React-kehittäjän päiväkirja

Niko Lehtiniemi

Haaga-Helia ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Amk-opinnäytetyö
2022

Tiivistelmä

Tekijä(t)
Niko Lehtiniemi

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
React-kehittäjän päiväkirja

Sivu- ja liitesivumäärä
53 + 0

Päiväkirjaopinnäytetyön tavoitteena on seurata opinnäytetyöntekijän kehitystä työroolissaan ohjelmisto- ja käyttöliittymäkehittäjänä. Käyttöliittymäkehittäjän työtehtäviä seurataan kahdeksan raportointiviikon ajan. Seurantajakso liittyy ensimmäisen raportointiviikon aikana alkavaan puoli vuotta kestävään ohjelmistoprojektiin.

Seurantajakson raportointiviikoilla on otsikoitu teema, johon keskitytään kyseisellä raportointiviikolla ja erityisesti sen viikkoanalyyseissa. Analysoitavat teemat syntyvät sekä meneillä olevan viikon että edeltävien viikkojen työtehtävien pohjalta. Viikkoanalyysit ja teemat keskittyvät keskeisimpiin käyttöliittymäkehittäjän työtehtäviin, kuten React-komponenttien ja yksikkötestien kirjoittamiseen.

Seurantajakso sijoittuu tekijän töihin paluuseen yli vuoden kestäneeltä opintovapaalta. Opintovapaan aikana opinnäytetyöntekijän työrooli oli muutettu yhteisymmärryksessä työnantajan kanssa verkkosivukehittäjästä käyttöliittymäkehittäjäksi.

Ensimmäiset viisi raportointiviikkoa vastaavat projektin ensimmäistä viittä viikkoa. Viimeiset kolme raportointiviikkoa sijoittuvat projektin loppupuolelle, eivätkä ne seuraa suoraan toisiaan, vaan pitävät kaikki välissään useita raportoimattomia viikkoja. Raportointiviikot valittiin mahdollisimman kattavan kuvan saamiseksi tekijän kehittymisestä.

Päiväkirjaopinnäytetyö toteutetaan kehitysyhtiössä X, joka on erikoistunut kiinteistö- ja rakennusalan ohjelmistoihin. Opinnäytetyöhön liittyvän ohjelmistoprojektin tavoite on kiinteistöjen ja rakennusten kunnon ja korjausten mallintaminen. Ohjelmistoprojekti liittyy puolestaan laajempaan ohjelmistokokonaisuuteen ja toteutettava moduuli on yksi ohjelmistokokonaisuuden noin kymmenestä vastaavasta.

Päiväkirjatyyppisenä opinnäytetyönä kyseessä on laadullinen tutkimus ja tutkimusmenetelmä on autoetnografinen. Opinnäytetyön aineiston muodostaa päiväkirjaraportointi. Raportoituihin teknologioihin ja menetelmiin liittyvät kirjoitukset muodostavat opinnäytetyön keskeiset lähteet. Raportointia edeltävät kolme ensimmäistä lukua esittelevät opinnäytetyön tavoitteet, ympäristön ja menetelmät.

Työ luo realistisen kuvan alalla vaadittavasta osaamisesta ja auttaa hahmottamaan ohjelmistoja käyttöliittymäkehityksessä yleisesti käytössä olevia teknologioita ja toimintatapoja. Ohjelmistoprojektin kesto sekä raportointiviikkojen hajautus projektin alku- ja loppupäähän antavat kattavan kuvan tekijän kehittymisestä työtehtävissään.

Asiasanat
ohjelmistokehitys, käyttöliittymä, ohjelmointirajapinnat, yksikkötestaus

Sisällys

Käsitteet

Repository, repo	Ohjelmiston lähdekoodin ja tiedostojen säilytyspaikka. Synonyymeja suomeksi mm. tietovaranto, pakettivarasto, ohjelmalähde. Sekä suomeksi että englanniksi keskustellessa käytetään yleensä lyhennettä repo.
Git-versionhallinta, Git, versionhallinta	Git on lähes versionhallinnan synonyyminä käytetty versionhallintaohjelmisto. Se kirjaa ohjelmistokehityksen muutoshistorian ja mahdollistaa eri versioiden vertailun ja niiden välillä liikkumisen. Yleisesti käytössä ohjelmistokehityksessä.
PR, Pull Request	Muutospyyntö. Versionhallinnan termi muutosten ehdottamiselle repositoryn tiedostoihin.
Scrum	Projektinhallinnan viitekehys iteratiiviseen kehitystyöhön. Käytössä yleisesti ketterän kehityksen ohjelmistoprojekteissa.
Kirjasto, apukirjasto	Resurssikokoelma, joka liitetään osaksi ohjelmistoa. Apukirjasto helpottaa ohjelmistokehittäjän työtä sisältämällä valmiita resursseja, kuten valmista ohjelmistokoodia.
Tausta, BE	Backend, ohjelmiston palvelinpuoli, taustaohjelmisto. Käyttäjälle näkymätön osa ohjelmistoa, joka sisältää ohjelmiston datan ja toimintalogiikan.
Käyttöliittymä, FE	Frontend, ohjelmiston käyttöliittymä ja käyttäjälle näkyvä osa. Käyttöliittymällä käyttäjä kertoo taustalle mitä haluaa ohjelmiston tekevän.
Ohjelmointirajapinta, API	Rajapinta, jonka avulla eri ohjelmat voivat vaihtaa tietoja keskenään. Tyypillisesti API:lla viitataan rajapintaan, joka mahdollistaa käyttöliittymän ja taustan välisen kommunikoinnin.

REST-rajapinta, REST-API	API-arkkitehtuurityyli käyttöliittymän ja taustan väliseen kommunikointiin. RESTful-filosofia sovelletussa muodossaan ohjelmistokehityksessä: 1) API:lla on lähdeosoite, 2) se käyttää standardeja HTTP-pyyntöjä ja 3) määrittelee kommunikoinnissa käytetyn tiedostomuodon.
JSON	Kirjainlyhenne sanoista JavaScript Object Notation. Yleinen tiedostomuoto käyttöliittymän ja taustan välisessä kommunikaatiossa.
CRUD	Kirjainlyhenne sanoista “Create, Read, Update, Delete”. Yleisiä REST-rajapintaoperaatioita datan luomiseen, lukemiseen, päivittämiseen ja poistamiseen.
HTTP	Selainpohjaisten käyttöliittymien käyttämä protokolla kommunikointiin ohjelmiston taustan kanssa. Tyypillisiä HTTP-metodeja ovat GET (hae), POST (lisää), PUT (muokkaa) ja DELETE (poista).
OpenAPI-määritykset, Swagger-tiedostot	Virallisesti OpenAPI Specification, ent. Swagger Specification. REST-rajapinnan määritykset ohjelmistokehittäjien noudatettaviksi ja kehitystyön avuksi.
MFE, Micro-Frontend Architecture	Käyttöliittymäkehityksen arkkitehtuurityyli, jossa erilliset mikro-käyttöliittymät muodostavat yhdessä koko ohjelmiston käyttöliittymän.
DOM	Kirjainlyhenne sanoista Document Object Model, suomeksi dokumenttioliomalli. Ohjelmointirajapinta selainpohjaisten käyttöliittymien ohjelmointiin.
JavaScript, JS	Yksi verkkosivujen ydinteknologioista HTML:n (verkkosivujen rakenne) ja CSS:n (verkkosivujen ulkoasu) lisäksi. Käytetään selainpohjaisten käyttöliittymien ohjelmoinnissa DOM:n HTML-elementtien ja CSS-tyylien manipuloimiseen sekä HTTP-pyyntöjen tekemiseen.

React	JavaScriptin apukirjasto käyttöliittymien kehittämiseen.
TypeScript, TS	JavaScriptin ylijoukko, nk. staattisesti tyypitetty JS. Sisältää kaikki JavaScriptin toiminnallisuudet ja lisäksi omia vain TS:lle ominaisia toiminnallisuuksia.
Yksikkötesti	Yksittäiseen käyttöliittymäkomponenttiin liittyvä testi. Yksikkötestien on tarkoitus varmistaa, että ohjelmiston komponentit toimivat halutusti, ja että muutoksia tehtäessä kaikki komponentit jatkavat toimimista halutusti.

1 Johdanto

Opinnäytetyöpäiväkirja sijoittuu sen tekijän paluuseen työelämään yli vuoden kestäneeltä opintovapaalta. Ennen töihin paluuta opinnäytetyöntekijän työtehtävät oli muutettu yhteisymmärryksessä työnantajan kanssa verkkosivukehittäjän töistä käyttöliittymäkehittäjän työtehtäviin. Työnantaja on kehitysyhtiö, joka tuottaa ammattilaisohjelmistoja rakennussuunnittelun ja kiinteistönhallinnan tarpeisiin. Opinnäytetyössä käsiteltävä ohjelmistoprojekti keskittyy kiinteistöjen korjausohjelmien mallintamiseen. Päiväkirjanpitäjän toimenkuva on mainitun ohjelmiston käyttöliittymän toteuttaminen osana kolmehenkistä käyttöliittymäkehittäjätiimiä ja n. tusinan hengen ohjelmistoprojektiryhmää.

Aiemmista työtehtävistä verkkosivukehittäjänä opinnäytetyöntekijällä oli kokemusta vakituisena työntekijänä vuoden 2016 keväästä aina vuoden 2019 syksyyn. Ensimmäinen harjoittelu verkkoviestinnän ammattiopintojen kautta työpaikalla alkoi jo vuoden 2014 syksyllä. Omaehtoisissa ja ammattikorkeakoulussa käydyissä opinnoissa opinnäytetyöntekijä on keskittynyt ohjelmistokehitykseen. Suuri osa ohjelmistoprojektissa käytetyistä kehitystavoista ja teknologioista olivat entuudestaan tekijälle tuttuja. Siten opinnäytetyöntekijällä on tätä työtä varten tarvittava osaaminen uusien työtehtävien suorittamiseksi.

1.1 Opinnäytetyön tarkoitus, tavoitteet ja rajaus

Opinnäytetyön tarkoitus on seurata päiväkirjanpitäjän kehittymistä käyttöliittymäkehittäjänä ja antaa alasta kiinnostuneille realistinen kuva alan vaatimuksista ja toimintatavoista. Ammatilliselle kehittymiselle asetetut ensisijaiset tavoitteet liittyvät käyttöliittymien kehittämiseen Reactilla ja TypeScriptillä sekä yksikkötestien kirjoittamiseen Jestillä ja Testing Librarylla. Toissijaiset tavoitteet liittyvät yleisissä ohjelmistokehityksen ja -projektin toimintatavoissa ja menetelmissä oppimiseen sekä ammattiroolissa kasvamiseen.

Ohjelmistoprojekti, johon päiväkirjaraportointi liittyy, kestää kuusi kuukautta. Seurantajakso rajataan kahdeksan raportointiviikon mittaiseksi. Raportointiviikot valitaan mahdollisimman kattavan kuvan saamiseksi päiväkirjanpitäjän kehittymisestä.

Opinnäytetyön tavoitteet ovat

1. Päiväkirjanpitäjän kehittyminen käyttöliittymäkehittäjänä.
    a. Kehittyminen käyttöliittymien ohjelmoimisessa Reactilla ja TypeScriptillä.
    b. Kehittyminen yksikkötestien kirjoittamisessa Jestillä ja Testing Libraryllä.
2. Päiväkirjanpitäjän kehittyminen ohjelmistokehittäjän ammattiroolissa.

a. Kehittyminen ammattilaisohjelmien ja -menetelmien, kuten versionhallinnan, ohjelmointiympäristön, ongelmanratkaisun ja ketterän kehityksen hallitsemisen saralla.

b. Oppiminen ja kehittyminen muissa ohjelmistoprojektissa tarvittavissa teknologioissa ja toimintatavoissa.

1.2 Keskeinen alan tietämys ja ammattikirjallisuus

Käytettyjen teknologioiden oma dokumentaatio on yleensä ensisijainen lähde, mistä kehitystyössä lähdetään teknologioiden kanssa liikkeelle (taulukko 1). Se on teknologian kehittäjien luoma manuaali, joka kertoo, kuinka teknologiaa on tarkoitus käyttää. (Grace 2018.) Tämän lisäksi on kuitenkin joukko ohjelmisto- ja verkkokehityksessä yleisesti käytettyjä resursseja, jotka eivät ole teknologiaspesifejä, kuten “https://stackoverflow.com” kysy/vastaa -palsta, “https://www.w3schools.com” opetusmateriaalit www-teknologioille ja “https://developer.mozilla.org” avoimen lähdekoodin dokumentaatiot. Googlettamisen merkitystä ei voi vähätellä ja se on myös olennainen osa ammattitaitoa (Hora 2021). Alan kirjojen ja tutkimuksen lisäksi myös verkkosivujen artikkelit ja blogijulkaisut näyttelevät suurta osaa sekä tässä opinnäytetyössä että alan keskeisessä tietämyksessä (Grace 2018).

Taulukko 1. Keskeisiä ohjelmistoprojektissa käytettäviä teknologioita ja niiden dokumentaatioiden kotisivut

Teknologia	**Dokumentaation kotisivu**
React	https://reactjs.org/docs/getting-started.html
TypeScript	https://www.typescriptlang.org/assets/typescript-handbook.pdf
Material-UI (v4)	https://v4.mui.com/
Jest	https://jestjs.io/docs/getting-started
Testing Library	https://testing-library.com/docs/react-testing-library/intro/
OpenAPI 3.0	https://swagger.io/docs/specification/about/

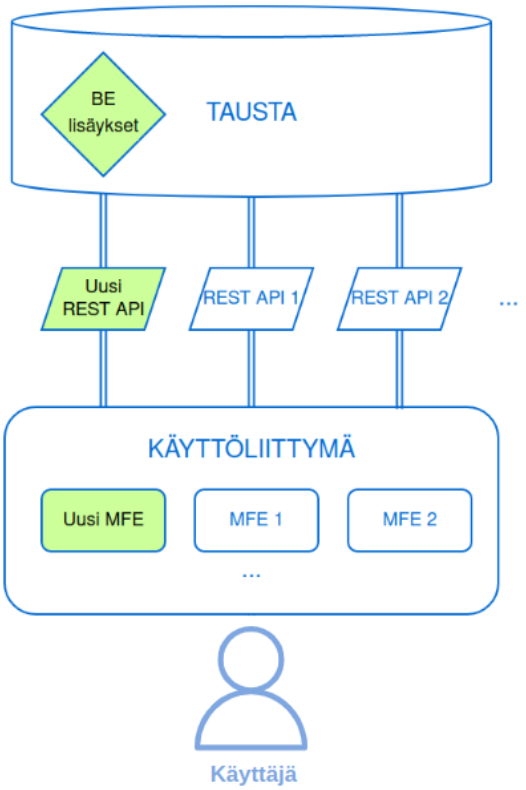
2 Lähtötilanteen kuvaus

Kuten johdannossa mainittiin, opinnäytetyön kirjoittaja palasi yli vuoden kestäneeltä opintovapaalta töihin tammikuun alussa 2022. Tällä välin hänen toimenkuvansa oli yhteisymmärryksessä työpaikan kanssa muutettu verkkosivukehittäjästä ohjelmistojen käyttöliittymäkehittäjäksi. Uusi projekti alkoi töihin paluuta seuraavalla viikolla. Töihin paluun viikolla tutuksi tuli uusi työympäristö ja -välineistö.

