja JavaScript ovat tekniikat, joita tarvitaan web-sovelluksen luomiseen. Tässä opinnäytetyössä käytetään myös JavaScriptin kirjastoa Reactia, joka mielestäni helpottaa suuresti web-sovelluksen luomista.

4.1 HTML

HTML (hypertext markup language) antaa kehittäjän luoda ja rakentaa kappaleita, otsikoita, linkkejä ja lohkoja verkkosivuille ja sovelluksille. Se ei ole ohjelmointikieli, koska sen avulla ei voida luoda dynaamista toiminnallisuutta. Sen sijaan asiakirjojen järjestäminen ja muotoilu on mahdollista, kuten Microsoft Wordilla. Hypertext tarkoittaa tekstiä, joka sisältää viittauksia (linkkejä) muihin teksteihin, joihin lukijalla on välitön pääsy. HTML on merkintäkieli, joka on erittäin suoraviivainen ja helppo oppia jopa aloittelijoille. Useimmat ihmisistä oppivat perusteet jo yhdellä oppimiskerralla. HTML-tiedostot päättyvät päätteeseen .html tai .htm. (Shannon 2012; B 2018.)

Kun kirjoitetaan HTML-kielellä, käytetään yksinkertaisia koodirakenteita (attribuutteja ja tunnisteita), jotta voidaan merkitä verkkosivu. Markup tarkoittaa, miten HTML tunnisteet kohtelevat sisällään olevaa tekstiä. On olemassa kolme tunnistetta, jotka jokaisessa HTML-dokumentissa tulee olla. Ne ovat html, head ja body. Html-tunniste on korkeatasoisin, joka sulkee kaiken muun sisäänsä. Head-osiosta löytyy metainformaatio kuten sivun otsikko ja millä merkistöllä sivu on tehty. Body sisältää kaiken sisällön. Kappale voidaan esimerkiksi luoda sijoittamalla teksti ja tunnisteen väliin. Jälkimmäisessä oleva kauttaviiva ”/”, merkkaa sen, että se sulkee kyseisen tunnisteen. Kuvassa 6 näemme yksinkertaisen esimerkin HTML-koodista. (Shannon 2012; B 2018.)

Example of a Basic HTML Document

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example Web Page</title>
  </head>
  <body>
    <h1>Example Page</h1>
    <p>This is a sample page.</p>
    <p>To read more, go to our <a href="info.html"</a>info</a> page.</p>
    <!--- this is a comment in the code -->
  </body>
</html>
```

Kuva 6. Esimerkki HTML-koodista. (Crawford 2019.)

4.2 CSS

CSS määrittää web-sivun tyylit, kuten sivun asettelut, värit, fontit. CSS on siis websivuston esteettiset valinnat, kun verrataan vaikka viktoriaanisen aikakauden kartanoa ja vuosisadan puolivälin modernia kotia, niin huomataan suuri ero. CSS tuo tyylimäärittelyt web-sivuille HTML-elementtien kanssa. Elementit ovat web-sivun yksittäisitä HTML-komponentteja, esimerkiksi kappale HTML:ssä voisi näyttää tältä: ” Tässä on lause ”. Jos halutaan, että tämä kappale olisi väritykseltään pinkki ja teksti lihavoituna, CSS-koodi näyttäisi tältä: ”p {color:pink; font-weight:bold;}”. CSS-koodissa kerrotaan ensin, mitä elementtiä muokataan, tässä tapauksessa ”p” eli kappaletta. Sen jälkeen annetaan määrittelyt sille elementille. CSS-tiedoston pääte on .css. (Morris 2018.)

Kuinka sitten voidaan käyttää CSS-määrittelyjä HTML-tiedostossa? Niin kuin HTML, myös CSS kirjoitetaan normaalina tekstinä tekstieditorissa tai IDE:ssä. On olemassa kolme päävaihtoehtoa, miten ja mistä käyttää CSS-määrittelyjä. Ne ovat erillinen tiedosto, HTML-tiedoston sisäinen CSS-määrittely sekä koodin seassa oleva ”inline” määrittely. Kuvassa 7 nähdään yksinkertainen esimerkki, kuinka tehdään erillinen CSS-tiedosto. Kuvassa 8 on esimerkki HTML-tiedoston sisäisestä tyylimäärittelystä. Kuvassa 9 esimerkki HTML koodin seassa olevasta tyylimäärittelystä. (Morris 2018.)

```
body {
    background-color: #d0e4fe;
}

h1 {
    color: orange;
    text-align: center;
}

p {
    font-family: "Times New Roman";
    font-size: 20px;
}
```

Kuva 7. CSS-koodin esimerkki (Muhammad 2018.)

```
<head>
<style>
Body  {  background-color:thistle;  }
P  {  font-size:20px;  color:mediumblue;  }
</style>
</head>
```

Kuva 8. HTML-tiedoston sisäinen tyylimäärittely (Morris 2018.)

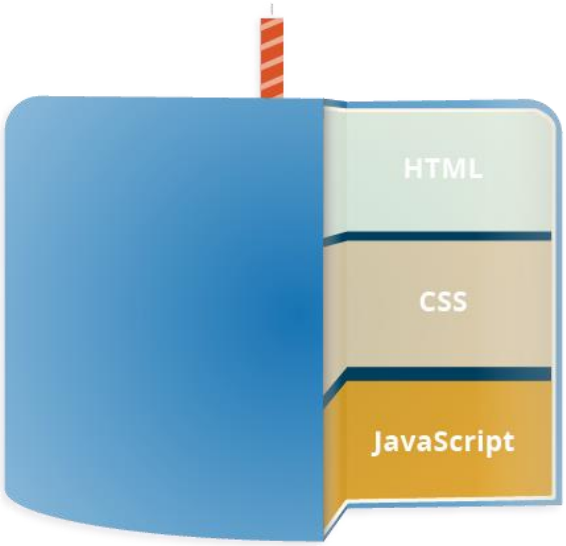
```
<h1  style=”font-size:40px;color:violet;”>Check out this headline!</h1>
```

Kuva 9. HTML-koodin seassa oleva tyylimäärittely (Morris 2018.)

4.3 JavaScript

4.3.1 Yleistä JavaScriptistä

JavaScript on skriptaus- tai ohjelmointikieli, jonka avulla voidaan toteuttaa monimutkaisia asioita verkkosivuilla. Aina kun web-sivu tekee jotain muutakin kuin vain on ja näyttää staattista tietoa, on JavaScript todennäköisesti siinä mukana. Näitä muuttuvia asioita ovat esimerkiksi sisällönpäivitykset, interaktiiviset kartat ja animoitu grafiikka. JavaScript siis laskee, käsittelee ja validoi dataa. Voidaan ajatella, että HTML, CSS ja JavaScript luovat kolmikerroksisen kakun. Kakun päällimmäisenä on HTML, joka määrittelee sisällön. Kakun toisessa kerroksessa on CSS, joka määrittelee tyylit. Pohjimmaisena löytyy JavaScript, joka hoitaa kaiken toiminnallisuuden. Kuvassa 10 näkyy kolminaisuuden kuvaus kakkuna kuvattuna. (MDN web docs 2019; w3schools C 2019.)



Kuva 10. Web-kehityksen kolminaisuus (MDN web docs 2019.)

JavaScript-kieli koostuu yleisistä ohjelmointitoiminnoista, joiden avulla voidaan tehdä useita asioita, mm. säilyttää hyödyllisiä arvoja muuttujien sisällä. Voidaan esimerkiksi pyytää käyttäjän nimi, joka tallennetaan muuttujaan nimeltään ”nimi”. JavaScriptillä on olemassa vain yksi numerotyyppi. Sen avulla voidaan kirjoittaa numerot desimaalien kanssa tai ilman. JavaScript hoitaa myös käynnistyskoodit vastauksena tiettyihin web-sivun tapahtumiin. Esimerkiksi nappulaa painettaessa JavaScript havaitsee sen ja suorittaa koodin, joka sille on asetettu. JavaScript suoritetaan selaimen JavaScript-

moottorilla. JavaScriptin suoritus alkaa, kun HTML ja CSS on jo koottu yhteen verkkosivulla. Näin varmistetaan, että sivun rakenne ja tyylit ovat käytössä, kun JavaScriptin suorittaminen alkaa. Jos JavaScriptiä yritetään ladata ja suorittaa ennen kuin HTML ja CSS ovat latautuneet, syntyy virheitä. (MDN web docs 2019; w3schools C 2019.)