Työskentely tapahtui etätöinä kotitoimistolta, sillä vuoden 2022 tammikuussa elettiin edelleen koronarajoitusten ja poikkeusolojen aikaa. Työpaikan toimistolla oli menossa samaan aikaan myös remontti, joten myös koko muu projektiryhmä työskenteli etänä projektin alusta pitkälti sen loppuun asti. Työpaikka tarjosi töissä tarvittavat työvälineet: kannettavan tietokoneen, ulkoisen näytön, näppäimistön, hiiren, sekä muut oheislaitteet ja ohjelmistot.

2.1 Oman nykyisen työ analysointi

Työtehtävä uudessa työroolissa on käyttöliittymän toteuttaminen suunnitteilla olevaan ohjelmistoprojektiin. Suunniteltu ohjelmistoprojekti on osa suurempaa ohjelmistokokonaisuutta, yksi ohjelmiston moduuleista (kuva 1).



Kuva 1. Ohjelmiston arkkitehtuuri karkeasti hahmoteltuna ja uusi projekti vihreällä ilmaistuna

Ohjelmistoprojektin suunniteltu kesto oli kuusi kuukautta. Moduulin käyttöliittymä, kuten muutamat ohjelmistossa olevat sitä ennen, toteutetaan MFE:lla (Piispanen 2021, 5-6). Uutta moduulia varten määritellään myös uusi REST-rajapinta päätepisteineen, kuten kuvassa 1 sivulla 6 on esitetty, joka mahdollistaa käyttöliittymän ja taustan välisen tiedonvaihdon (Vuohijoki 2018, 4-5).

Työrooli

Projektin työskentelymenetelmä on mukautettu Scrum. Projektiryhmää johtaa tuoteomistaja ja projektiryhmän kehittäjät on jaettu sisältö-, design-, frontend- ja backend-tiimeihin, jotka työskentelevät saman tavoitteen eteen yhdessä Scrum-oppaan mukaisesti. Scrum-viitekehyksessä oleellista Scrum Masterin roolia projektissa ei kuitenkaan käytetä. (Schwaber & Sutherland 2020, 5-6.)

Opinnäytetyöntekijä kuuluu kolmehenkiseen FE-tiimiin, jota mentoroi muiden töidensä ohella lisäksi neljäs vanhempi käyttöliittymäkehittäjä. Uudessa työroolissa keskeiset osaamisen vaatimukset sekä kaventuivat että syventyivät huomattavasti verrattuna verkkosivukehittäjän työrooliin. Käyttöliittymäkehittäjän ja verkkosivukehittäjän työnkuvien välillä on paljon yhtäläisyyttä. Molemmissa työtehtävissä keskitytään selainpohjaisten ratkaisujen luomiseen, ja lokaali kehitysympäristö, versionhallinta, selaimen kehittäjätyökalut, sekä verkkosivujen rakenne ovat kaikki aiemmasta työnkuvasta tuttuja.

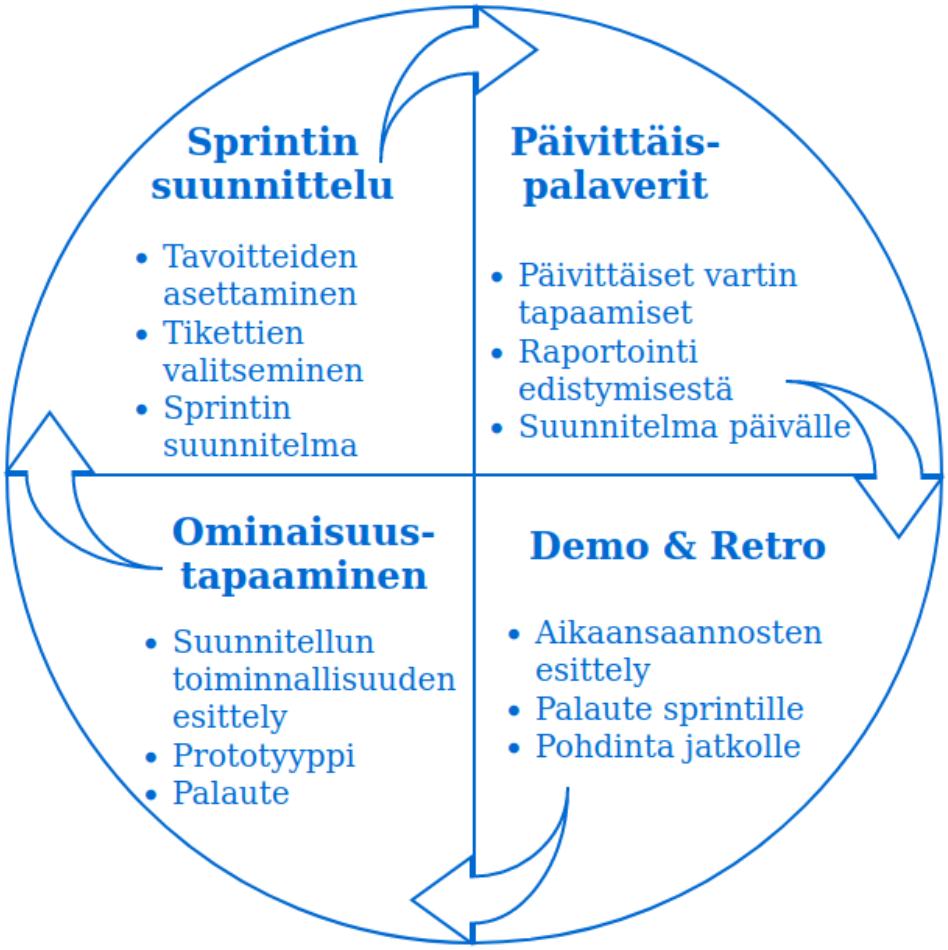
Verkkosivukehittäjän töissä keskityttiin sisällönhallintajärjestelmien ja niille suunniteltujen valmiiden komponenttien ja lisäosien käyttöön ja muotoiluun, sekä palvelinten ylläpitoon ja sivustojen tekniseen sisältöön. Käyttöliittymäkehittäjän töissä on paljon uusia työtehtäviä, kuten Reactilla kehittäminen, yksikkötestien kirjoittaminen ja REST-rajapintakutsujen ohjelmoiminen. Teknologioista TypeScript ja OpenAPI 3 (ent. Swagger) ovat täysin uusia tuttavuuksia tekijälle.

Työnkulku

Työskentely tapahtuu Scrum-oppaan mukaisesti sprinteissä. Ketterän kehityksen iteratiivinen työskentelymalli toistetaan sprintistä toiseen projektin alusta projektin päätökseen. Sprintit ovat kahden viikon mittaisia ja projektin soveltamassa mallissa ne sisältävät sprintin suunnittelun, päivittäispalaverit, sprintin demon ja retrospektiivin sekä ominaisuustapaamisen (kuva 2). (Schwaber & Sutherland 2020, 7-10.)

Sprintin suunnittelukokous edeltää sprintin käynnistymistä. Ennen suunnittelukokousta kehittäjät jakavat vaadittavan työn mitattaviksi inkrementeiksi tiketeille (Schwaber & Sutherland 2020, 12). Suunnittelukokouksessa tiketit siirretään projektin kehitysjonosta sprintin kehitysjonoon. Suunnittelua edeltää tarvittaessa ominaisuustapaaminen, jossa esitellään mahdolliset muutokset tai uudet

ominaisuudet prototyypin avulla. Sen jälkeen kehittäjät ottavat tikettejä työn alle sprintin kehitysjonosta ja raportoivat edistymisestään päivittäispalavereissa. Demossa sprintin aikaansaannokset esitellään ja retrospektiivissä käydään läpi, miten sprintti meni ja pohditaan tarvittavia jatkotoimenpiteitä.



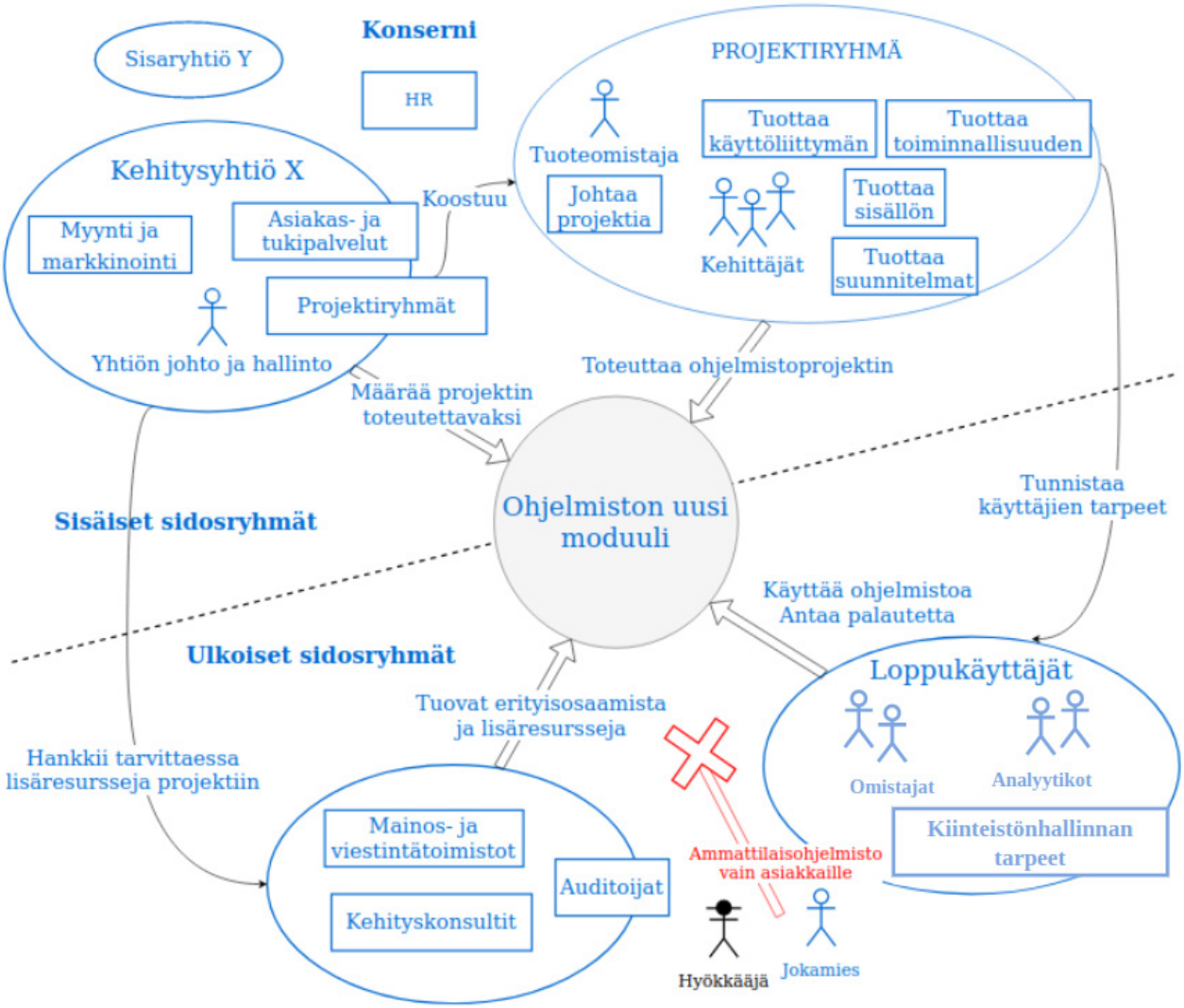
Kuva 2. Projektin mukautetun Scrum-mallin tapahtumat (mukaillen Morcov 2021, 78)

2.2 Sidosryhmien esittely

Ohjelmistoprojektilla on kaksi oleellista sidosryhmää: projektiryhmä mukaan lukien tuoteomistaja, ja asiakkaat eli loppukäyttäjät (kuva 3). Projektiryhmä toteuttaa ohjelmiston ja loppukäyttäjät käyttävät ohjelmistoa. (Savolainen 2011, 49-51.) Projektiryhmä esiteltiin jo edellisessä alakappaleessa työroolin yhteydessä ja siihen kuuluvat:

- tuoteomistaja (PO, Product Owner)
- kehittäjät
  - sisältötiimi
  - designtiimi
  - FE-tiimi
  - BE-tiimi

Yhteensä projektiryhmään kuuluvia henkilöitä on noin tusina. Määrä vaihtelee hieman projektin aikana. Sisältökonsultteja oli projektissa muutama projektin varrella. Käyttöliittymäkehittäjillä ei ollut vuorovaikutusta loppukäyttäjien kanssa.



Kuva 3. Projektin sidosryhmäkaavio

Ohjelmistoprojekti tai sen tarve ei synny kuitenkaan tyhjiössä, vaan projekti toteutetaan aina yhtiön tarpeiden mukaan. Projektiryhmän tehtävä on puolestaan tunnistaa käyttäjien lopulliset tarpeet (kuva 3). Loppukäyttäjiksi tunnistettuja ryhmiä ja heidän tarpeitaan olivat:

- Omistajat
- Projektien ohjaaminen kohti haluttuja tavoitteita simuloimalla lopputuloksia ennen varsinaista suunnittelua ja rakennustöitä
- Kiinteistöportfolioon liittyvien päätösten tekeminen simuloimalla kiinteistöjen elinkaaria
- Analyytikot
    - Arvon ja hiilijalanjäljen arviointi simuloimalla kiinteistöjen elinkaaria helposti arvioitavalla datalla

Kehitysyhtiö on osa suurempaa kiinteistö- ja rakennusalan konsernia. Konsernin hallinto hoitaa mm. palkat, lomat, sairaspoissaoloihin ja muuhun henkilöhallintoon liittyvät asiat. Yleensä markkinointi- ja viestintä, sekä asiakas- ja tukipalvelut kuuluvat myös osaltaan ohjelmistoprojektin sidosryhmiin, kuten tässäkin projektissa. Myynti ja markkinointi huolehtii, että loppukäyttäjät löytävät tuotteen. Asiakas- ja tukipalvelut huolehtivat, että loppukäyttäjät onnistuvat ohjelmiston käyttämisessä kaikilta osin.