Jokainen selaimen välilehti on oma suoritusympäristönsä käynnissä olevalle koodille. Jos näin ei olisi, voisivat haitallisessa tarkoituksessa luodut ulkopuoliset sivustot varastaa toisilta sivustoilta tietoja käyttäjän tietämättä siitä mitään. Kuvassa 11 on JavaScriptillä luotu vakio ”para”, joka etsii HTML tiedostosta ensimmäisen <p>-elementin ja valitsee sen. Tämän jälkeen määritetään EventListener, joka luo elementin klikattavaksi ja kuuntelee kyseistä elementtiä. Kun kyseistä elementtiä klikataan, EventListener kutsuu funktiota updateName. Funktiossa määritetään muuttuja ”name”. Klikattaessa se avaa myös ponnahdusikkunan, jossa pyydetään syöttämään uusi nimi. Tämän jälkeen funktio määrittää valitun elementin sisällöksi ”Player 1: ’syötetty nimi’”. (MDN web docs 2019.)

```
1  const para = document.querySelector('p');
2
3  para.addEventListener('click', updateName);
4
5  function updateName() {
6    let name = prompt('Enter a new name');
7    para.textContent = 'Player 1: ' + name;
8  }
```

Kuva 11. JavaScript esimerkki (MDN web docs 2019.)

JavaScriptiä voi kutsua melko samoin tavoin kuin CSS-tiedostojakin. Sen on joko oma erillinen tiedostonsa tai sitten voit kirjoittaa sitä suoraan HTML-tiedostoon. Tätä ei kuitenkaan suositella ”normaalissa” JavaScriptissä tehtävän sen ollessa hieman huono ohjelmointikäytäntö. Tässä on ristiriitaisuus Reactin kanssa, josta kerron luvussa 4.3.2. JavaScript-tiedoston tiedostopääte on .js. (MDN web docs 2019.)

4.3.2 React

React on Faceookin kehittämä avoimen lähdekoodin JavaScript-kirjasto käyttöliittymien rakentamiseen. Reactilla on yli 100 000 tähteä GitHubissa. GitHub on kehitysalusta, kuten Bitbucket. JavaScript osiossa viittasin ristiriitaan. Erään React ohjeen kirjoittaja Rascia hämmentyi Reactin koostuessa HTML:n ja JavaScriptin sekoituksesta. Rascia pohtii, että eikö se ole juuri sitä, mitä on yritetty välttää? Ennen kuin alkaa käsittelemään Reactia, kannattaa opetella HTML, CSS ja perusymmärrys JavaScriptistä. (React 2019.; Rascia 2018; GitHub 2019.)

Yksi Reactin tärkeimmistä näkökulmista on se tosiasia, että voit luoda komponentteja. Komponentit ovat eräänlaisia luotuja, uudelleen käytettäviä HTML-elementtejä, jotka nopeuttavat ja tehostavat käyttöliittymien rakentamista. React pyrkii myös virtaviivaistamaan sitä, miten tietoja käsitellään ja tallennetaan. Tämä tapahtuu käyttämällä tiloja ja propseja. React käyttää myös elinkaarimetodeja. Komponentin elinkaari on se, missä järjestyksessä eri metodeja kutsutaan Reactissa. On olemassa esimerkiksi componentDidMount(), joka käynnistyy, kun komponentti on rakentunut ruudulle. (Rascia 2018.)

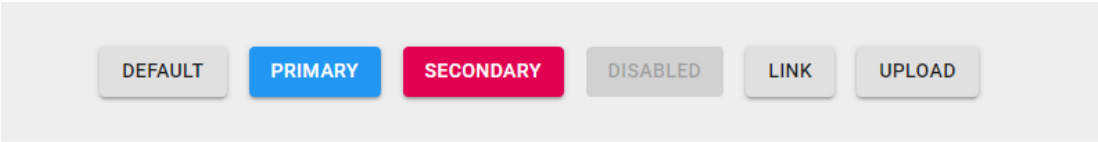
Tilat eli statet, ovat komponentin sisäistä datan käsittelyä. Tilaa voidaan ajatella datana, jota halutaan käsitellä, tallentaa ja muokata ilman, että sitä lähetetään tietokantaan. Esimerkkinä verkkokaupan ostoskärry, sinne halutaan lisätä ja poistaa tavaroita ennen tilauksen suorittamista. Props on lyhenne sanasta properties, joka tarkoittaa ominaisuuksia. Propseja käytetään datan siirtämisessä komponentista toiseen. Päätason komponentin kutsuessa alemman tason komponenttia, sille voidaan syöttää mukana propseja. Propsit voidaan nimetä haluamallaan tavalla, käyttämättä varattuja avainsanoja. Alemman tason komponentti ottaa vastaan datan ja käyttää sitä tarvitsemissaan asioissa. Näin data kulkee komponentilta toiselle. Propsien avulla voidaan lähettää dataa myös takaisin ylöspäin päätason komponentille. Propseja ei kykene muuttamaan, ne ovat vain luettavia. (Rascia 2018.) Reactin esimerkkikoodeja esittelen käytännön osassa Pikaviestintä.

4.4 Material design ja Material-UI

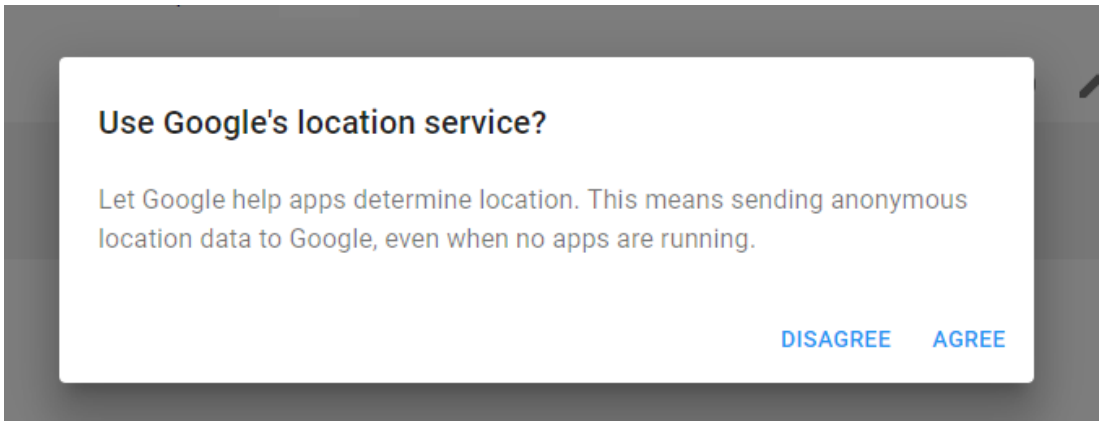
Material Design on visuaalinen kieli, joka syntetisoi klassiset periaatteet hyvästä suunnittelusta tieteen ja teknologian innoittamana. Material Designin tavoitteena on olla yksittäinen taustajärjestelmä, joka yhdistää käyttäjäkokemuksen eri laitteilla, syöttötavoilla ja alustoilla. Materiaalinen muotoilu on saanut innovaationsa fyysisestä maailmasta ja sen kuvioista, miten ne heittävät varjoja ja heijastavat valoa. (Material Design 2019.)

Material-UI eli lyhesti MUI on Reactille suunnattu käyttöliittymäkirjasto. MUI sisältää React komponentteja, jotka toteuttavat Googlen Material Designia. MUI aloitti inline-tyylillä, mutta niiden alioptimaalinen suorituskyky ja rajoitettu tuki ominaisuuksille siirsivät MUI:n käyttämään JSS:ää. JSS on siis CSS-in-JS, eli CSS-määrittelyn tiedoston alussa, josta niitä kutsutaan. Yhtenä sisäänrakennettuna ominaisuutena MUI:ssa ovat ikonit. Virallisia Material design ikoneita on yli 900. Mallikappaleet niistä löytyvät osoitteesta https://material.io/icons (Code Realm 2018.)

MUI:lla on tarjolla useita eri valmiita komponentteja, joita voi käyttää Reactissa. Niitä ovat esimerkiksi dialogit, nappulat, taulut, listat ja tekstikentät. Kuvassa 12 näemme esimerkkejä valmiista nappulakokonaisuuksista. Niitä saa valmiina eri värisinä ja määriteltynä, onko sen käyttö estetty. Kuvassa 13 on yksinkertainen käyttäjältä varmistuksen saanti dialogi. Siihen sisältyy dialogin otsikko, sisältö ja nappulat, joilla voi hylätä tai hyväksyä dialogissa esitetyn asian. (Material-UI A 2019.)



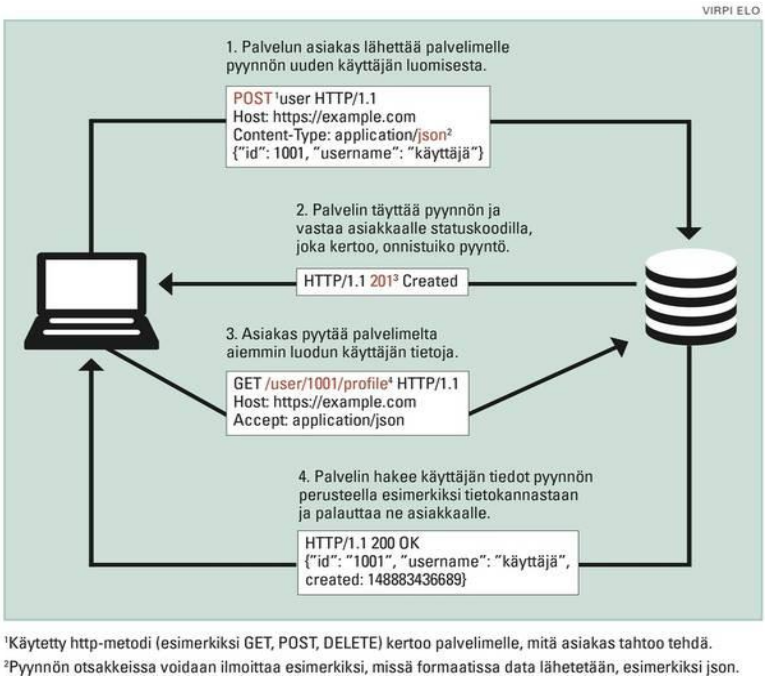
Kuva 12. Material-UI Nappuloita (Material-UI B 2019.)



Kuva 13. Material-UI dialogi (Material-UI C 2019.)

4.5 OpenAPI

OpenAPI on REST-rajapintojen kuvausformaatti. REST on nettipalveluiden yhteinen kieli. Kuvassa 14 nähdään RESTin toimintaperiaatteet. REST sisältää neljä eri metodia, ne ovat GET, POST, PUT ja DELETE. GET-metodilla haetaan tietoa API:sta. POST-metodilla lisätään tietoa API:n tietokantaan. PUT-metodilla voidaan muokata jo olemassa olevaa tietoa. DELETE-metodi poistaa kyseisen tiedon. OpenAPI tunnettiin ennen nimellä Swagger. OpenAPI:n avulla voidaan kuvata koko API:n metodit. Määritykset kirjoitetaan joko YAML- tai JSON-muodossa. YAML ja JSON ovat tekstitiedoston muotoja eli formaatteja. OpenAPI:n tarkoitus on luoda rajapintakuvaus. Tämän rajapintakuvauksen avulla front-end-kehittäjät ymmärtävät rakenteen ja voivat hyödyntää tietoa UI:n datan hakemisessa ja näyttämisessä kysymättä API:n tietoja back-end-kehittäjiltä. (Mikkonen 2017; Codeacademy 2019; Swagger 2019.)



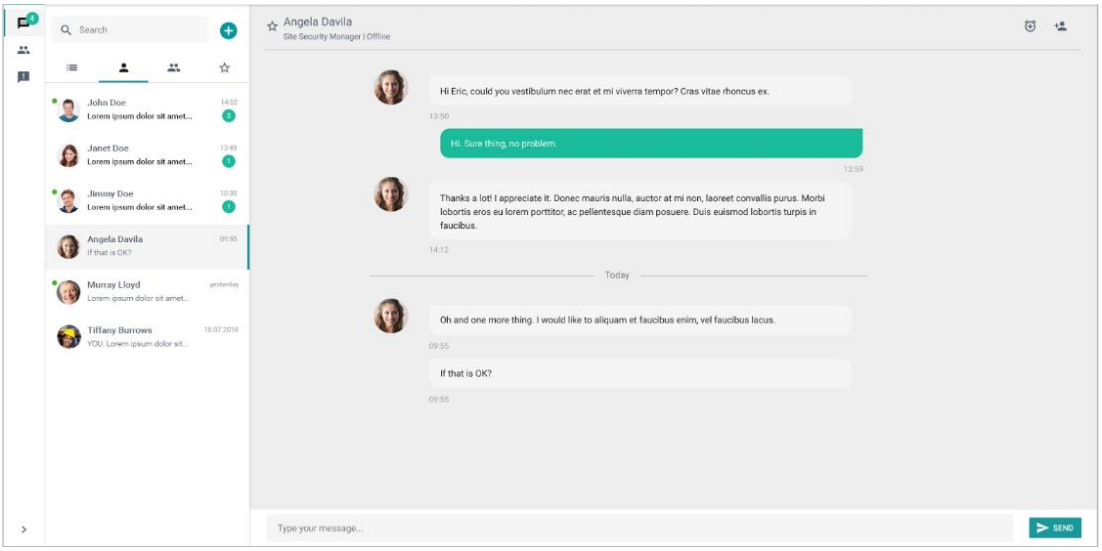
Kuva 14. RESTin toiminta (Mikkonen 2017.)

5 PIKAVIESTINTÄ- MAHDOLLISUUS WEBSOVELLUKSEEN

Idea pikaviestimen kehittämiseen tuli opinnäytetyön toimeksiantajalta, Enersense Solutions Oy:ltä. Suunnitelma pikaviestimestä oli ollut kehityslistalla jo pidemmän aikaa, mutta tekijää ja aikaa sille ei vielä ollut löytynyt. Pidimme aiheesta palaveria, jossa luotiin vaatimusmääritelmä työlle. Toteutuksena olisi front-end ja front-endiin ja back-endiin liittyvät suunnittelut. Opinnäytetyö tehdään Enersensen toiveena tästä aiheesta.

Suunnitelmana oli lisätä jo olemassa olevaan Enersensen omaan sovellukseen E-Senseen pikaviestintämahdollisuus. ”E-Sense on SaaS-mallin mukainen palvelu toimitusketjun auditointiin, materiaalihankintoihin ja rakennushankkeiden resurssien hallintaan.” (Enersense 2019) Alkuperäisenä ideana oli, että toteuttaisin ryhmäviestit ja projektikohtaiset tiedotteet. Pian työni aloittamisen jälkeen totesin, että samalla vaivalla tulee myös tehtyä yksityisviestit. Tämän työn valmistuttua kehitettäisiin myös yleiset tiedotteet. Yleiset tiedotteet tässä tarkoittaa järjestelmätiedotteita, eli tiedotteita, joita E-Sensen ylläpito voisi lähettää kaikille käyttäjille.

Kuvassa 15 näemme alkuperäisen suunnitelman miltä UI näyttäisi. Suunnitelma oli tehtynä valmiiksi Enersensen toimesta. Suunnitelma oli tehty Photoshop-ohjelmaa käyttäen, joten mitään koodillista apua siitä ei saanut.



Kuva 15. Suunnitelma viestintäosiosta.

Pikaviestintä tämän opinnäytetyön laajuudessa koskee tietokoneen ruudulle sopivaa resoluutiota eli desktop-versiota. E-Sensessä on myös tuki mobiililaitteiden resoluutioille, mutta totesimme että on parasta luoda vain desktop-versio sovelluksesta alkuunsa. Tarkoituksena on kuitenkin kehittää E-Senselle oma, itsenäinen mobiilisovellus, jonka pystyisi lataamaan mobiililaitteille riippumatta käyttöjärjestelmästä.