2.3 Työpaikan vuorovaikutustilanteet

Koko projektiryhmä työskenteli lähes koko projektin etänä lukuun ottamatta virkistystoimintaa ja demopäiviä. Demopäivät ovat koko päivän kestäviä tapahtumia, joissa esitellään tulevia, meneillä olevia tai päättyviä projekteja, sekä muuta organisaation toimintaa ja tavoitteita. Etätöissä demopäivien yhteisöllinen luonne korostui. Ne jatkuivat virallisen osuuden jälkeen vapaamuotoisena seurusteluna myöhään iltaan, tosin virallista osuutta oli tosin mahdollista seurata myös etänä.

Vuorovaikutustilanteet tapahtuivat lähtökohtaisesti virtuaalisessa ympäristössä joko Atlassian-tuoteperheen tai Microsoftin ohjelmistojen välityksellä. Virallisemmat työasiat ja viestintä konsernin hallinnon kanssa tapahtui ensisijaisesti sähköpostilla. Vapaamuotoinen pikaviestintä tapahtui Teams-chateissa ja asteen virallisempi Teams-tiimeissä. Etätapaamiset olivat Teams-videopuheluita. Vaikutti, että niin kutsuttuja varjo-it ratkaisuja, eli it-ratkaisuja, joita IT-osasto ei ole hyväksynyt, ja jotka muodostavat ylimääräisen tietoturvariskin ei työasioissa käytetty (Myers, Starliper, Summers & Wood 2016, 1-4).

Scrum

Scrumin suunnitellut vuorovaikutustilanteet esiteltiin sivuilla 7 ja 8 kuvan 2 kera. Suunnitellut vuorovaikutustilanteet ovat:

- Ominaisuustapaaminen
- Sprintin suunnittelu
- Päivittäispalaverit
- Demo ja retrospektiivi

Tämä on työpaikan soveltama malli Scrum-oppaan Scrumin tapahtumista. (Schwaber & Sutherland 2020, 7-10). Näiden tapaamisten suunniteltu kesto on tunnin päivittäispalavereja lukuunottamatta, jotka ovat suunniteltu varttitunnin mittaisiksi. Tapaamiset ovat Teams-videopuheluita. Teams-videopuhelun ruudunjaon avulla tapaamisissa tukena käytetään prototyyppi-työkalu Axurea ominaisuustapaamisessa, Jira-boardia päivittäispalavereissa ja sprintin suunnittelussa, kehitysympäristöjä demossa ja Confluence-projektidokumentaatiota retrospektiivissä.

PR:t

Yksi tärkeä vuorovaikutustilanne on PR:t ja niiden kommentointi. PR:n laatiminen on kommunikaatiotilanne, lähtien aina PR:ssä olevasta koodista ja PR:n sisältämistä versionhallinnan viesteistä. Lisäksi PR:n tekijän laatima PR:n kuvaus on tärkeä osa PR-kommunikaatiota. (Atlassian 2022a). PR:iin pyritään lisäämään kuvakaappauksia ja gif-animaatioita ohjelmoidun toiminnallisuuden havainnollistamiseksi käyttöliittymässä. PR:ssä kommunikoinnista on kirjoitettu paljon artikkeleita ja blogijulkaisuja. Puhutaan PR-etiketistä (Pountney 2021). Koska kyse on kirjoitetusta tekstistä ja ohjelmistokoodista, oleellista on pyrkiä kommunikaatiossa täsmällisyyteen, positiivisuuteen ja vuoropuhelun mahdollistamiseen (McMinn 2015).

3 Opinnäytetyön tietoperusta, menetelmä ja lähteet

Opinnäytetyö on kehittämispäiväkirjamuotoinen laadullinen tutkimus ja sen menetelmä perustuu autoetnografisen tutkimuksen käytänteisiin (Bister 2019, 59). Kuten johdanto-osuudessa mainittua, opinnäytetyöntekijä pitää seurantajakson eli kahdeksan seurantaviikon ajan päiväkirjaa töistään. Jokaiselle työviikolle asetetaan tavoite ja päiväkirjanpitäjä raportoi työviikon tapahtumat päivittäin. Raportointiviikon lopuksi seuraa viikkoanalyysi, jossa pohditaan työviikon tapahtumia ja asetetun tavoitteen toteutumista. Viikkoanalyyseissa myös syvennytään viikon tavoitteiseen ja kartutetaan niihin liittyvää osaamista. Seurantajakso valitaan niin, että tekijän taitojen kehittymisestä saadaan mahdollisimman kattava kuva.

Opinnäytetyön rakenne ammentaa Räisäsen & Virkkalan (2020) sekä Bisterin (2019, 59-60) kuvaamista päiväkirjamallisen opinnäytetyön rakenteista. Opinnäytetyö alkaa käsitetaulukolla ja johdanto-osuudella. Seuraavaksi kuvataan lähtötilanne ja opinnäytetyön tieteellinen viitekehys. Viimeisenä seuraa pohdinta. Pohdintaosuudessa tarkastellaan opinnäytetyön tavoitteiden toteutumista.

Pohjan opinnäytetyössä käytetyille lähteille muodostaa kappaleessa 1.2 “Alan tietämys ja kirjallisuus” esiteltyjen teknologioiden dokumentaatiot ja mainitut yleiset kehitystyössä käytetyt resurssit. Lähdeluettelo kasvaa kuitenkin näitä dokumentaatioita huomattavasti laajemmaksi. Koska opinnäytetyöhön liittyvässä ohjelmistoprojektissa käytetty työskentelymenetelmä on Scrum, on Scrum-opas myös oleellinen osa opinnäytetyön viitekehystä. Menetelmien teoriapohjaa laajentaa PR:ien tekemiseen liittyvät kirjoitukset, versionhallintaohjelmisto Git:n dokumentaatio, ohjelmarajapintojen arkkitehtuurityyli REST:iin ja käyttöliittymäarkkitehtuurityyli MFE:hen liittyvät lähteet.

Opinnäytetyöt, Piispanen (2021) ja Suvisuo (2020), ovat toimineet esimerkkeinä tässä opinnäytetyössä käyttöliittymäkehityksen osalta. Oppimisalusta Udemy Businekseen ja siellä suoritettujen yksikkötestaus- ja OpenAPI -kurssien merkitystä tekijän ammatilliselle kehittymiselle ei voi vähätellä. Helsingin yliopiston avoin verkkokurssi “Full stack open 2022” materiaaleineen on myös toiminut merkittävänä apuna opinnäytetyön rakennetta ja aiheita pohtiessa. Päiväkirjamuotoisen opinnäytetyön kirjoittaminen ohjelmistoprojektiin osallistumisen ohessa on hyvin reaktiivista kaikesta suunnittelusta huolimatta. Opinnäytetyöntekijä on osa omaa opinnäytetyötään suurempaa projektia, ja opinnäytetyön lopullinen tietoperusta, käytetyt menetelmät ja lähdeluettelo kuvastavat tätä.

4 Seurantajakson raportointi viikkoanalyyseineen

Ensimmäistä raportointiviikkoa edeltävällä töihin paluun viikolla päiväkirjanpitäjä sai työvälineet ja ohjeet, kuinka ohjelmistoprojektin työympäristö asennetaan ja otetaan käyttöön. Samalla saatiin myös dokumentaatio käyttöliittymäkehityksessä käytetyistä teknologioista ja ”Hello world” -harjoitus vanhemmalta kehittäjältä projektiteknologioihin tutustumiseksi. Monet projektiteknologiat olivat raportoijalle entuudestaan joko täysin tai verrattain vieraita.

Seurantajakson raportointi alkaa töihin paluuta seuraavalla viikolla, jolloin ohjelmistoprojekti käynnistetään. Ensimmäiset viisi raportointiviikkoa seuraavat toisiaan, sillä projektin alkupuolella riittää eniten raportoitavaa. Viimeiset kolme viikkoa, jotka sijoittuvat ohjelmistoprojektin loppupuolelle, kuvaavat raportoijan kehittymistä käyttöliittymäkehittäjänä. Lisäksi toiseksi viimeisellä seurantaviikolla kurotetaan käyttöliittymäkehityksestä kohti backend-kehittäjien työtehtäviä REST-rajapinnan ohjelmoinnin merkeissä. Viimeisten kolmen seurantaviikon välissä on ajallisia raportointitaukoja, jotka käydään läpi tarkemmin raportissa.

Jokaiselle seurantaviikolle nimetään teema, johon seurantaviikolla keskitytään raportoinnissa ja erityisesti viikkoanalyyseissa. Seurantaviikkojen teemat valitaan kulloisenkin raportointiviikon ja sitä edeltävien seurantaviikkojen tarpeiden perusteella. Päiväraportointeja edeltää seurantaviikon alustus, jossa käydään läpi teema ja sen tarve. Listaus seurantaviikkojen teemoista:

1. Projektin käynnistäminen ja työskentelymenetelmät
2. React-komponentit
3. TypeScript
4. Yksikkötestaus
5. Legacy-ratkaisut ja teknologioiden kehityshistoria
6. REST-rajapinnan käyttö
7. REST-rajapinnan muokkaaminen
8. Komponenttikokonaisuus ja CRUD-operaatiot

4.1 Seurantaviikko 1: Projektin käynnistäminen ja työskentelymenetelmät

Ensimmäinen seurantaviikko on töihin paluuta seuraava viikko, jolloin raportointijakson aikana seurattava ohjelmistoprojekti alkaa. Viikon ohjelmaan kuuluu paljon projektin hallinnallisia ja kehittäjien työskentelyyn olennaisesti vaikuttavia asioita. Viikon tiedetään olevan poikkeuksellisen sisältörikas ja täynnä kokouksia jo etukäteen. Tämä on päiväkirjanpitäjän ensimmäinen ohjelmistoprojekti töissä. Seurantaviikon tavoitteena on saavuttaa kokonaiskuva käynnistyvästä uudesta projektista, sen työskentelytavoista ja tavoitteista.

Ma 10.1.2022, Kickoff

Maanantai alkaa projektin ensimmäisellä kickoff-kokouksella. Tämä päivä ja koko viikko on omistettu projektien ja laajemmin työvuoden käynnistämiselle. Minua oli informoitu jo edeltävällä viikolla, että viikko tulisi olemaan täysin poikkeuksellinen kokouksien määrässä, kestossa ja laadussa. Koska oli edelleen pandemia, poikkeusajat ja etätyökäytännöt voimassa, kokoukset pidettiin etäpalavereina Microsoft Teams-sovelluksessa.

Ensimmäisessä puoleenpäivään kestävässä etäpalaverissa esiteltiin tuoteomistajan johdolla projektin työryhmä toisilleen ja käytiin läpi mitä projektin kickoffilla ylipäänsä tavoiteltiin, sekä tulevan viikon ohjelma ja yleiskuva projektin päämääristä. Alustana esityksissä käytettiin Miroa (Miro, 2022). Tulisimme rakentamaan uuden moduulin ohjelmistokokonaisuuteen. Uuden moduulin tarkoitus on mallintaa kiinteistöjen korjaussuunnitelmia. Projektiryhmä jakautuu tuoteomistajan lisäksi sisältö-, design-, backend- ja frontend-tiimeihin. Olen osa frontend-tiimiä.

Projektikokouksen jälkeen käytiin läpi vielä erillisessä design-palaverissa projektin prototyyppi yksityiskohtaisesti frontend- ja design-tiimien kesken. Samassa palaverissa pohdittiin projektin käyttöliittymäkehityksessä tarvittavia teknologioita. Vaikka projektin kehitystyö potkaistiin käyntiin vasta tänään, oli selvää, että sitä edelsi paljon suunnittelutyötä.

Ti 11.1.2022, Kickoff

Tiistai alkoi yhteisellä vuoden kickoff-etäpalaverilla, jossa esiteltiin alkavat projektit ja samalla esiteltiin organisaation nykytilaa ja tavoitteita käynnistyvälle vuodelle. Puolenpäivän jälkeen projektin kickoff jatkui seuraavalla kokouksella, jonka tavoitteena oli tunnistaa moduulin ylätason toiminnallisuudet ja luoda niihin liittyviä käyttäjätarinoita. Suunnittelu tapahtui jälleen Miro-boardilla, josta tulokset dokumentoitiin jälleen Conflunceen. Ylätason ominaisuuksista luotiin epicejä ja käyttäjätarinoista epiceihin liittyviä tikettejä Jira-boardiin.

Tänään kokousten jälkeen luotiin myös uusia viestikanavia projektiviestinnän tueksi Teamsiin koko projektikehitysryhmälle ja jokaiselle tiimille. Myöhään iltapäivään kestävien kokousten jälkeen jatkan tutustumista projektiteknologioihin lokaalissa kehitysympäristössäni vanhemman kehittäjän toiveesta. Opettelen käyttämään TypeScriptiä entuudestaan tutun Reactin kanssa. Luon oman React “hello world” -komponentin TS-tyypityksiä ja tarkistusohjelman (linter) sääntöjä noudattaen jo päättyneen projektin etähaaraan.

Ke 12.1.2022, Sprint 1

Keskiviikko jatkaa suoraan eilen kesken jääneistä epiceistä ja se jatkuu FE:n yhteisellä tiketöintisessiolla. Luomme tiketöitävissä olevat ominaisuudet kehitysjonoon odottamaan. Näistä valitsemme sprintin suunnittelussa sprintin kehitysjonoon seuraavan sprintin tehtävät. Katselmoimme lopuksi sprintin kehitysjonon tuoteomistajan johdolla yhteisesti koko projektiryhmän kanssa. Taustakehittäjillä on paljon kysymyksiä liittyen erääseen suunniteltuun toiminnallisuuteen, mutta se ei vaikuta suoranaisesti työskentelyyn vielä. Lisäksi tiedostetaan, että taustakehittäjät tulisivat työskentelemään edellisissä projekteissa vielä uuden projektin alkaessa.

To 13.1.2022, Sprint 1

Torstai alkaa vanhemman kehittäjän johdolla projektin käyttöliittymäkehittäjien yhteisellä MFE:n pystytyksellä. Prosessi on jo tuttu, sillä se on sama kuin “hello world” -harjoituskomponenttia luodessa. Vanhempi kehittäjä on luonut projektin MFE:lle oman repositoryn, jonka muut kehittäjät kloonaavat ja aloittavat kehittämisen omissa lokaaleissa kehitysympäristöissään. Valitsin yhteisymmärryksessä muiden kehittäjien kanssa ensimmäisen tikettini tänään ja aloin kehittämään moduullemme kehikkoa, johon muut moduulin komponentin sijoitettaisiin. Projektinjohto saa tänään Jira-tehtäväpöydän automatisoitua projektiin yhteensopivaksi.