5.1 Front-end suunnittelu

Tärkeintä tällaisen pikaviestimen kehittämisessä on kuunnella asiakkaita ja heidän toiveitaan. Varsinkin kun sovellus luodaan tietylle käyttäjäkunnalle tiettyyn tarkoitukseen. Tästä syystä aloitin suunnittelun luomalla Google Forms- kyselyn, jossa kartoitettiin pikaviestimen ja sen ominaisuuksien tarvetta. Kysely lähetettiin yrityksen sisäisesti valituille henkilöille, jotka kuuluvat muun muassa asiakaspalveluun. Henkilöitä, jotka ovat päivittäin tekemisissä asiakkaiden neuvonnassa E-Sensen kanssa ja muutenkin käyttävät E-Senseä työssään. Kyselyyn vastasi 8 henkilöä.

Kyselyssäni selvisi toiveina olevan, että vain tietyillä henkilöillä olisi mahdollisuus luoda ryhmiä ja rajata niitä haluamallaan tavalla. Ryhmissä olisi mahdollisuus laittaa myös korkeamman tärkeyden viestejä esimerkiksi eri värillä. Näistä korkeamman tärkeyden viesteistä lähtisi myös desktop-ilmoitus. Desktop-ilmoitus tarkoittaa järjestelmän lähettämää ponnahdusikkunaa, joka käväisee ruudulla näyttäen uuden viestin saapuneen. E-Sensen yläpalkkiin tulee viestit-osio, josta näkee uusimmat viestit ja sen onko uusia viestejä saapunut. Työssä rakennan sekä yksityisviestit että ryhmäviestit. Projektitiedotteiden luominen on rajattuna vain tietyille henkilöille. Projektitiedotteita tulisi myös pystyä kommentoimaan, mutta niistä ei tulisi ilmoituksia.

Työssä tulee siis ottaa huomioon useita erilaisia asioita. Useat asioista tulisi näkyä vain tietyille henkilöille, viestinnän tulisi olla reaaliaikaista eikä vaatia sivun päivitystä ja viestit-osio olisi piilossa mobiilikäyttäjiltä. Työssä tulee siis pohdittua myös tiettyjä tietoturvaan liittyviä asioita, kuten kuinka varmistaa, että kukaan ei näe tietoa, joka on tarkoitettu jollekin muulle. Haastavan työstä tekee reaaliaikaisuus -vaatimus. Tämä johtuu sivun datan ollessa alati muuttuvaa, ei pelkästään vain se, että keskustelussa tulee uusi viesti esille. Täytyy huomioida mitkä kaikki asiat vaativat jatkuvaa päivittämistä uuden viestin tai tiedotteen saapuessa.

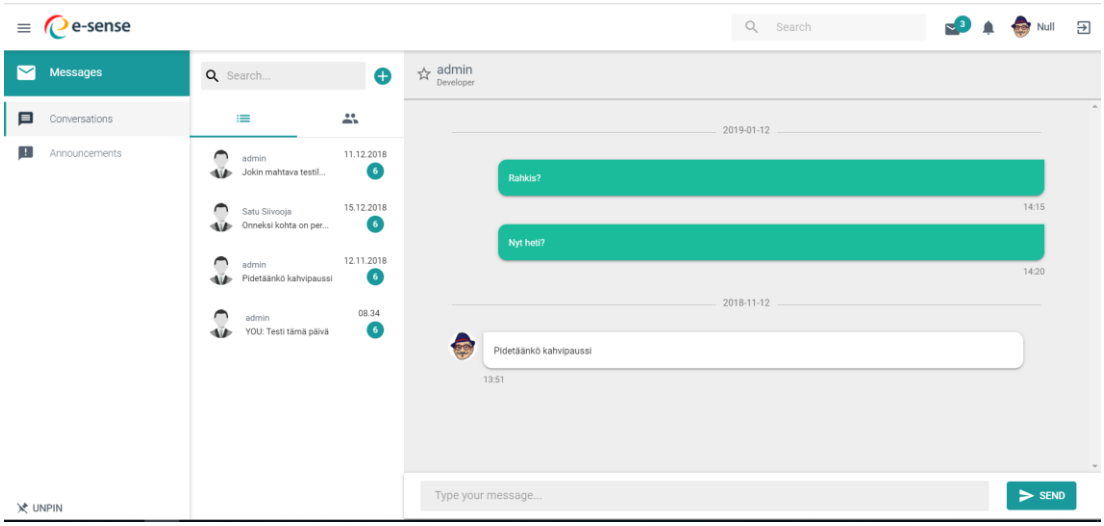
5.2 Front-end kehitys

Aloitin kehittämään UI:ta valmiina olleiden mockup-kuvien perusteella. Mockup on malliversio tuotteesta, jonka avulla voidaan esitellä ominaisuutta ennen kuin se on rakennettu. Front-end puolella ohjelmoinnissa on käytössämme JavaScriptin kirjasto, ReactJS. UI:n kehitystä helpottaa suuresti käyttämämme Reactille suunnattu kirjasto, Material-UI. MUI:n komponentteja kutsumalla saan valmiin rungon esimerkiksi nappulalle tai taulukolle. Työssäni kutsuin siis erilaisia MUI:n komponentteja, joille annoin sisällön ja tyylimäärittelyt. Tyylimäärittelyihin kuuluu esimerkiksi mihin valitsemani komponentit tulevat ja minkä värisiä ne ovat. MUI käyttää mieluiten CSS-in-JS tyylimäärittelyjä, mutta itse käytän suurimmaksi osaksi ”inline-style” tyylimäärittelyä.

Koska E-Sense on ”valmis” ohjelmisto, oli minulla käytettävissä valmiina jo sovelluksen runko. Sovelluksesta sain valmiiksi käytettyä jo sivuvalikkoa ja yläpalkkia.

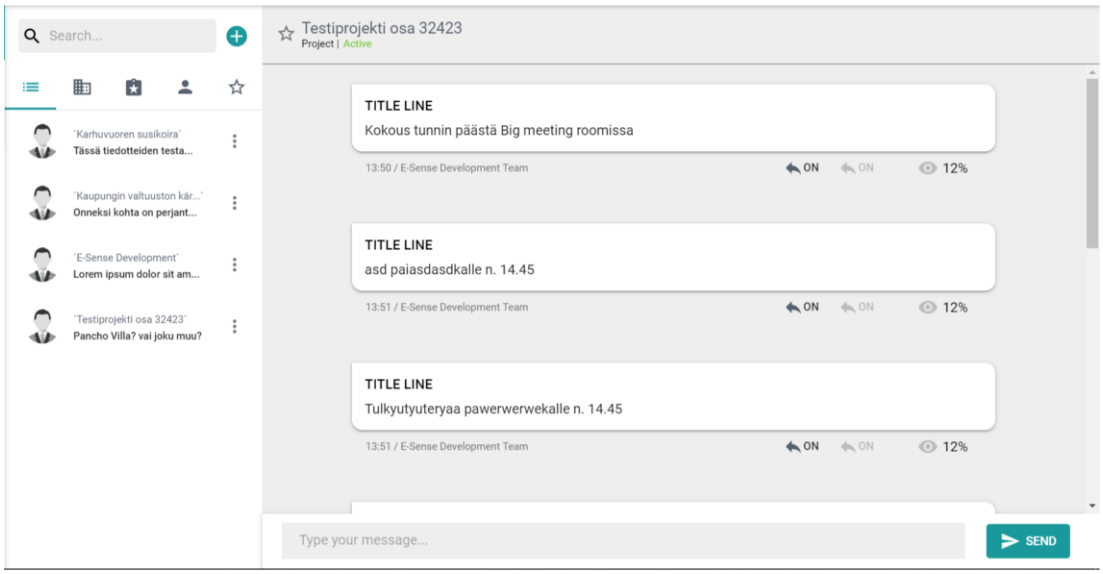
Päätason sivuvalikkoon lisäsin ensimmäisenä linkin viestintäosiolle. Alemman tason sivupalkille tuli valikko, jossa on kaksi vaihtoehtoa, viestit ja tiedotteet. Näistä aina jompikumpi on valittuna ja sen perusteella määräytyy koko muun sivun sisältö.

Kuvassa 16 näemme toteutuneen ulkonäön. Tämä on siis ohjelmoitu versio. Ylhäällä oikealla näkyy aiemmin mainitun yläpalkin viestit-osio. Kuvanottohetkellä lukemattomia viestejä on kolme. Vasemmalta oikealle katsottuna ensimmäisenä on sivuvalikko, josta valitaan, onko valittuna keskustelut vai tiedotteet. Seuraavana on keskustelut, joiden yläpuolella haku. Hakutoiminnolla pystyy hakemaan sekä ihmisiä että tiettyä tekstiä. Haun vieressä olevalla plus-ikonilla pystyy lisäämään keskustelujen puolella uuden keskustelun ja tiedotteiden puolella siitä luodaan uusi tiedote.



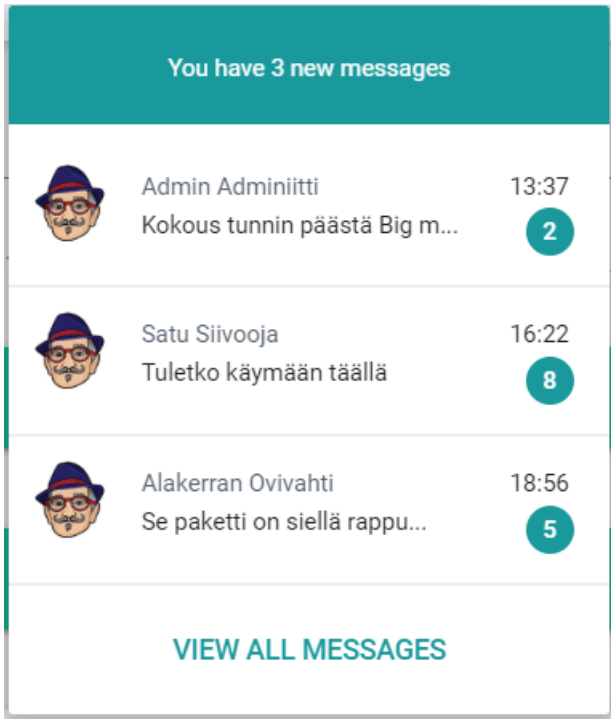
Kuva 16. Toteutunut ulkonäkö keskusteluille.

Kuvassa 17 on tiedotteiden puoli viestinnästä. Tiedotteet sisältävät otsikon ja sisällön pääasiallisesti. Tiedotteen alapuolella vasemmalla kerrotaan, milloin se on julkaistu ja kuka sen on julkaissut. Oikealla puolella voidaan vielä valita, hyväksytäänkö vastaukset ja nähdään, onko tiedotteesta lähtenyt sähköposti. Viimeisenä moniko prosentti tiedotteen vastaanottajista on lukenut viestin.



Kuva 17. Toteutunut ulkonäkö tiedotteille.

Kuvassa 18 on yläpalkista klikkaamalla auennut ponnahdusikkuna. Ponnahdusikkuna sisältää ilmoituksen uusien viestien lukumäärästä. Kuvassa tällä hetkellä näytetään 3 eri keskustelua, joissa on uusia viestejä. Keskustelujen määrä ponnahdusikkunassa ei kuitenkaan rajoitu kolmeen.



Kuva 18. Yläpalkista aukeava ponnahdusikkuna.

Sivuvalikkojen jälkeen loin uuden komponentin, joka pitää sisällään kaiken muun, mitä sivu pitää sisällään. Tälle kaiken sisältävälle sivulle tuli kaksi (2) muuta ylätason komponenttia.

Ensimmäinen näistä pitää sisällään sivun vasemman laidan, johon sisältyy viestintäosiossa avoimet keskustelut ja valikon, josta saa valikoitua näytetäänkö kaikki, ryhmät, yksityisviestit vai tähdellisiksi merkatut keskustelut. Lisäksi vasemmasta laidasta löytyy hakutoiminto ja uuden keskustelun luonti. Tiedotteiden ollessa valittuna näytetään listaus projekteista, joissa on tiedotteita.

Toinen komponentti pitää sisällään oikean puolen sivusta. Tällä puolella olisi keskustelu tai tiedote, oletuksena aina uusin keskustelu on avattuna. Keskustelun/tiedotteen ollessa valittuna sivun ylälaita vaihtelee, riippuen kumpi on valittuna.

Kuvassa 19 näemme kutsun komponentille ”MessageWrapper”, joka sitoo sekä keskustelunvalintaosion (MessageAreaWrapper) että keskusteluosion (MessageAreaContent). Kuvassa 20 näkyy CSS määrittelyjä näille kahdelle osiolle ja kuinka ne sijoittuvat ”MessageWrapper”-komponentissa.

```
if (this.props.userId) {
    return (
        <div style={{width: '100%'}}>
            <MessageWrapper
                leftContent={
                    <MessageAreaWrapper
                        data={this.state.mockupMessages}
                        onSelectMessage={(event, id) => this.handleOnSelectMessage(id)}
                        onSearch={(value) => console.log('hakusana: ', value)}
                        userId={this.props.userId}
                    />}
                rightContent={
                    <MessageAreaContent
                        data={this.state.mockupSelectedMessage}
                        onTextFieldBlur={(value) => this.onTextFieldBlur(value)}
                        onSend={() => console.log(this.state.messageValue)}
                    />}
            />
        </div>
    )
} else {
    return (<div>Something is not quite right</div>)
}
```

Kuva 19. Keskustelunvalinta-palkin ja keskusteluosion sitova komponentti.

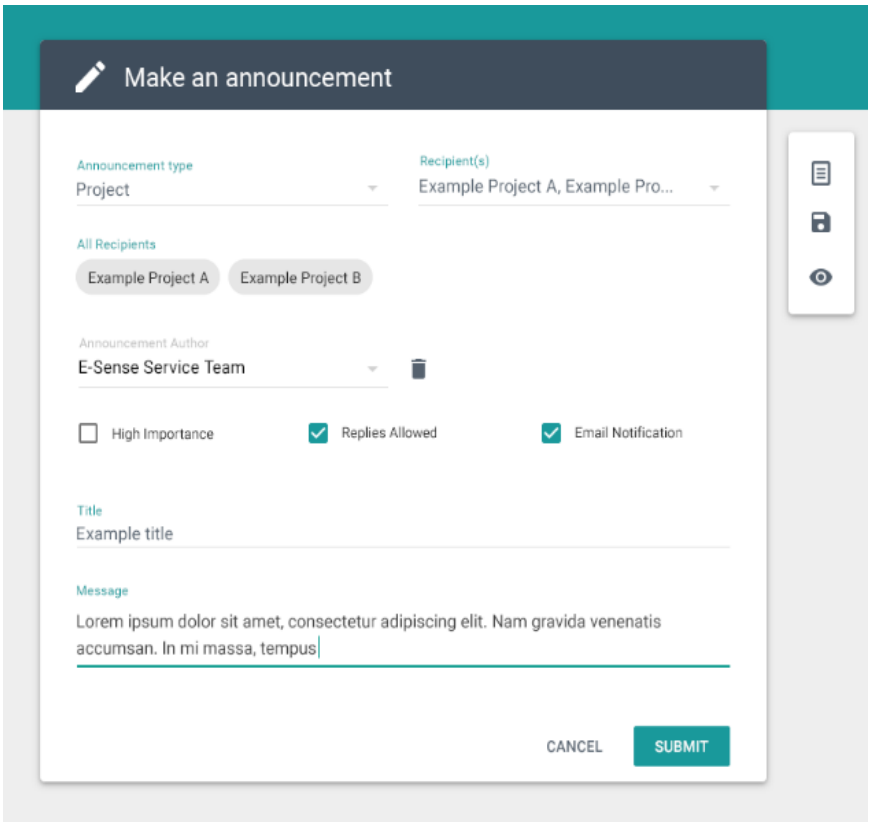
```
<div style={{width: '100%', height: 'calc(100vh - 132px)'}}>
    <div style={{display: 'inline-block', width: '300px', backgroundColor: esenseCanvasColor, height: '100%', verticalAlign: 'top'}
        {this.props.leftContent}
    </div>
    <div style={{display: 'inline-block', width: 'calc(100% - 300px)', height: '100%', backgroundColor: esenseGrey200}}>
        {this.props.rightContent}
    </div>
</div>
```

Kuva 20. MessageWrapper -komponentin tietoa vastaanottava koodi.