Pe 14.1.2022, Sprint 1

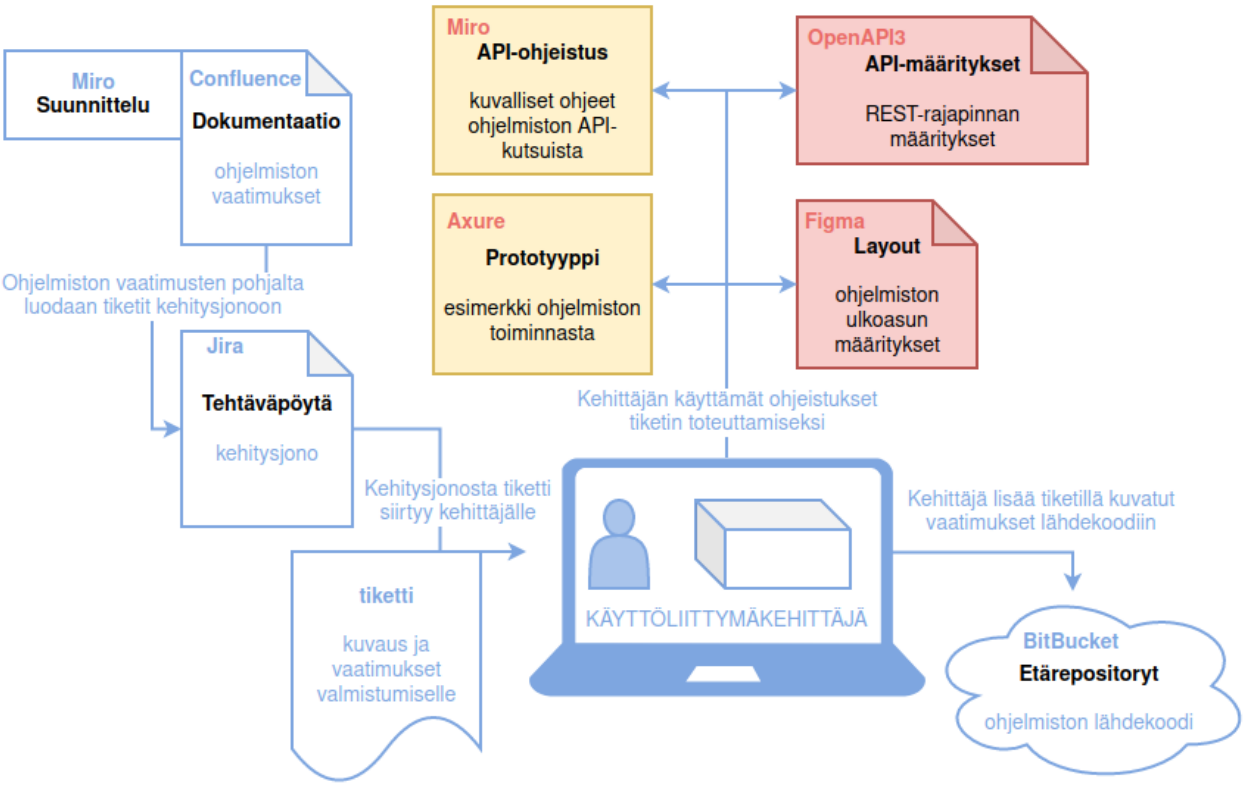
Perjantaina jatkan ensimmäistä aloitettua tikettiäni, ensimmäistä komponenttiani projektiin. Teen työpäivän lopulla ensimmäisen PR:ni MFE:n repositoryyn. PR:ssäni luodaan moduulin käyttöliittymän kehikko ja placeholder-komponentit, jotka myöhemmin korvataan oikeilla toiminnallisilla komponenteilla.

Viikkoanalyysi 1

Ensimmäisellä seurantaviikolla tapahtui poikkeuksellisen paljon. Ensinnäkin projekti alkoi ja projektin johtaja eli tuoteomistaja pyrki varmistamaan, että jokaisella projektin kehittäjällä oli selkeä kuva projektin tavoitteista ja prosesseista, kuinka tavoitteisiin pyrittiin. Projektin kuuden kuukauden aikataulu oli jo etukäteen tiedossa, mutta projektin toiminnallisuudet aikataulutettiin ja priorisoitiin. Kehitysprojektia edeltävässä suunnittelutyössä mukana olleet avasivat dialogin ohjelmiston suunnitellusta toiminnallisuudesta varmistuakseen sen toteutettavuudesta. Keskustelun päätteeksi varmistettiin, että kaikilla kehittäjillä oli käsitys ohjelmiston halutusta toiminnallisuudesta.

PowerPoint-esityksien sijaan suunnittelutyössä ja projektin videopalaverien esityksissä käytettiin Miroa. Miro-taulussa hyvä puoli on se, että kaikki voivat työskennellä samassa näkymässä yhtäaikaisesti. Lisäksi PowerPointin esitysefekteihin ja lisätehosteisiin ei tarvitse kiinnittää huomiota, kun suunnitelma on kehitysvaiheessa.

Käyttöliittymäkehityksen aloittamiseksi kaikki vaikutti valmiiksi mietityltä (kuva 4). Uusia aiemmin kokeilemattomia teknologioita ei ollut tiedossa. Suunnitellut kehitystavat vaikuttivat moderneilta ja projektiin soveltuvilta. BE-tiimillä oli omia epäilyksiään projektin alussa projektin toiminnallisuusvaatimuksista, mutta ne eivät heti vaikuttaneet heti käyttöliittymäkehityksen suunnitelmiin.

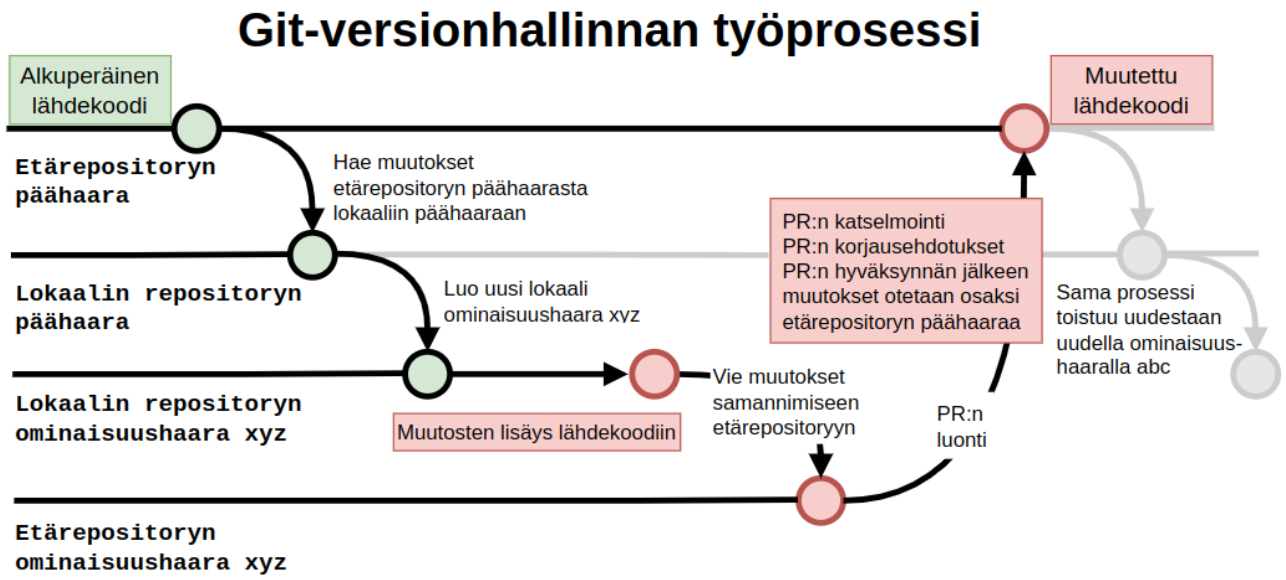


Kuva 4. Projektin käyttöliittymäkehityksen työprosessi

Uusia ominaisuuksia kehittäessä käyttöliittymäkehittäjät luovat ensin uuden lokaalin kehityshaaran, jonka jälkeen he puskevat sen samannimiseen etäkehityshaaraan. Sitten muut kehittäjät vertaavat alkuperäistä etäkehityshaaraa muutosehdotuksia sisältävään etäkehityshaaraan, ja mikäli he päätyvät hyväksymään sen, alkuperäinen etäkehityshaara ja muutosehdotuksia sisältävä etäkehityshaara yhdistetään (muutosehdotukset sisällytetään alkuperäiseen etäkehityshaaraan). Yleensä

etäkehityshaaran nimi on development. Yleensä development-haaran lisäksi repository sisältää lisäksi main-haaran (master-haara vanhentunut vastine) ja deployment-haaran. Tämä riippuu kuitenkin kehitystavoista ja -tarpeista.

Sen jälkeen kehittäjät muokkaavat ensin omaa lokaalia repositoryaan uusia ominaisuuksia kehittäessään ja avaavat PR:eja etärepositoryyn. Omaa lokaalia repositoryaan muokatessaan kehittäjät luovat ensin uuden lokaalin kehityshaaran, johon muutokset tehdään ja joka viedään samannimiseen etähaaraan. Tästä etähaarasta tehdään lopulta PR etärepositoryn pääkehityshaaraan, ja tämän PR:n muutokset lopulta muut kehittäjät tarkastavat (kuva 5).



Kuva 5. Git-versionhallinnan yleinen työnkulku projektissa

Tämä prosessi toistuu aina uudestaan etäpäähaaraan tulleiden muutosten jälkeen: muutokset haetaan ensin lokaaliin päähaaraan, sitten luodaan uusi lokaali ominaisuushaara ja tehdään lisää muutoksia. Sama prosessi Git-komennoilla aina PR:n luomiseen asti näyttäisi tältä, komentojen selitykset suluissa:

> git status (Varmistetaan, että ollaan oikeassa lokaalissa haarassa ja eikä esteitä jatkokehitykselle ole.)
> git pull (Haetaan muutokset etärepositorysta.)
> git checkout -b xyz (Luodaan uusi lokaali kehityshaara nimeltä xyz.)
> git add . (Lisätään kaikki kehityshaaraan tehdyt muutokset odottavaan tilaan.)
> git commit -m “new feature” (Vahvistetaan odottavat muutokset ja liitetään vahvistusviesti “new feature” selitteeksi muutoksista.)
> git push -u origin HEAD (Luodaan samanniminen etäkehityshaara xyz ja viedään muutokset

sinne.)
git checkout main (Palataan takaisin lokaaliin päähaaraan nimeltä “main”. Seuraava ominaisuus kehitettäisiin jälleen aloittamalla komentosarja alusta ja nimeämällä uusi kehityshaara, esim. abc.) (Atlassian 2022b; Chacon & Straub 2014, luku 5.2.)

Tämän jälkeen PR luodaan GUI:lla (graafisella käyttöliittymällä) palvelussa, jossa etärepositoryt sijaitsevat. PR:iin lisätään lyhyt kuvaus mahdollisesti video- tai kuvakaappauksen kera. Niihin kohtiin PR:ssä kommentoidaan, mihin muiden kehittäjien halutaan erityisesti kiinnittävän huomiota. (Atlassian 2022a). PR olisi todennäköisesti mahdollista luoda myös terminaalissa Git-komennoilla (Chacon & Straub 2014, luku 5.2). Koko Git-prosessin hallintaan on olemassa graafisia työkaluja, kuten Atlassianin SourceTree. Bitbucket, jossa repositorymme sijaitsevat, on myös Atlassianin työkalu.

Prosessi oli entuudestaan tuttu lähinnä töistä, sillä kouluprojekteissa mentiin yleensä huomattavasti pienemmällä määrällä Git-komentoja. Opinnoissa oli yleistä, että sekä omissa että ryhmäprojekteissa pusketaan muutokset suoraan päähaaraan, eikä PR:iä yleensä tehty tai ainakaan katselmoitu. Muutosten puskemista suoraan etäpäähaaraan pitäisi välttää, koska se lisää muutokset suoraan etäpäähaaran lähdekoodiin ilman katselmointia ja muiden kehittäjien hyväksyntää.

4.2 Seurantaviikko 2: React-komponentit

Seurantaviikko kaksi jatkaa raportointia suoraan ensimmäisestä seurantaviikosta. Ensimmäinen sprintti, jolle kaksi ensimmäistä seurantaviikkoa osittain sijoittuvat, kesti poikkeuksellisesti vain viikon. Tällä seurantaviikolla keskiöön nousee varsinainen kehitystyö, React-komponenttien kirjoittaminen. Edellisviikon päätteeksi ennätin jo luomaan ensimmäisen PR:ni. Tällä viikolla analysoidaan React-komponenttien kirjoittamisessa tarvittavia työkaluja.

Ma 17.1.2022, Sprint 1

Tämän päivän tavoite oli jatkaa hyvin alkanutta kehitystyötä ja uusien komponenttien kirjoittamista. Sain ensimmäisen PR:ni aikaiseksi edellisviikon perjantaina, joten valitsin Jiran kehitysjonosta seuraavan tiketin ja aloitin siinä määritellyn komponentin kehittämisen. Aloitin tänään myös testien kirjoittamisen. Ensimmäinen PR:ni ei sisältänyt testejä. Ymmärsin, että testeissä tulisi olemaan haastetta, sillä minulla oli Reactin yksikkötesteistä kokemusta vain muutaman testin verran entuudestaan ammattikorkeakoulun kursseilta. Seuraava komponentti oli myös erittäin yksinkertainen, mutta sisälsi jo testattavaa toiminnallisuutta. Tänään törmäsin myös sisäiseen listakomponenttiimme.

Ti 18.1.2022, Sprint 1

Tänään muut FE-tiimin jäsenet olivat poissa, toinen sairaana ja toinen lomalla, joten oli vastuullani esitellä projektin ensimmäisessä demossa FE-tiimin aikaansaannokset. Vaikka ensimmäinen PR:ni on avattu, ei sitä ole vielä tarkastettu muiden kehittäjien toimesta, mutta esittelen sen siitä huolimatta demossa. Retrossa pohditaan, kuinka ensimmäinen sprintti meni: mitkä asiat menivät hyvin, missä on parannettavaa ja onko jotain, jota uupuu suunnitelmista kokonaan.