Keskusteluissa tekee muutoksen myös se, onko yksityis- vai ryhmäkeskustelu valittuna. Yksityiskeskustelussa yläreunassa näytetään, onko keskustelu tähdellinen, kenen kanssa juttelee, mikä on hänen roolinsa/työnimikkeensä ja onko henkilö paikalla sillä hetkellä. Ryhmäkeskusteluissa yläpalkissa on keskustelun tähdellisyys, ryhmän nimi ja ryhmään kuuluvien henkilöiden listaus.

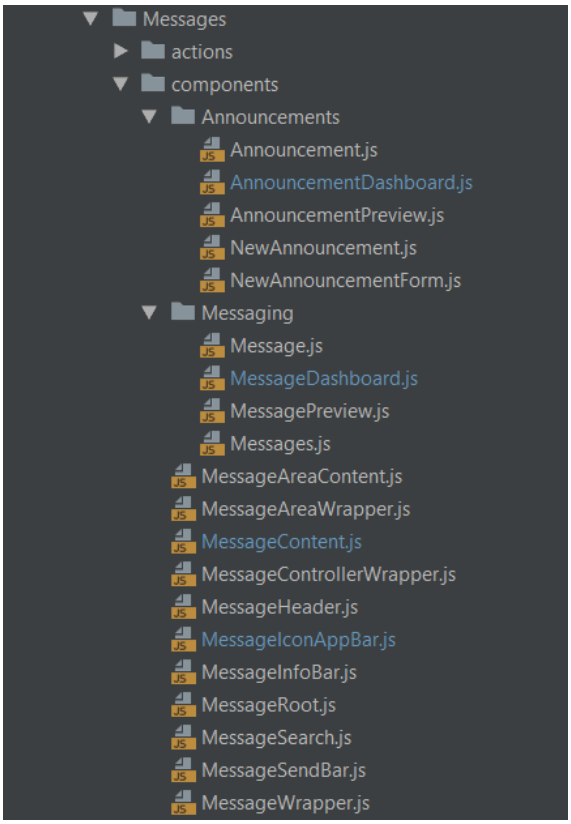
Tiedotteissa yläreunasta löytää tähdellisyyden, projektin nimen, tiedon, onko projekti aktiivinen vai lopetettu/alkamassa, tiedotteiden kokonaismäärän ja henkilömäärän, jota tiedotteet koskevat. Yläpalkki tehtiin myös tulevaisuutta varten valmiiksi, jos tulee useamman tyyppisiä kuin projektitiedotteita. Yläpalkista kykenee näyttämään myös siis tiedon, minkä tyyppinen tiedote-'keskustelu' on, esimerkiksi projekti, yritys vai järjestelmätiedote.

Seuraavana haasteena oli luoda uuden tiedotteen luontilomake. Tässäkin olisi monta muuttuvaa tekijää, jotka vaikeuttavat lomakkeen luontia ja kuinka sitä käsitellään. Kuvassa 21 nähdään uuden tiedotteen luonnin monimuotoisuus. Valmiiksi rakennettu tiedotteen tyyppi. Tämän hetkisessä tilanteessa ei ole vielä muita kuin projektitiedotteet, mutta tulevaisuutta varten mahdollisuus lisätä vaihtoehtoja. Seuraavana valitaan tiedotteen vastaanottajat ja ne näkyvät vielä eriteltyinä valinnan alapuolella. Sitten valitaan tiedotteen omistaja. Kolmen valintaruudun avulla valitaan tiedotteen tärkeys, sallitaanko vastaukset ja lähetetäänkö sähköposti vastaanottajille. Tämän jälkeen onkin jo helppo osio. Valitaan tiedotteelle otsikko ja kirjoitetaan sen sisältö.



Kuva 21. Uuden tiedotteen luontilomake.

Kuvassa 22 nähdään kuinka monta komponenttia loin saadakseni kaiken tarpeellisen näkymään. Jokainen tiedosto on siis yksi komponentti. Koodirivejä kyseisissä komponenteissa on noin 3000. Järjestelin vain tiedotteita tai keskusteluita koskevat komponentit omiin kansioihinsa. Yleisessä ”components” kansiossa ovat yleiskäyttöiset komponentit, joita molemmat puolet käyttävät.



Kuva 22. Viestintäosion komponentit.

5.3 Back-end

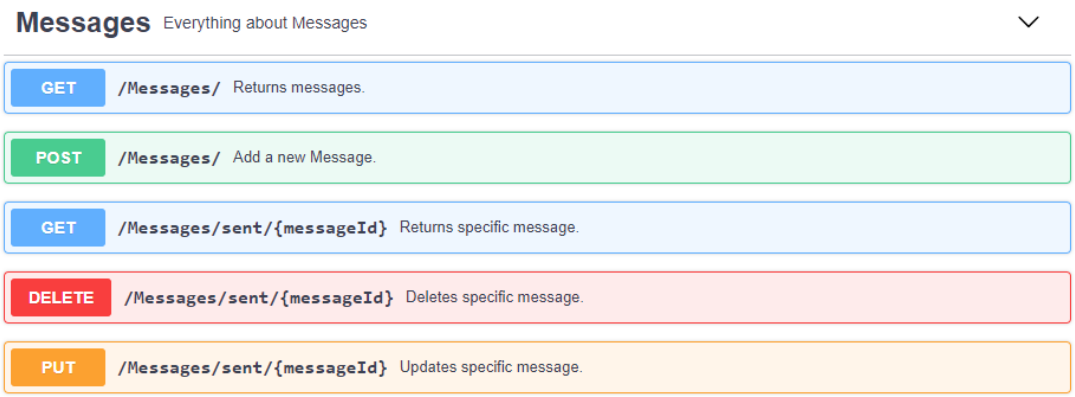
Back-end-puolelta toteutettiin ainoastaan suunnitelmat, millainen rakenne API:lla olisi. Aloitin tutkimalla olemassa olevaa API:n rakennetta, josta otin viittauksia viestintäosioon. Tämän jälkeen sovittiin, että teen OpenAPI spesifikaation rakenteesta ja dataobjektien luokkamallin eli UML-kaavion. Tässä opinnäytetyössä näytetään vain alustavia suunnitelmia, koska lopullinen rakenne halutaan pitää salaisena.

Kuvassa 23 näemme luomani OpenAPI spesifikaatiosta sovellukselle olennaisin GET-metodi. Tällä metodilla haetaan tietokannasta kaikki viestit, jotka liittyvät kirjautuneeseen käyttäjään. Mahdollista on myös hakea annetun käyttäjätunnisteen avulla viestejä. Nämä viestit liittyvät myös siis kirjautuneeseen henkilöön, mutta sen avulla haetaan tietty viestiketju. Haun onnistuessa API palauttaa vastauksen, jossa lukee ”200: OK”. Jos haetuilla tiedoilla ei löydy viestejä, API palauttaa ”404: Not Found”. Numerosarjat ovat HTTP kutsujen statuskoodeja. Kaikki koodit löytyvät muun muassa sivulta: https://www.restapitutorial.com/httpstatuscodes.html.