Ke 19.1.2022, Sprint 2

Keskiviikon olisi pitänyt alkaa ominaisuustapaamisella, mutta se oli peruttu, sillä prototyyppiin tai designiin ei ollut tulossa uusia päivityksiä. Myöskään päivittäispalaveria ei ollut, koska tänään on sprintin suunnittelu. Kysyn, onko meillä FE-kehittäjillä keskinäisiä tapaamisia yli projektirajojen. Niitä on kuulemma ollut, mutta ei tällä hetkellä aktiivisina. Yhteiset vapaaehtoiset kahvitauot päätetään aloittaa jälleen perjantaisin FE-kehittäjien kesken.

Tänään jatkoin myös tutustumista omaan sisäiseen listakomponenttiimme, joka oli niin suuri komponenttikokoelma, että sitä varten oli oma repository ja opetusvideo. Lisäksi ohjelmiston muut riippuvuudet tuottavat päänvaivaa, ja opin muutamia tapoja riippuvuuksien päivittämiseen.

To 20.1.2022, Sprint 2

Jouduin eilen poistamaan toimimattoman testin PR:stä ja hakemaan sille hyväksynnän muilta kehittäjiltä, joka aiheutti laajaa kehittäjien välistä keskustelua testausfilosofiasta ja -vaatimuksista. En kykene ratkomaan ulkopuolisten komponenttien testausongelmia yksin, joten olen tyytyväinen, että saan luvan jatkaa ilman testien kirjoittamista komponentille. Tiedän, että TDD-filosofian mukaisesti en tällä hetkellä etene, mikä oli vielä ennen projektin alkua aikeeni. Koen kuitenkin pelkkien komponenttien kirjoittamisen tarpeeksi haastavaksi, joten en ole asiasta harmissani.

Olen saanut Udemy Business -tunnukset ja tarkoituksena on aloittaa testauskurssi, koska tuntuu että myös muut kehittäjät yli projektirajojen ovat toisinaan ongelmissa testien kirjoittamisen kanssa. Komponenttien kirjoittaminen on kiinnostavaa, mutta yksikkötestaaminen on hiukan haasteellista. Testien kirjoittaminen ei ole myöskään samalla tavalla palkitsevaa kuin konkreettisten kaikille näkyvien komponenttien koodaaminen. Pohdin yksikkötestaamiseen erikoistumista.

Käytän muuten päivän uuden komponentin kirjoittamisen parissa. Ensimmäinen PR:ni on viimein hyväksytty muiden kehittäjien toimesta ja liitetty osaksi projektia. Saan tänään myös uuden PR:n valmiiksi, nyt kun komponentilta sai poistaa testit. Aloitan myös seuraavan komponentin kirjoittamisen.

Pe 21.1.2022, Sprint 2

Perjantaina jatkan uuden komponentin kirjoittamista. Tämä komponentti osoittautuu paljon haasteellisemmaksi kuin kaksi edellistä yksinkertaista komponenttia. Lisäksi syitä tai tekosyitä testien kirjoittamatta jättämiseen ei ole. Tänään on myös ensimmäinen FE-kehittäjien yhteinen kahvitauko – juttelemme niitä näitä lähinnä työ- ja tutustumisasioissa, ja sovimme seuraavalle kerralle yhteisen pelisession.

Opinnäytetyöhön, Suvisuo (2020), tutustun työaikana perjantain kunniaksi. Kuulen myös keskustelua käyttämästämme MUI-versiosta ja mahdollisuudesta, että siirrymme nykyisessä projektissa uusimpaan MUI-versioon.

Viikkoanalyysi 2: React-komponentit

React-komponenttien kirjoittamista projektissa määrittelivät useat vaatimukset. Ensinnäkin kirjoitimme funktionaalisia komponentteja erotuksena luokkakomponentteihin (React 2022a). Funktionaaliset komponentit mahdollistivat vuonna 2018 Reactin versiossa 16.8 esitellyn React Hookien käyttämisen. Ennen React Hookeja komponentin tilan muuttaminen oli mahdollista vain luokkakomponenteissa. (React 2022b). Luokkakomponentit tulivat puolestaan mahdollisiksi Reactissa JavaScriptin ECMAScript 2015 version eli ES6 myötä. (React 2022a).

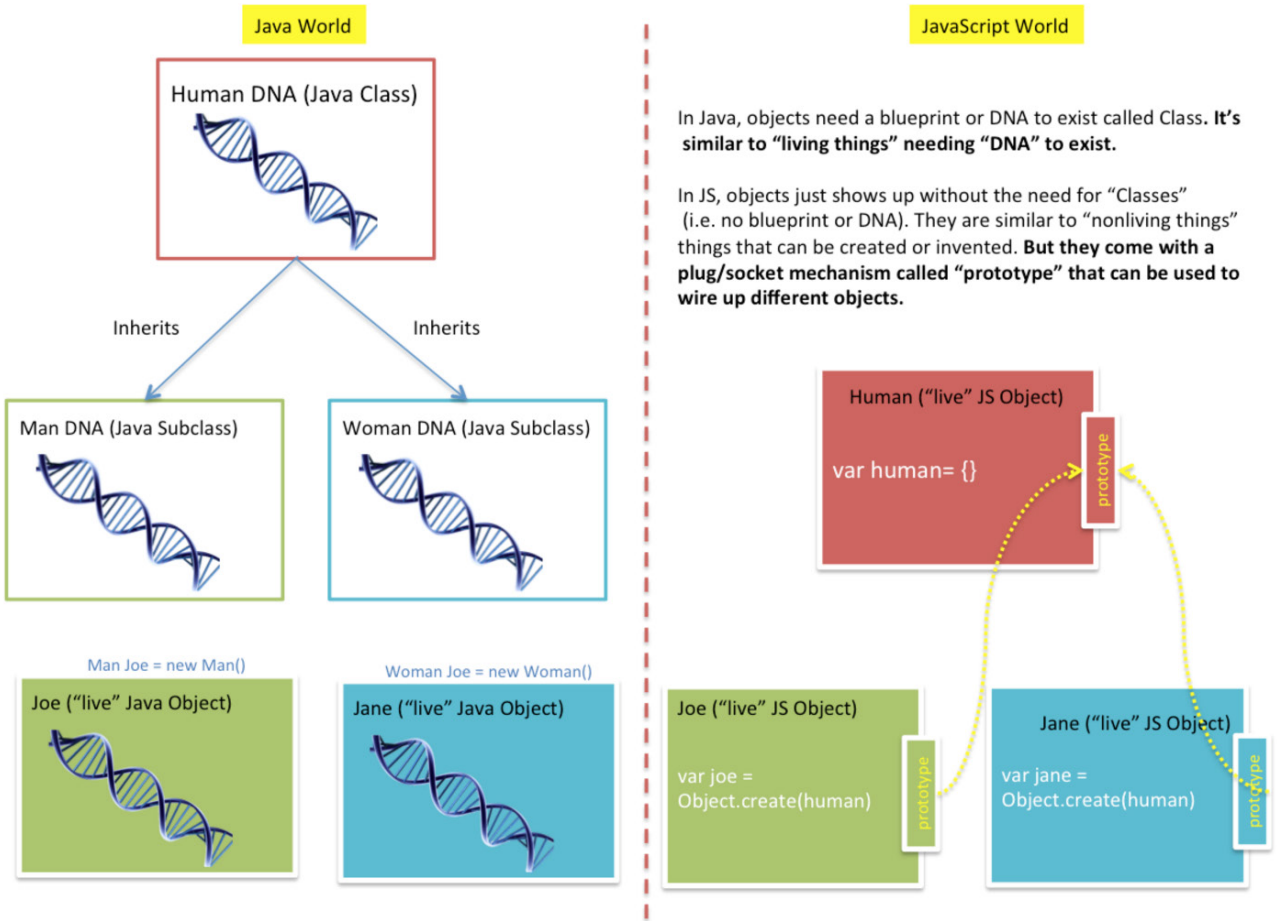
Luokkia käyttävä olio-ohjelmointi oli opintojeni pohjalta paljon funktionaalista ohjelmointia tutumpaa ennen projektia (kuva 6). Etsin Udemy Busineksesta kursseja Reactin kirjoittamiseen TypeScriptillä, ja niissäkin opetettiin käyttämään luokallisia komponentteja. Reactin oma virallinen dokumentaatiokin käyttää esimerkeissään jatkuvasti luokkakomponentteja. JavaScript ei ole kuitenkaan luokallinen kieli, toisin kuin esimerkiksi Java (MDN 2022a). Funktionaalinen ohjelmointi ja React Hookit yksinkertaistavat, selkeyttävät ja typistävät tarvittavaa koodia huomattavasti (Nagey 2019).

React Hookien käyttäminen oli kuitenkin tuttua jossain määrin ja olin erittäin tyytyväinen, että uudessa projektissa oli omaksuttu Reactin uudet toiminnallisuudet (React 2022c). Verkkosivukehittäjälle React-syntaksi oli varsin helppolukuista ja muistutti verkkosivujen HTML-rakennetta (Suvisuo 2020, 8).

Komponentin tilan viemiseksi komponentista toiselle ohjelmistossa käytimme Reactin sisäänrakennettua Context API -ratkaisua. Toinen yleinen ratkaisu Reactissa on käyttää Redux-kirjastoa tilojen välittämiseen komponentilta toiselle. Jos tarkoitus on pelkästään välittää tiloja komponentilta toiselle, ei Redux todennäköisesti ole kuitenkaan oikea ratkaisu ongelmaan (Erikson 2018). Tiesin

molemmista ratkaisuista nimellisesti. Olin helpottunut, että tässä projektissa käytimme kuitenkin Reduxiin verrattuna yksinkertaisemmalta vaikuttavaa Context API:a (Bose, 2021).

Ero aiempaan työhön verkkokehittäjänä, jossa kaikki erilaiset selaintyypit ja näyttökoot oli otettava huomioon, oli huomattava. Ammattilaisohjelmistoa optimoidaan ainoastaan yhdelle selaimelle ja näyttökoolle. Aiemmin iso osa työajasta meni sen varmistamiseen, että verkkosivujen ratkaisut toimivat myös esimerkiksi mobiililaitteilla ja nykyisin jo elinkaarensa päähän tulleella Internet Explorer -selaimella. Projektin ohjelmistolle oli asetettu kaksi vaatimusta toimintaympäristöstä: toiminta Chrome-selaimen viimeisellä versiolla täysteräväpiirron (1920x1080) selainkoossa.



Kuva 6. Objects in Java World VS JavaScript World (DV 2016)

Komponentit

Monet React-komponentit ovat monipuolisia listoja, joiden pitää taipua mitä erilaisimpiin toiminnallisuuksiin. Toiminnallisuusvaatimukset määritellään vasta projekteja suunnitellessa ja tästä syystä oma listakomponentti on välttämättömyys. Kehitysfilosofiassa oma komponentti päihittää ulkopuolisen riippuvuuden, koska oma komponentti mahdollistaa komponentin muokkaamisen kaikkien tarpeiden mukaiseksi. Ulkopuolisen komponentin etuna olisi puolestaan kehitysnopeus eli valmiin komponentin käyttö, ja haittapuolena puolestaan riippuvuus komponenttiin määritellyistä toiminnallisuuksista.

Ulkopuolisista komponenttikirjastoista käytimme Material-UI:ta, jonka listakomponentti on hyvin yksinkertainen. Ammattikorkeakoulun React-kurssilla olin oppinut käyttämään AG Grid listakomponenttia. AG Gridin listakomponentti on puolestaan hyvin monipuolinen, mutta silti sekään ei todennäköisesti sisällä useaa listakomponenteiltamme vaadittua toiminnallisuutta – ja erityisesti niitä potentiaalisia tulevaisuudessa vaadittavia toiminnallisuuksia.

Joskus projektin ulkopuolelta tuodun komponentin, ulkopuolisen tai oman, tyylejä joutuu muokkaamaan projektin vaatimalle tasolle. Jos komponentti tulee omasta apukirjastosta, voi komponentille lisätä ominaisuuden apukirjastossa, jonka avulla halutut tyylit voi ottaa käyttöön (Piispanen, 9-11). Toisinaan kuitenkin muutokset voivat olla lukuisia, pieniä ja vaikuttavat projektispesifeiltä. Voi myös vaikuttaa helpommalta ylikirjoittaa tyylimääritykset vasta projektissa. Jos komponentti tulee ulkopuolisesta kirjastosta ja se on yleisesti käytetty, tehdään siitä oma versio komponenttikirjastoomme yhtiön graafisilla ohjeistuksilla.

Mahdollisuuksia on monia ja tulisi aina miettiä tilanteen yleispätevyyttä: onko kyseessä vain projektissa käytetty muunnos – tyylit tehdään projektiin, vai onko kyseessä yleinen tyyli – tyylit on tehtävä omiin sisäisiin kirjastoihimme uusina ominaisuuksina. Aina näin ei luonnollisestikaan menetellä. Tämä on ohjelmistokehityksessä yleinen helmasynti, että ratkaisu samaan ongelmaan luodaan useita kertoja eri paikassa, kun yksi monistettavissa oleva ratkaisu riittäisi DRY-periaatteen eli “Don’t repeat yourself” mukaisesti (Hunt & Thomas 2019, luku 2).

4.3 Seurantaviikko 3: TypeScript

Ma 24.1.2022, Sprint 2

Olen sairaana! Tutuksi tulevat sairaspoissaolokäytännöt, kuten merkintä työajanseurantaan, hallinnolle ja tiimille. Autoin siitä huolimatta kanssakehittäjää eri projektitiimistä Ubuntun Nvidia-ajurei-

den kanssa, sillä ajaudun keskustelemaan sairaspäivänä työviestimissä. Olen sunnuntaina ratkaissut omat ongelmani ajureiden kanssa ja yritän auttaa kanssakehittäjää hänen ongelmiensa kanssa.

Ti 25.1.2022, Sprint 2

Tiistaina FE-tiimimme on jälleen täysilukuinen, sillä olemme kukin vuorollamme olleet joko matkoilla tai sairaana. Viime viikon lopulla aloitin jälleen uuden komponentin kirjoittamisen. Tämä kolmas komponentti osoittautui huomattavasti monimutkaisemmaksi kuin kaksi hyvin yksinkertaista aiempaa komponenttia. Tässä komponentissa piti käyttää jo aiemmin mainittuja omaa komponenttikirjastoa ja MUI-komponenttikirjastoa. Saan lisäksi versionhallintani sekaisin, ja käytän sen tutkimiseen ja korjaamiseen huomattavan osan päivää.

Käytän myös ison osan päivästä muiden kehittäjien PR:ien katselmointiin ja kommentointiin. Tätä olen tehnyt lähes päivittäin, vaikken ole sitä aiemmin raportoinut. Se on tärkeä osa kehittäjän työtehtäviä, erityisesti jos omat tiketit eivät etene, on hyvä pitää muu tiimi liikkeessä. Kannustan muita ripeämpään katselmointitahtiin. Todellisuudessa kuitenkin omakin katselmointitahtini hidastuu tikettieni monimutkaistuessa – olen saanut aloittaa helpoilla tiketeillä. Tarkastamme myös erään kolmannen kehittäjän PR:n yhdessä toisen kehittäjän kanssa, ja ehdotan toimintamallia yleisemmin käytettäväksi.

Ke 26.1.2022, Sprint 2

Keskiviikkona toinen PR:ni viimein hyväksytään ja yhdistetään yleiseen kehityshaaraan. Avaan tänään kolmannen PR:n projektiin tälle monimutkaisemmalle komponentille. Komponentin kirjoittamisessa tarvitsen kuitenkin paljon apua muilta kehittäjiltä. Päädymme poikkeuksellisesti parikoodaussessioon kanssakehittäjän kanssa, jossa jaan hänelle oman IDE:ni ja muokkaamme samaa projektia yhtäaikaisesti. Tämä temppu oli tuttu opinnoista ja oli hauska päästä hyödyntämään sitä myös töissä, ja se oli uutta myös kollegalle. Tarjoan vastavuoroisesti tyylittelyapua kollegan tiketille.

Olemme FE-kehittäjien kanssa huomanneet, että prototyypin Timeline-komponentista puuttuu mielestämme kannaltamme oleellisia tietoja komponentin käyttäytymisestä äärimmäisissä tapauksissa, ja avaamme keskustelua design-tiimin kanssa aiheesta.

To 27.1.2022, Sprint 2

Torstaina lisäämme lisää riippuvuuksia projektiin, nyt kaikki kovakoodatut tekstit muutetaan käännösfunktiolla toimiviksi. Käännökset tulevat sisäisestä apukirjastosta. Käyn läpi jälleen rutiininomaisesti auki olevat PR:t ja kommentoin niihin ehdotuksia. Usein hyväksyn PR:t vaikka kommentoin

niihin, sillä monet kommenttini ovat enemmän kysymyksiä ja ehdotuksia kuin vaatimuksia. Opettelen edelleen talon tapoja tehdä komponentteja ja yritän hahmottaa mikä ratkaisu on tehty harkiten ja mikä puolestaan vain koska se toimii nyt. Meillä on kehitystavoista myös dokumentaatiota, jonka olen lukenut ennen kuin projekti alkoi, joten kertaan sitä.

Tänään on käyty yleistä keskustelua Scrum-käytännöistä tiimin kanssa. Olen aiemmin huomannut, että tiketit pisteytetään sen kehittäjän toimesta, joka tiketin kirjoittaa. Tikettejä lisätään sprintille tarpeen mukaan jokaisen kehittäjän omalla päätöksellä, mutta tämä toki pitää aina varmistaa ennen tiketin lisäämistä sprintille.

Pe 28.1.2022, Sprint 2

Tänään on FE-kehittäjien yhteinen kahvitauko ja pelaamme pelejä yhdessä. Käytän paljon aikaani PR:ien katselmointiin ja kommentointiin, koska omat tikettini ovat sen verran yksinkertaisia. Myös omaan PR:ni on kommentoitu, ettei se täytä testauskattavuutta. Käytän tämän päivän testauskurssilla opiskeluun. Tänään eri paikoissa keskustellaan myös muuten paljon testaamisesta.

Viikkoanalyysi 3: TypeScript

Projektissa oli käytössä TypeScript ohjelmointikielenä FE-kehitystä varten. TypeScript on JavaScriptin ylijoukko, eli se sisältää kaikki samat toiminnallisuudet, jotka JS:kin, mutta sen lisäksi pelkästään TS:lle ominaisia toiminnallisuuksia (TypeScript 2022a). Dynaamisesti tyypitettynä kielenä JS asettaa kaikille arvoille tyypin vasta ohjelmaa suoritettaessa. Tämä saattaa aiheuttaa monenlaisia ongelmatilanteita, koska aina tyypit eivät ole sitä mitä niitä kirjoittaessa kuvittelisi (Jaya 2018, 6-10). Staattisesti tyypitetty TS ratkaisee tämän eksplisiittisesti määrittelemällä jokaisen arvon tyypin (TypeScript 2022a).

TypeScript tarjosi omat haasteensa hetkeksi, kun kaikkialla oli määriteltävä tyyppi arvoille kuten merkkijono, numero ja totuusarvo. Edellä lueteltujen primitiivisten arvojen kohdalla ongelma oli kuitenkin vielä yksinkertainen (TypeScript 2022b). Projektissa omat tyyppimääritykset tehtiin rajapinnoilla. Toinen vaihtoehto olisi määritellä omia tyyppejä tyyppialiaksella. (TypeScript 2022c.) Komponentin omat ominaisuudet määriteltiin aina Props-nimisellä rajapinnalla, mutta jos komponentissa tarvittiin muitakin tyyppejä, vaikkapa tietyntyyppistä riviä, määriteltiin rajapinta mahdollisimman kuvaavasti nimeämällä ja ominaisuudet kommentteina selitettyinä, esim:

```
interface OtsikkoRiviSolu {
   id: string, // Yksilöllinen solutunniste
   description: string, // Solun tekstisisältö
   onClick?: () => void // Solun onClick-funktio
}
```

Edellisviikolla otettu uusi tiketti pakotti tutustumaan tarkemmin JavaScriptin korkean tason funktioihin ja erityisesti JS:n taulukkometodeihin (Jaya 2018, 22-30). Jouduin pohtimaan kompleksista toistorakennetta ja päädyin käyttämään reduce-funktiota checkbox-listaa luodessani. Käytin esimerkkinäni A Few Minutes Of Code artikkelia (2019). Huomasin, kuinka vähän tunsin JavaScriptiä. Olen ohjelmoinut opinnoissani lähinnä Javalla, ja verkkosivujen kehitystöissä käyttänyt lyhyitä skriptejä joko jQueryllä tai puhtaalla JavaScriptillä yksittäisiä pieniä toiminnallisuuksia varten, yleensä lähes sellaisenaan kopioituina verkossa olevien esimerkkien pohjalta.

Käytimme kaikkialla koodissa nuolifunktioita funktioiden luomiseen, jopa olioiden metodeissa kuten yllä olevassa esimerkissä onClick, josta MDN varoittaa (2022b). Varoitus tosin taitaa liittyä luokkakomponentteihin, eikä funktionaalisiin komponentteihin (Stack Overflow 2022). Funktioiden kirjoittamisen tapoja on JS:ssa lukuisia (Pavlutin 2019). JS:n versiossa ES6 esitellyt nuolifunktiot ovat moderni ja yksinkertainen tapa luoda funktioita (W3Schools 2022).

Tärkeimmät modernin JS:n taulukkometodit ovat .map(), joka palauttaa uuden saman pituisen taulukon olioita, .filter(), joka palauttaa annetuilla kriteereillä suodatetun taulukon, ja .find(), joka palauttaa ensimmäisen kuvaukseen sopivan olion. Muut kehittäjät kertoivat, että .reduce()-metodia, joka palauttaa taulukosta halutun mallisen arvon tai olion, tarvitsee todellisuudessa harvoin, mutta sekin on syytä hallita. (Cepe 2019; Copes 2018.)

4.4 Seurantaviikko 4: Yksikkötestaus

Seuraavan haasteen tarjoaa jo monesti aiemmilla viikoilla mainittu yksikkötestaaminen. Komponentit ja testitapaukset ovat vielä kohtuullisen yksinkertaisia. Käyttöliittymän kompleksisuus kasvaa kuitenkin jatkuvasti. Kuulemani mukaan yksikkötestit aiheuttivat päänvaivaa myös toisille kehittäjille halki projektirajojen. Tämä mielessä olen tehnyt edeltävän viikonlopun perjantaina aloittamaani yksikkötestauskurssia Udemy Busines -alustalla. Tällä viikolla keskityn yksikkötestien kirjoittamiseen.

Ma 31.1.2022, Sprint 2

Heti maanantaina edessä on ensimmäinen testausongelma: lokaalissa kehitysympäristössäni uuden PR:ni testit menevät läpi, mutta etärepositoryn automaattitestaus ei päästä testejä läpi. Selviää, että PR:ssäni on käyttämättömiä muuttujia, ja että linterini ei ole herjannut muistakaan koodissani olevista ongelmista, joista se on kyllä aiemmin ilmoittanut. Mahdollisesti viikonloppuna tekemäni testauskurssin takia jotakin on muuttunut kehitysympäristöni asetuksissa ja kopioin linterisäännöt kollegalta. Joudun jättämään testit kirjoittamatta, ja niitä varten luodaan uusi tiketti.

Ti 1.2.2022, Sprint 2

Tänään on jälleen demo ja retro. Teen myös jälleen uuden PR:n projektiin uudelle painikekomponentille. Tänään käy myös selväksi, että joudumme työskentelemään ad hoc jo päättyneiden projektien tekstikäännösten kanssa. Pohdimme kehittäjien kesken lokalisointia projekteissa ja sitä, pitäisikö joidenkin apukirjastosta tulevien tekstikäännösten olla MFE:ien kesken jaettuja. Jatkamme kuitenkin projekteille yksilöidyillä käännöstiedostoilla.

Edeltävä PR:ni vaativalle checkbox-listalle hyväksytään ja yhdistetään osaksi projektia tänään, ja jatkan sen parissa “select all” -toiminnallisuuden kanssa. Teen tänään myös apukirjastoon PR:n käännöksistä. Retrossa käy selväksi, että BE-tiimi tulisi olemaan kiinni vanhoissa projekteissa vielä hetken, ja että FE-tiimi joutuisi auttamaan myös käännösten kanssa muissa projekteissa. Tänään aloitetaan myös projektin API-määritysten tekeminen.

Ke 2.2.2022, Sprint 3

Keskiviikkona tärkeimmät tapahtumat ovat ominaisuustapaaminen ja sprintin suunnittelu. Päivä alkaa ominaisuustapaamisella, jossa esitellään Timelinen uudet toiminnallisuudet prototyypin avulla. Olimme kommentoineet aiemmin designille, että prototyyppi kaipaisi enemmän äärimmäisiä esimerkkitapauksia.

Ennen sprintin suunnittelua pidämme yhteisen tiketöintisession toisen kehittäjän kanssa. Käymme läpi Miro-boardille luodut kuvat API-kutsuista ja tiketöimme kaikki toiminnallisuudet, jotka mielestämme vielä puuttuvat kehitysjonosta, mutta löytyvät Miro-boardin esimerkeistä. Löydämme kolme puuttuvaa toiminnallisuutta ja komponenttia:

- Laskelmalistausmodaalin aukaiseminen
- MUI Timeline-komponentin POC (Proof of Concept)
- Uuden korjausmodaalin aukaiseminen

Sprintin suunnittelussa meidät ohjeistetaan tekemään käännöksiä muihin projekteihin, koska tilanne omassa projektissa on käyttöliittymän osalta niin hyvä. Joudun nyt ottamaan muita MFE projekteja käyttöön ja harjoittelemaan, kuinka niitä käynnistellään etäkehitysympäristöä taustana käyttäen, koska tarvitsen oikeaa dataa käännöksille. Oma projektimme ei ole vielä tässä vaiheessa, että käyttäisimme oikeaa dataa, joten tämä on uutta.

Aloitan tänään ohjelmoimaan useampaa PR:ää eri projekteihin, tärkeimpänä itselleni laskelmalistauksen implementointi omaan projektiimme. Projektinjohdon mielestä on tärkeämpää kuitenkin muihin projekteihin aloitetut käännös-PR:t, sillä ne tulisivat menemään asiakkaille jo kuun lopussa.

To 3.2.2022, Sprint 3

Torstaina avaan kolme PR:ää kolmeen eri projektiin, joista yksi on omaan projektiimme. Jatkan tänään testien korjaamista. Tänään suuri osa päivästä kuluu eri projektien käännösten jäljittämiseen ja huomaan, että eri projektien legacy-ratkaisut vaativat oman viikkoanalyysinsa. Kaikkien checkboxien valinta yhdistetään tänään projektiin, samoin kuin yksi käännös-PR ja avausnäkymään tehty sloganin käännös kieltä vaihdettaessa.

Pe 4.2.2022, Sprint 3

Tänään painike-PR:ni yhdistetään. Olemme ongelmissa toisen kehittäjän kanssa jälleen testien kanssa, ja vanhempi kehittäjä auttaa meitä testin korjaamisessa. Muuten perjantai on hyvin leppoisa ja sisältää jälleen kehittäjien yhteisen kahvitauon ja pelailua. Kahvitauon jälkeen paneudun jälleen testauskurssiin.

Viikkoanalyysi 4: Yksikkötestaaminen

TDD-filosofian mukaisesti yksikkötestikattavuuden pitää olla 100 prosenttia komponenteissa (Martin 2008, luku 9). Kun komponentteihin lisätään koodia ja toiminnallisuutta, on tarkastettava, että testikattavuus täyttyy jälleen. Testit ja testikattavuus ovat projektien sopimuksellisia seikkoja. On kehittäjien vastuulla pitää huoli testikattavuudesta ja sen tarkastamisesta. Jest-kattavuusraportointi on sisäänrakennettu (Jest 2022).

Udemy Businekseen testauskurssi keskittyy TDD-kehitysfilosofian soveltamiseen. Kuten Johdannossa mainittua, myös Clean Code -klassikko markkinoi TDD-filosofiaa. TDD:ssa testit kirjoitetaan ennen komponentteja, ja kattavuudessa pyritään täydelliseen testikattavuuteen. (Martin 2008, luku 9.) TDD:n kehitystapaa kutsutaan red, green, refactor –viitekehykseksi. Testit kirjoitetaan komponentteja ennen ja ne eivät mene aluksi läpi ja kehittäjä on “punaisessa” vaiheessa. Lopuksi testien mennessä läpi koodi refaktoroidaan. (Codeacademy 2022).

Olen tyytyväinen, ettemme tavoittele tällaista täydellistä testikattavuutta. Minulla menee yksikkötestien kirjoittamiseen karkeasti arvioiden sama aika, mikä itse komponenttien kirjoittamiseen. Toisaalta TDD-testausfilosofia saattaisi saada kehittäjät opiskelemaan enemmän testausta ja näkyisi pienempänä teknisenä velkana ja vähäisempinä virheinä ohjelmistokoodissa nyt ja tulevaisuudessa (Martin 2008, luku 9).