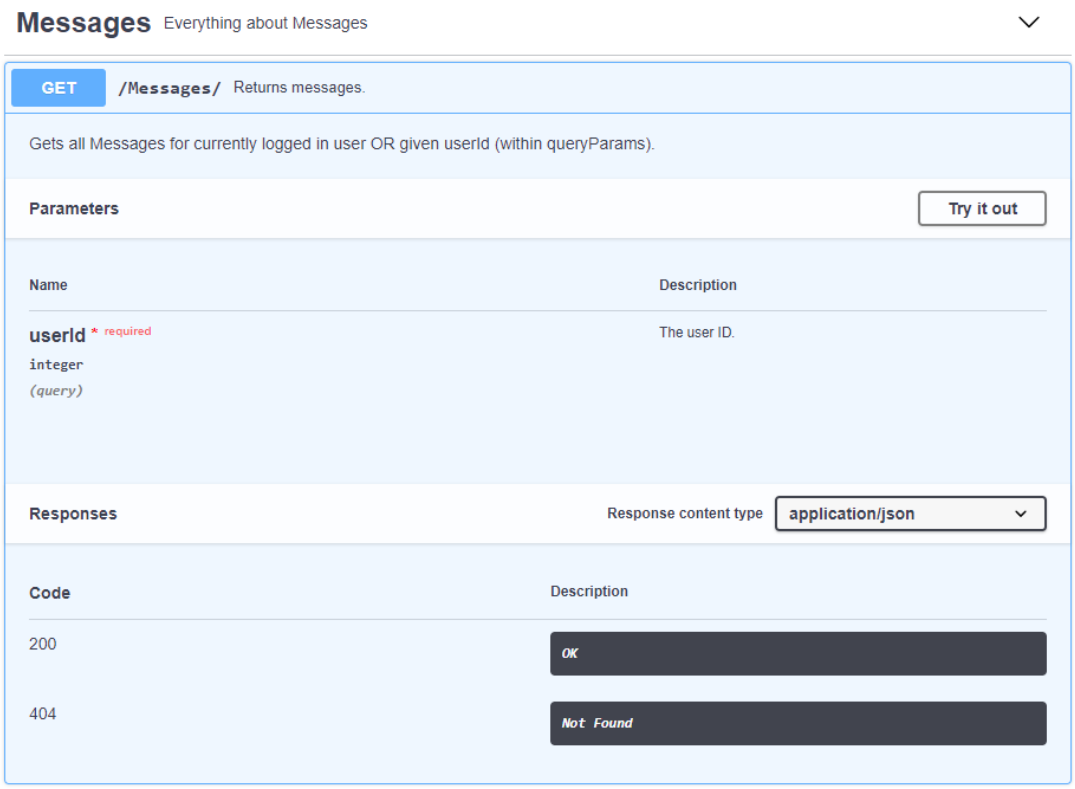
```yaml
/Messages/:
  get:
    operationId: getSentMessages
    tags:
    - Messages
    summary: Returns messages.
    description: Gets all Messages for currently logged in user OR given userId (within queryParams).
    produces:
      - application/json
    parameters:
    - in: query
      name: userId
      type: integer
      required: true
      description: The user ID.
    responses:
      200:
        description: OK
      404:
        description: Not Found
```

Kuva 23. OpenAPI spesifikaation syntaksia.

Kuvassa 24 näemme ensimmäiset viisi kutsua. Näihin viiteen kutsuun sisältyy kaikki 4 mahdollista metodia, jotka REST API:lla on. Kuvassa esitetään metodit viestien hakemiselle ja yksittäisen viestin hakemiselle, lisäämiselle, muokkaamiselle ja poistamiselle. Kun käsitellään yksittäistä viestiä, lähetetään pyynnössä mukana viestin tunniste. Tunnisteet ovat yleensä numeroita, tekstiä tai niiden sekoitusta. Kuvassa 25 tarkastellaan tarkemmin kutsua, jossa haetaan kaikki viestit. Kuvassa näkyy pyynnön tiedot, eli parametrit yksilöityinä. Kuvan tapauksessa niitä on vain yksi, käyttäjän tunniste. Kuvan alalaidassa näkyvät mahdolliset vastaukset pyyntöön. Tässä tapauksessa niitä on kaksi, ”OK” tai ”Ei löydy”. Ylälaidan nappulasta ”Try it out”, pystyisi testaamaan pyyntöjä, jos API olisi rakennettuna taustalla.

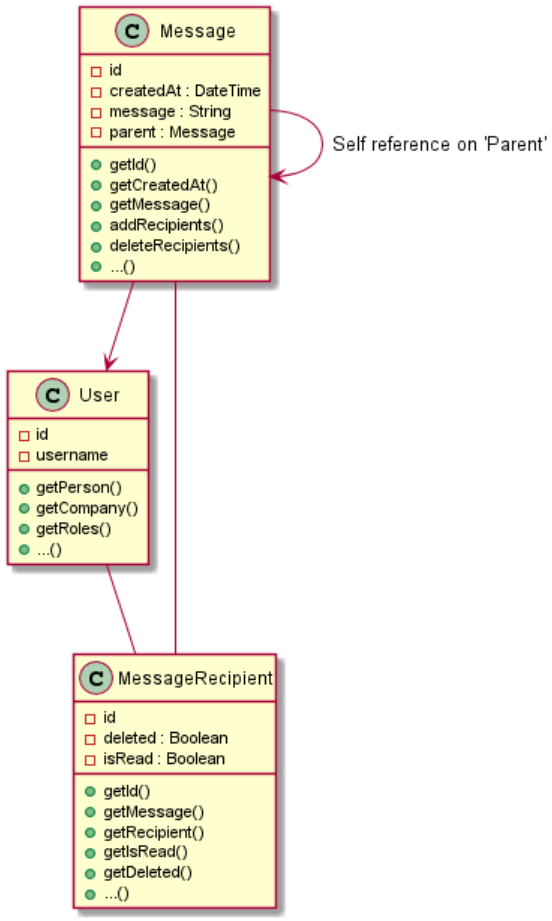


Kuva 24. OpenAPI kuvauksella esitetty API:n ensimmäiset viisi pyyntöä.



Kuva 25. OpenAPI yksittäisen metodin tarkempi katselmointi.

OpenAPI spesifikaation lisäksi tein dataobjektien luokkamallista UML-kaavion. Kaavio on vain alustava suunnitelma tulevasta. Kuvassa 26 näkyy kaavio. Siinä näemme tärkeimmät kolme osapuolta viestittelystä. Nämä kolme ovat: viesti, käyttäjä ja viestin saaja. Kuvasta näemme, että jokaisella näistä on id, eli tunniste. Mahdolliset viestikumppanit määräytyvät niihin projekteihin ja yrityksiin, joissa käyttäjä on. Viesti saa tiedoikseen luontiajan, kuka on luonut ja ketkä siihen viestiin kuuluvat. Viesti voidaan myös määrittää luetuksi ja poistetuksi.



Kuva 26. Alustava suunnitelma dataobjektien luokkamallista

6 KEHITYSMAHDOLLISUUDET

Tulevaisuudessa sovellukseen voidaan kehittää API puoli kokonaisuudessaan. Tämän jälkeen se on käyttökelpoinen toteutus, jota voidaan alkaa hyödyntämään kokonaisuudessaan. API:n valmistuessa sitä voidaan myös käyttää muistakin käyttöliittymistä. Esimerkiksi voidaan rakentaa mobiilisovelluksia Androidille ja IOS:lle. Samaa API:a voi siis käyttää selaimien lisäksi mobiililaitteet. Mobiililaitteiden sovellukset tietenkin vaativat hieman erilaisten kielien käyttöä kuin selainpohjaiset sovellukset.

Olemassa olevan käyttöliittymään voidaan lisätä erinäköisiä mahdollisuuksia kommunikoinnille. Todennäköistä on, että sovellukseen lisätään järjestelmätiedotteet. Järjestelmätiedotteiden avulla voidaan ilmoittaa käyttäjille uusista ominaisuuksista ja saapuvista päivityksistä.

7 LOPPUSANAT

Web-sovelluksen kehittäminen on loppupeleissä paljon haastavampaa kuin miltä se saattaa aluksi kuulostaa. Jo aloittaessani kehittämään UI:n koodia, huomasin että en ollut kerennyt ajattelemaan puoliakaan asioista mitä se vaatii. Tämän opinnäytetyön teoriaosiota kirjottaessani opin vielä paljon tutkimistani asioista. Käytän tässä opinnäytetyössä mainittuja asioita työssäni. Pohjatietoni olivat suhteellisen hyvät, koska olin jo oppinut suuren osan asioista töitä tehdessäni.

Kuten Scrummista mainitsin aiemmin, se on helppo ymmärtää, mutta vaikea hallita. Tämä pätee myös omalla osallani. Vaikka olinkin jo työssäni käyttänyt Scrummin tekniikkaa, opin siitä vielä paljon uutta ja lähenin sen hallitsemista. Hyvistä ohjelmointikäytännöistä on hyötyä myös omassa työssäni. Olen myös huomannut että hyvistä ohjelmointikäytännöistä tulee lipsuttua, jos kukaan ei tarkista koodia ja sen laatua. Väsyneenä minkään tekeminen ei suju kovinkaan hyvin. Tämän olen huomannut myös muissa omissa projekteissani, kun olen kirjoittanut jotain väsyneenä illalla. Seuraavana aamuna kun jatkanut työtäni, olen joutunut hetken aikaa pohtimaan mitä ja miten olen oikein kirjoittanut.