Kaikkia testejä ei tarvitse aina ajaa läpi testejä kirjoittaessa. Vasta ennen PR:n tekemistä koko ohjelmiston kattavuus ja kaikkien testien toimivuus on testattava. Rikkinäisiä testejä ei voi PR:iin jät-

tää, etärepositoryn buildi rikkoutuu tällöin, eikä PR yhdisty lähdekoodiin. Terminaalissa testikomennon perään tiedoston nimeämällä voi tarkastaa yksittäisen komponentin testit. Edelleen .only() -metodia testitiedostossa käyttämällä voi tarkistaa komponentin testeistä yksittäisen testin. Rikkinäistä testiä ei tarvitse välttämättä kommentoida pois, ne voi sivuuttaa .skip() -metodilla. (Jest 2022.)

Tämä helpottaa testaamisessa olennaiseen keskittymistä. Jestin lisäksi projektissa oli käytössä Kent C Doddsin luoma React Testing Library. Rikkinäisen testin debuggaamisessa Testing Libraryn screen.debug() -metodi on kaikista tärkein työkalu. Sen avulla on mahdollista nähdä testin DOM:n tilanne kyseisellä hetkellä. (Testing Library 2022.) Esimerkiksi, jos testin DOM:ssa ei näy painiketta, jonka painamista yrittää testissä simuloida, on testin ongelmakohtaa jo huomattavasti rajattu. Tämä ei vielä toki kerro, miksei painike näy DOM:ssa. (React 2022d.)

4.5 Seurantaviikko 5: Legacy-ratkaisut ja teknologioiden kehityshistoria

Tällä viikolla jo aiemmilla viikoilla mainitut sisarprojekteissa auttaminen sekä Reactin ja JavaScriptin historia nousevat keskiöön. Sisarprojektien tekniset ratkaisut eivät vastaa kaikilta osin omaa projektiamme. Yksikkötesteissäkin riittäisi vielä vaikka kuinka paljon opeteltavaa. Lisäksi tuleva API:en käyttö kummittelee jo valmiiksi takaraivossa. Kaikesta huolimatta, viikon painopiste on Reactin ja Javascriptin rinnakkaisratkaisuissa, joita on käytetty sisarprojekteissa. Viikkoanalyysissa keskitytään eriävien teknologioiden vertailuun.

Ma 7.2.2022, Sprint 3

Maanantaina on ensimmäinen super-projektin pystytysyritykseni. Super-projekti sisältää kaiken ohjelmistossa olevan: käyttöliittymät, rajapinnat ja taustat. Tällä hetkellä työskentelen puhtaasti lokaalissa MFE-kehitysympäristössä, joka ei tiedä todellisesta taustasta tai datasta mitään muuta kuin yhdessä sovitut määritykset.

Super-projektin dokumentaatio on vaativa, koska se on aikojen saatossa muuttunut ja sen versioita on tehty readme-tiedoston lisäksi projektin dokumentaatioon eri paikkoihin ja eri kehittäjien toimesta. Onnistunut super-projektin pystytys mahdollistaisi työskentelyni eri projektien käännösten kanssa ilman riippuvuutta kehityspalvelimen tilasta. Tällä hetkellä käytän kehityspalvelinta taustanani, ja jos se menee pois pelistä, en voi jatkaa kehitystyötäni käännösten osalta.

Ikävä kyllä ensimmäinen pystytysyritys ei onnistu, mutta en anna sen häiritä sillä kehityspalvelin toimii hetken ja teen yhden käännös-PR:n. Kun kehityspalvelin antaa palvelinvirheilmoituksen, jatkan käännösten lisäämistä “sokkona”, koska tiedän osasta käännöksiä mihin tiedostoon kyseiset

käännökset ovat tulossa, ja tarkastan asian palvelimen palattua toimintaan. Jatkan super-projektin pystytysyrityksiä myöhälle iltaan, mutta ilman onnistumista.

Ti 8.2.2022, Sprint 3

Tiistaina teen kaksi käännös-PR:ia, yhden ohjelmiston alkuperäiseen “monoliittiin” ja toisen erääseen MFE:iin. Kaikki ohjelmistokokonaisuuden käyttöliittymämoduulit eivät ole MFE:eja, vaan sen vanhempia osuuksia voisi jakaa MFE:ksi. Alun perin ohjelmistolla on todennäköisesti suunniteltu olevan yksi käyttöliittymä, mutta vaatimusten ja palvelujen kasvaessa on ollut järkevää ruveta pilkkomaan eri moduuleita omiksi kokonaisuuksiksi, jotka eivät ole riippuvaisia toistensa toiminnasta. Tämä mahdollistaa eriävien teknologioiden käytön ohjelmistokokonaisuuden eri moduuleissa.

Aloitan lisäksi kaksi tikettiä eri moduuleihin kuin mihin tein päivän PR:t ja katselmoin muiden kehittäjien PR:iä. Päivän muita teemoja ovat semanttisen HTML5 ja MUI-kirjaston versioiden pohtiminen. Aina silloin tällöin nousee esiin, pitäisikö meidän jo oikeastaan siirtyä käyttämään MUI-versiota 5. HTML5 osalta olemme melko tietoisesti päättäneet jättää hyödyntämättä semanttista HTML5, koska ohjelmiston käyttäjäkunta on niin erikoistunut pieni joukko, ettei esim. saavutettavuusvaatimukset merkitse ohjelmistolle paljoakaan tällä hetkellä. Tilannehan voi toki aina muuttua tulevaisuudessa, joten tällaisiakin asioita on hyvä pohtia.

Ke 9.2.2022, Sprint 3

Keskiviikkona nukun pommiin, mutta onneksi herään ennen päivittäispalaveria. Tänään lisäksi myöhästyn kokouksesta, koska se on vasta luotu ja kertaluontoinen, enkä saa siitä kalenteri-ilmoitusta. Kokouksessa keskustellaan WebSocketin käytöstä Timeline-komponentin yhteydessä. En ole varma, onko kokous tarkoitettu minulle, sillä en tulisi työskentelemään tämän toiminnallisuuden parissa, mutta mielenkiinnosta seuraan sitä silti.

Lisäksi tänään on proton tarkistus projektitiimin kesken. Designin luoma prototyyppi on saanut kehuja projektitiimin ulkopuolella tapahtuneessa demossa, joten Timeline-komponenttiin liittyvät kysymysmerkit vaikuttavat ratkaistuilta. Teen tietoisen päätöksen keskittyä käsillä oleviin muihin ongelmiin, sillä tätä ns. tärkeämpää asiaa ratkoo jo monta paljon kokeneempaa tekijää.

Minulla on tällä hetkellä kolme käännöksiin ja kuun lopulla tulevaan julkaisuun liittyvää PR auki, ja pyydän muita kehittäjiä katselmoimaan ne, että niissä päästään eteenpäin. Lisäksi sovimme, että toinen tiimin kehittäjistä ottaa yhden MFE-moduulin käännökset hoitaakseen, sillä hän on ollut sen projektin alkuperäisessä kehitystiimissä ja tuntee projektin entuudestaan. Tämä helpottaa omaa työtaakkaani käännösten osalta huomattavasti.

To 10.2.2022, Sprint 3

Torstaina saan kyselyä käännöstikettien tilanteesta tuon projektin tuoteomistajalta. Opin viimeistään tänään, että ohjelmiston uusi julkaisu valmistuu kehityspalvelimelle aina öisin. Tänään kuitenkin poikkeuksellisesti julkaistaan uusi julkaisu jo kesken päivän. Muutosteni pitäisi olla näkyvissä jo tässä julkaisussa. Saan tänään myös poikkeuksellisen paljon apua vanhemmalta kehittäjältä flow-tyypityksien kanssa. Myös checkbox-listan “valitse kaikki” -toiminnallisuus vaatii jatkokehittämistä.

Pe 11.2.2022, Sprint 3

Perjantaina on jälleen FE-kehittäjien yhteinen kahvitteluhetki iltapäivällä. Muuten käytän päivän oman projektimme ja sen riippuvuuskirjastojen parissa. Huomaan, että laskelmalistaukseen liittyvien komponenttien ja ominaisuuksien viemistä ei tehdä riippuvuuskirjastossa, ja lisään sinne viennit näille puutteille. Tänään myös jatkan eilen alkanutta “valitse kaikki” -toiminnallisuutta.

Viikkoanalyysi 5

Tämä sprint oli melkoista salapoliisityötä selvittää missä komponentissa ja kohdassa kukin käännös tapahtui ja oliko käännös FE:n vai BE:n vastuulla. Käytössä oli Miro-board jossa käännösten puutteet oli kuvattu. Erilaisia syitä mistä käännöksen uupuminen saattoi johtua:

1. FE oli kovakoodannut tekstin ilman käännösfunktiota
2. FE käytti käännösfunktiota mutta käännöstiedostosta uupui käännös
3. Käännöksen olisi pitänyt tulla taustasta mutta se uupui sieltä
4. Käännös oli oikein paikoillaan kaikilta osin, mutta käyttöliittymä ei päivittynyt kieltä vaihdettaessa

Viimeinen vaihtoehto monoliittimaisessa legacy-projektissa oli hankalin, sillä komponentin löytäminen missä varsinainen funktio ja riippuvuus kielen vaihtoon sijaitsi, oli monesti useiden komponenttien päässä varsinaisesta komponentista missä itse tekstikenttä sijaitsi. Eräänlaista sekavuutta työskentelyyn aiheutti tällainen sivusta tullut avustuspyyntö varsinaisen projektin kanssa, joka juuri oli alkanut ja jonka omatkin kehitystavat olivat vasta muodostumassa. Toisaalta tämä harhapolku auttoi hahmottamaan ehkä tietyllä tavalla kokonaisuutta.

Kuten jo aiemmassa viikkoanalyysissa mainittu, React Hookeja edelsivät React elinkaarimetodit. Aiemmin komponentin tilan muuttaminen oli mahdollista vain luokkakomponenttien elinkaarimetodien avulla. Lisäksi Context API:n sijasta monoliitissa oli käytössä Redux komponenttien tilojen hallitsemiseksi, ja TypeScriptin sijasta monoliitissa on käytössä Flow-tyypitykset. TypeScript on

Flow:ta vanhempi teknologia, ja nykyisin huomattavasti Flow:ta laajemmin käytössä (Ramirez 2021). Erot teknologioissa olivat siis:

- Arkkitehtuurisena ratkaisuna monoliittinen käyttöliittymä MFE:n sijasta
- Luokkakomponentit funktionaalisten komponenttien sijasta
- React elinkaarimetodit React Hookien sijasta (taulukko 2)
- Flow-tyypitykset TypeScriptin sijasta (taulukko 3)
- Redux-tilanhallinta Context-API:n sijasta (taulukko 4)

Taulukko 2. Reactin elinkaarimetodien ja –hookien vertailu (mukaillen crossML engineering 2020)

Reactin elinkaarimetodit	**Reactin elinkaarihookit**
Esitelty ReactJS:n alkuperäisissä versioissa 2013	Esitelty 2018 versiossa 16.8
ES5 yhteensopiva	Yhteensopiva ES6 eteenpäin
Käytetään luokallisten komponenttien kanssa	Käytetään funktionaalisten komponenttien kanssa
Vaatii konstruktorin	Ei tarvitse konstruktoria
Vaatii ‘this’-avainsanan käyttöä tilan päivittä-miseksi	Ei tarvitse ‘this’-avainsanaa

Taulukko 3. TypeScriptin ja Flown vertailu (mukaillen Ramirez 2021)

	TypeScript	**Flow**
Kehittäjä	Microsoft	Facebook
Julkaistu	2012	2014
Yhteisö	Suuri	Pieni
Suorituskyky	Nopein ja vähäbugisin	Joitain raportoituja muistion-gelmia

Stack Overflow –kysymykset	+150000	n. 900
Tekniikan käyttöönotto projekteissa	Valmiina useissa kirjastoissa	Yleensä Babel-kääntäjän avulla

Taulukko 4. Reduxin & Context API:n vertailu (mukaillen Bose 2021)

Context API	**Redux**
Tulee Reactin mukana	Erillinen apukirjasto, kasvattaa buildin kokoa
Minimaalinen asennus	Laaja asennus
Suunniteltu staattiselle datalle	Toimii loistavasti staattisella ja dynaamisella datalla
Lisättävät contextit luotava tyhjästä	Laajennettavissa helposti alkuasennuksen jälkeen
Debuggaus haasteellista laajoissa komponenttirakenteissa	Redux Dev Tools voimakas työkalu debuggaukseen
Käyttöliittymä- ja tilanhallintalogiikka samassa komponentissa	Erillinen käyttöliittymä- ja tilanhallintalogiikka

Funktionaalinen komponentti on yksinkertaisin tapa luoda React-komponentti (React 2022a). Funktionaalisen komponentin muuttaminen luokalliseksi komponentiksi tapahtuu viidessä askeleessa:

1. Luo ES6 luokka ja nimeä se.
2. Lisää elinkaarimetodi render().
3. Lisää funktionaalisen komponentin body render()-metodiin.
4. Korvaa props this.propsilla render() bodyssa.
5. Poista jäljelle jäänyt tyhjä funktio. (React 2022e.)

Näiden teknologisten erojen lisäksi myös kehitystyyli monoliittimaisessa käyttöliittymässä oli erilainen. Yksittäiset komponentit ovat tyypillisesti useiden satojen rivien pituisia. Uudessa projektissa

komponenttien pituus pyrittiin pitämään mahdollisimman lyhyenä, usein parisataa rivisinä (React 2022f).

4.6 Seurantaviikko 6: REST-rajapinnan käyttö

Ensimmäiset viisi raportointiviikkoa sijoittuivat projektin alkuun vuoden 2022 kalenteriviikoille 2-6 ja seurasivat toinen toistaan. Raportointiviikolla kuusi hypätään kahden kuukauden päähän edellisestä raportointiviikosta. Tällä raportointiviikolla alkaa suurempi komponenttikokonaisuus, jonka kehitystä seurataan projektiraportin loppuun. Tällä viikolla alkaa myös FE-kehittäjien “viralliset” tapaamiset yli projektirajojen yhteisten kahvitaukojen lisäksi. Näissä tapaamisissa tutustutaan sisarprojektien työhön PR ja kooditasolla. Tämän raportointiviikon tavoitteena on kuitenkin esitellä REST-API:n käyttö projektissa.

Raportoimatta jääneissä sprinteissä olin oppinut käyttämään sekä projektin että aiempien projektien REST-rajapintoja. Olin aloittanut aiemmilla viikoilla Ubuntu Business -kurssin OpenAPI3 määrityksistä. Olin myös jo työskennellyt API-pyyntöjen kanssa projektissa tässä vaiheessa.