Tulevaisuudessa aion vielä jatkaa web-ohjelmoinnin parissa. Mielestäni ala on mielenkiintoinen ja antaa teknologioiden kehittyessä myös haastettakin. Web-ohjelmoinnin lisäksi olen ajatellut, että voisin kokeilla peliohjelmointia. Peliohjelmointi on osittain samaa mutta suuressa osin erilaista kuin web-sovellusten kehittäminen.

LÄHTEET

Ahmed, K. 2018. Developer Roadmap. Viitattu 11.3.2019. Saatavissa: https://github.com/kamranahmedse/developer-roadmap/blob/master/images/intro.png

Atlassian A. Making a Pull Request. Viitattu 12.3.2019. Saatavissa: https://www.atlassian.com/git/tutorials/making-a-pull-request

Atlassian B. Jira. Viitattu 12.3.2019. Saatavissa: https://www.atlassian.com/software/jira

B. 2018. What is HTML? The Basics of Hypertext Markup Language Explained. Viitattu 13.3.2019. Saatavissa: https://www.hostinger.com/tutorials/what-is-html

Beal, V. A. High-Level Language. Viitattu 16.3.2019. Saatavissa: https://www.webopedia.com/TERM/H/high_level_language.html

Beal, V. B. DevOps – Development and Operations. Viitattu 16.3.2019. Saatavissa: https://www.webopedia.com/TERM/D/devops_development_operations.html

Bitbucket. Product. Viitattu 12.3.2019. Saatavissa: https://bitbucket.org/product

Bradford, L. 2018. The Skills You Need to Be a Back-End Developer. Viitattu 22.2.2019. Saatavissa: https://www.thebalancecareers.com/the-skills-you-need-to-be-a-backend-developer-2071184

Crawford, S. How HTML5 Works. Viitattu 13.3.2019. Saatavissa: https://computer.howstuffworks.com/html-five3.htm

Codeacademy. What is REST? Viitattu 18.3.2019. Saatavissa: https://www.codecademy.com/articles/what-is-rest

Code Conquest. Introduction to Web Development. Viitattu 22.2.2019. Saatavissa: https://www.codeconquest.com/what-is-coding/web-programming/

Code Realm. Meet Material-UI – Your New Favorite User Interface Library. Viitattu 13.3.2019. Saatavissa: https://medium.freecodecamp.org/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c

Enersense. E-Sense – Intelligent Platform. Viitattu 13.3.2019. Saatavissa: https://www.enersense.fi/fi/e-sense

Gazarov, P. 2016. What is an API? In English, please. Viitattu 16.3.2019. Saatavissa: https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82

Git A. Getting Started – About Version Control. Viitattu 12.3.2019. Saatavissa: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Git B. Getting Started – What is Git? Viitattu 16.3.2019. Saatavissa: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Gellert, L. 2012. What is a Full Stack Developer? Viitattu 22.2.2019. Saatavissa: https://www.laurencegellert.com/2012/08/what-is-a-full-stack-developer/

Goncalves, L. 2019. What is Agile Methodology. Viitattu 11.3.2019. Saatavissa: https://luis-goncalves.com/what-is-agile-methodology/

Heller, M. 2017. What is Jenkins? The CI server explained. Viitattu 12.3.2019. Saatavissa: https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html

Kadivar, N. 2018. Top 10 Version Control Systems. Viitattu 11.3.2019. Saatavissa: https://hackernoon.com/top-10-version-control-systems-4d314cf7adea

King, R. 2018. What is a Web Developer. Viitattu 22.2.2019. Saatavissa: https://www.bitdegree.org/tutorials/what-is-a-web-developer/

Liu, T. 2017. 6 Essential Tips on How to Become a Full Stack Developer. Viitattu 11.3.2019. Saatavissa: https://hackernoon.com/6-essential-tips-on-how-to-become-a-full-stack-developer-1d10965aaead

MDN web docs. What is JavaScript? Viitattu 13.3.2019. Saatavissa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Material Design. Introduction. Viitattu 13.3.2019. Saatavissa: https://material.io/design/introduction/#

Material-UI A. Viitattu 14.3.2019. Saatavissa: https://material-ui.com/

Material-UI B. Buttons. Viitattu 14.3.2019. Saatavissa: https://material-ui.com/demos/buttons/

Material-UI C. Dialogs. Viitattu 14.3.2019. Saatavissa: https://material-ui.com/demos/dialogs/

Mikkonen, J. 2017. Rest on Nettipalveluiden Yhteinen Kieli. Viitattu 18.3.2019. Saatavissa: https://www.tivi.fi/Kaikki_uutiset/rest-on-nettipalveluiden-yhteinen-kieli-6652501

Morris, S. 2018. Tech 101: The Ultimate Guide to CSS. Viitattu 13.3.2019. Saatavissa: https://skillcrush.com/2012/04/03/css/

Muhammad, F. 2018. How to Create a High-Converting HTML Landing Page. Viitattu 13.3.2019. Saatavissa: https://instapage.com/blog/html-landing-page-generator

Nikishaev, A. 2017. 13 Simple Rules for Good Coding. Viitattu 13.3.2019. Saatavissa: https://hackernoon.com/few-simple-rules-for-good-coding-my-15-years-experience-96cb29d4acd9

Power, V. 2016. What is a Continuous Integration and Delivery Pipeline, and Why Is it Imporant? Viitattu 16.3.2019. Saatavissa: https://codefresh.io/continuous-integration/continuous-integration-delivery-pipeline-important/

Rascia, T. 2018. Getting Started with React – An Overview and Walkthrough. Viitattu 13.3.2019. Saatavissa: https://www.taniarascia.com/getting-started-with-react/

React. A JavaScript Library For Building User Interfaces. Viitattu 13.3.2019. Saatavissa: https://reactjs.org/

Rouse, M. IT Operations. Viitattu 16.3.2019. Saatavissa: https://searchitoperations.techtarget.com/definition/IT-operations

Sacolick, I. 2018. What Is Agile Methodology? Modern Software Development Explained. Viitattu 16.3.2019. Saatavissa: https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html

Scrum A. What Is Scrum? Viitattu 11.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-scrum

Scrum B. What is a Sprint in Scrum? Viitattu 11.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-a-sprint-in-scrum

Scrum C. What is sprint Planning? Viitattu 11.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-sprint-planning

Scrum D. What is a Daily Scrum? Viitattu 12.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-a-daily-scrum

Scrum E. What is a Sprint Review? Viitattu 12.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-a-sprint-review

Scrum F. What is a Sprint Retrospective? Viitattu 12.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-a-sprint-retrospective

Scrum G. What is a Product Backlog? Viitattu 12.3.2019. Saatavissa: https://www.scrum.org/resources/what-is-a-product-backlog

Shannon, R. 2012. What is HTML? Viitattu 18.3.2019. Saatavissa: https://www.yourhtmlsource.com/starthere/whatishtml.html

Smashing Magazine. 2011. Responsive Web Design – What It Is and How to Use It. Viitattu 16.3.2019. Saatavissa: https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/

Sridhar, A. 2018. An Introduction to Git: What It Is: and How to Use It. Viitattu 12.3.2019. Saatavissa: https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61

Suojanen, J. 2015. Mitä Hakukoneoptimointi (SEO) on? Viitattu 22.2.2019. Saatavissa: https://www.digimarkkinointi.fi/blogi/mita-hakukoneoptimointi-seo-on

Swagger. What is OpenAPI? Viitattu 18.3.2019. Saatavissa: https://swagger.io/docs/specification/about/

Techopedia A. Web Development. Viitattu 15.3.2019. Saatavissa: https://www.techopedia.com/definition/23889/web-development

Techopedia B. Back-End Developer. Viitattu 22.2.2019. Saatavissa: https://www.techopedia.com/definition/29568/back-end-developer

ThoughtWorks. Continuous Integration. Viitattu 12.3.2019. Saatavissa: https://www.thoughtworks.com/continuous-integration

w3schools A. Web Development Roadmaps. Viitattu 22.2.2019. Saatavissa: https://www.w3schools.com/whatis/

w3schools B. HTML Responsive Web Design. Viitattu 16.3.2019. Saatavissa: https://www.w3schools.com/html/html_responsive.asp

w3schools C. What is JavaScript? Viitattu 18.3.2019. Saatavissa: https://www.w3schools.com/whatis/whatis_js.asp