Ma 11.4.2022, Sprint 7

Viikko alkaa poikkeuksellisesti ominaisuustapaamisella. Design-tiimi ei kuitenkaan ole maanantaina paikalla ja sisältö hoitaa uuden ominaisuuden esittelyn. Siirrymme tänään myös versionhallinnassa pilvialustan käyttöön, joka vaatii kaksivaiheisen autentikoinnin. Tähän käytän huomattavan osan työaikaa aamupäivästä. Tänään myös checkbox-lista komponentti on refaktoroitu kokonaisuudessaan uudestaan. Nykyinen komponentti on huomattavasti helppotajuisempi ja käytettävämpi.

Ti 12.4.2022, Sprint 7

Tiistai-iltapäivällä on jälleen demo ja retro. Retrossa nousee esiin toive FE-BE integraation kokeilemisesta. Tällä hetkellä työskentelemme edelleen täysin mock-testidatan kanssa FE:ssa. Teen tänään myös ensimmäisen PR:n laskelmalistaus-komponenttiin, jonka parissa tulen työskentelemään seuraavat kuukaudet. Törmään tätä komponenttia aloittaessa siihen, että MFE on konfiguroitu vain yhdelle, projektin omalle, mock-API:lle.

Laskelmalistaus-komponentin on kuitenkin tarkoitus keskustella myös laskelmat-API:n kanssa. Keskustelemme kuinka tämä olisi hyvä ratkaista. Huomaan, että aiemmassa komponentissa, jossa on tarvittu toista rajapintaaa, ohjelmakoodi on kirjoitettu valmiiksi mutta kommentoitu pois. Keskustelun tuloksena päädymme olemaan konfiguroimatta muita rajapintoja osaksi oman projektimme

MFE:ia ja jättämään toisiin rajapintoihin lähetettävät kutsut kommentoimattomiksi, vaikka ne eivät menekään läpi.

Ke 13.4.2022, Sprint 8

Keskiviikko alkaa kuten joka toinen viikko ominaisuustapaamisella. Timeline-komponentti on edelleen tapaamisen keskiössä. Keskityn itse kuitenkin tänään jokaisesta MFE:sta löytyvään laskelmalistaukseen. Jokaisessa MFE:ssa laskelmalistaus on luotu kuitenkin aina omalla tavallaan. Suoraa ratkaisua muista projekteista ei saa, mutta niistä voi katsoa paljon esimerkkiä.

Ominaisuustapaamista seuraa sprintin suunnittelu. Sitä ennen pidämme ehdotuksestani FE-kehittäjien kanssa yhteisen tiketöintisession edellissprintin hyvien kokemuksien vuoksi. Kävimme edellissprintin tiketöintisessiossa läpi Miro-boardilla olevat API-kutsut, ja tiketöimme kaiken minkä keksimme sen pohjalta. Teemme jälleen samoin. Tikettejä, jotka oli suunniteltu viime sessiossa, ei ollut kuitenkaan luotu ja lisätty sprintille, joten se tehtiin nyt.

Tänään keskustelemme myös PR-etiketistä FE-kehittäjien kesken ja siitä, milloin on soveliasta laittaa “needs work” -merkintä. Jos koko PR:ää ei ole katselmoitu silloin vielä kokonaisuudessaan, se voi viivästyttää prosessia turhaan, jos jokainen muutosvaatimus käydään yksitellen läpi. Tänään luomassani PR:ssa CRUD:sta luodaan ensin “Create” eli luo-toiminnallisuus laskelmalistauskomponenttiin. PR:n koko on kasvanut 25 tiedostoon ja sisältää mm. 100-rivisen uuden testitiedoston.

To 14.4.2022, Sprint 8

Tänään aloitamme FE-kehittäjien viralliset keskinäiset tapaamiset yli projektirajojen, joissa käsitellään eri projektien PR:eja ja tutkitaan kooditasolla erilaisia ratkaisuja. Muut FE-kehittäjät projektitiimistäni jäävät kuitenkin pääsiäislomalle jo puolenpäivän aikoihin. Projektimme tilanteen esittely jää siis vastuulleni. Esittelen vanhemman kehittäjän kanssa käydyn keskustelun pohjalta laskelmalistausta. Pohdin kokouksessa myös käyttöliittymäarkkitehtuuria ja kuinka MFE:t yhdistetään osaksi käyttöliittymän pääohjelmaa.

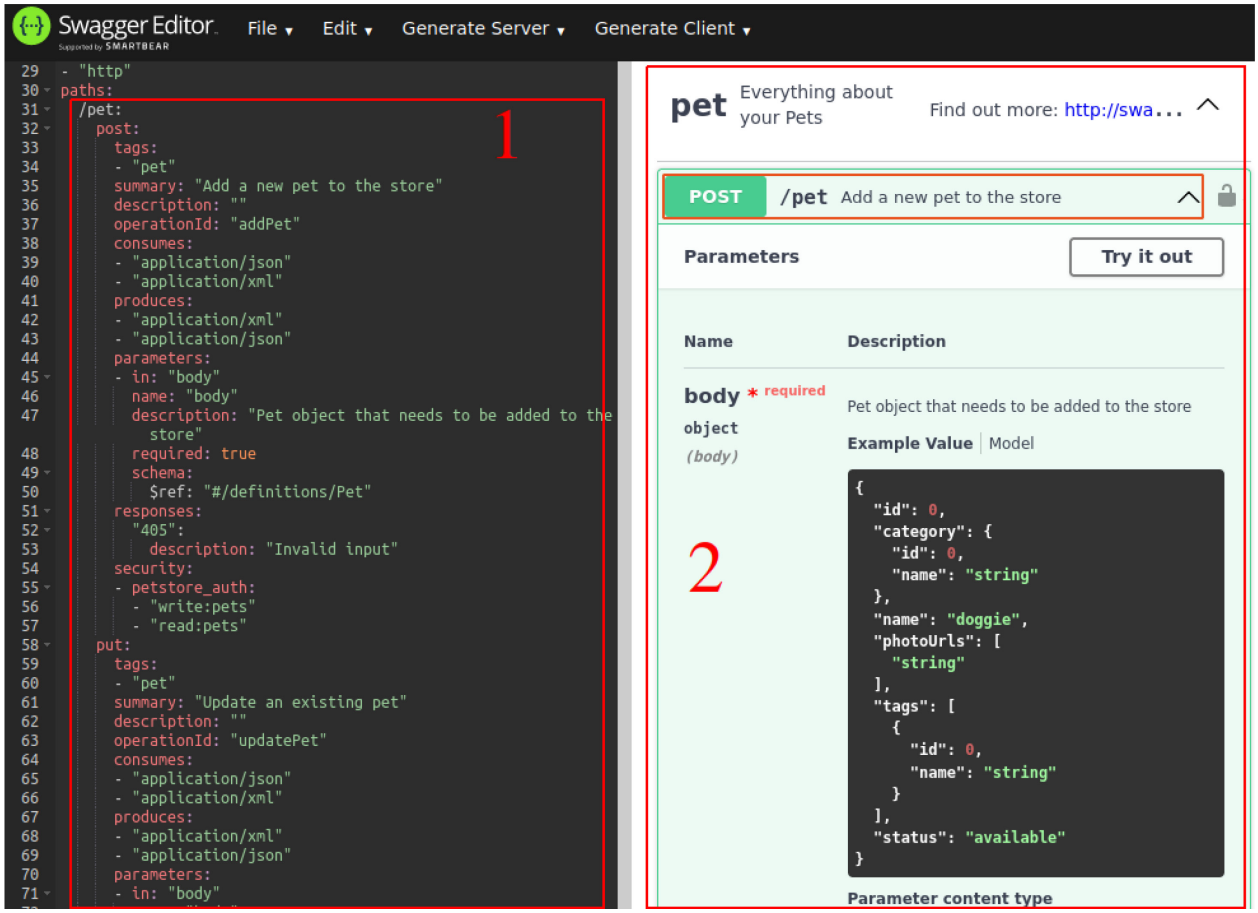
Tänään aloitetaan käyttämään mock-API:n sijasta oletuksena varsinaista projektin uutta REST-rajapintaa. Koska taustaan yhteydessä olevasta REST-rajapinnasta tulee vielä sen verran vähän dataa, käytetään kehityksessä edelleen mock-API:a.

Pe 15.4.2022, Sprint 8

Perjantai 15.4. on pitkä perjantai ja pääsiäisloma. Kehotan kollegoita tutustumaan pääsiäisen laskemisen algoritmin historiaan ja sen JS-implementaatioon pitkän viikonlopun aikana.

Viikkoanalyysi 6

Tällä viikolla keskiöön nousi ohjelmarajapintojen käyttö. Käyttöliittymän ja taustan välisen kommunikoinnin mahdollistamiseksi on tehty paljon määrityksiä ennen varsinaisen ohjelmarajapinnan, taustan tai käyttöliittymän toteuttamista. OpenAPI3-määritykset kertovat kehittäjille, miltä REST-rajapintakutsut ja sieltä saatava data näyttävät. OpenAPI3-määritykset sisältäviä tiedostoja voi kopioida verkossa sijaitsevaan Swagger-editoriin, joka luo määrityksistä helposti luettavan dokumentaation (kuva 7).

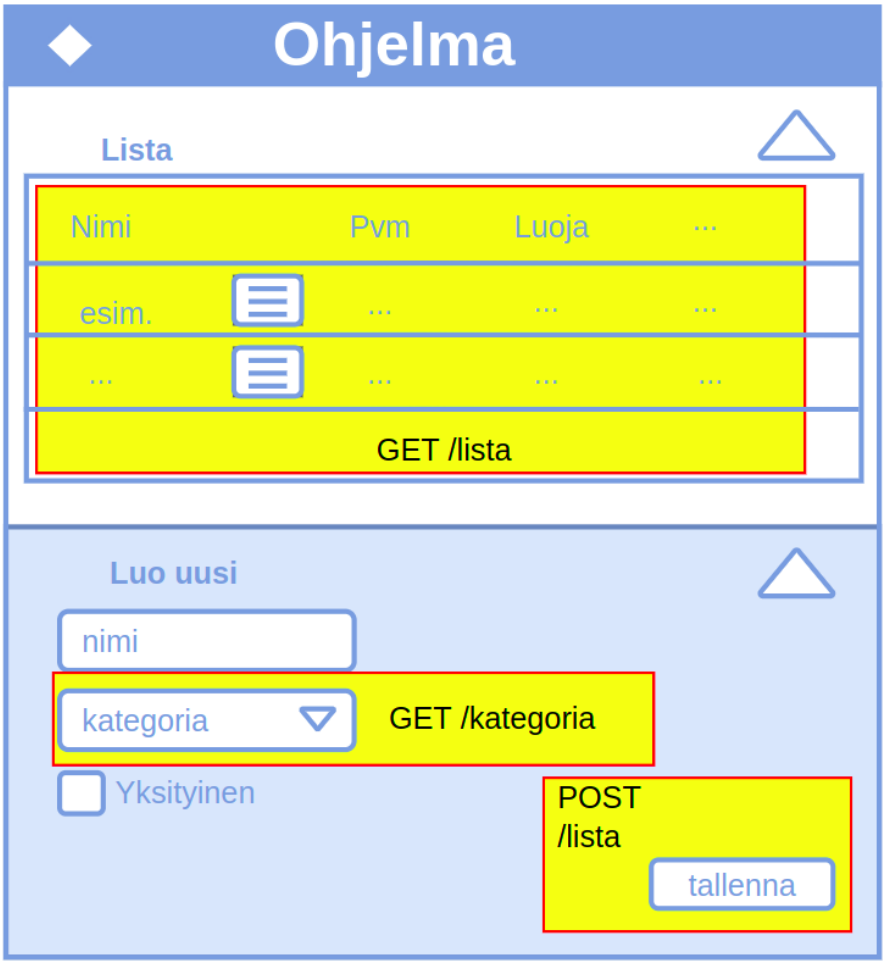


Kuva 7. Kuvakaappaus Swagger-editorista (Swagger 2022)

Editorin vasemmalle puolelle sijoitetaan OpenAPI3-määritykset sisältävä tiedosto JSON tai YAML muodossa, ja editorin oikealle puolelle generoituu rajapinnan sisältämät päätepisteet ja esimerkkidata (Swagger 2022). Vasemmalla kuvassa 7 YAML-koodi ja oikealla generoitu dokumentaatio.

Dokumentaatiossa REST-rajapinnan /pet-päätepisteen POST HTTP-pyyntö vaadittuine parametreineen käyttäjäystävällisesti esitettynä.

Tämän lisäksi projektin alkuvaiheessa luotiin kuvalliset ohjeet mikä rajapintakutsu on tarkoitus tehdä missäkin käyttöliittymän osassa. Ohjeet luotiin Miro-boardille ottamalla kuvakaappauksia prototyypistä ja lisäämällä tekstiselitteet tehtävistä rajapintakutsuista. Näin ohjelmarajapinnan määritykset ja ohjeistukset olivat olemassa jo ennen kuin sitä oli miltään osin toteutettu ja kehittäjät tiesivät mitä rajapinnalta odotetaan. Alla olevassa kuvitteellisessa kuvassa 8 Miro-ohjeistus on punaisella reunalla ja keltaisella taustalla rajattuna. HTTP-pyyntö ja sen päätepiste on mustalla tekstillä.



Kuva 8. Esimerkki Miro-ohjeistuksesta

Rajapintakutsuissa käytettävä data esitetään JSON-muodossa. Projektissa on käytössä Prism mock-API, joka luo mock-rajapinnan OpenAPI3-määritysten pohjalta. Näin käyttöliittymäkehittäjien ei tarvitse odottaa taustakehittäjiä uuden REST-rajapinnan ja siihen liittyvän taustan kehittämisen

kanssa. Mock-rajapintaan tehtävät HTTP-pyynnöt ja pyynnöissä käytettävän JSON-datan skeema on määritelty OpenAPI3-spesifikaatiolla. Kun uusi REST-rajapinta on valmis, lähetetään kutsut sinne mock-API:n sijasta. Käyttöliittymäkoodissa ei tarvitse tehdä muutoksia käytettävää rajapintaa vaihtaessa. (Stoplight 2022.)

REST-rajapintakutsujen apuna MFE:ssä käytettiin Axios-kirjastoa. Toinen mahdollisuus olisi käyttää JS:n sisäänrakennettua Fetch-rajapintaa ja fetch()-metodia. (GeeksforGeeks 2021.) Tyypillinen tilanne käyttöliittymässä oli, että ensin komponenttiin haetaan dataa GET HTTP-pyynnöllä. Datan säilömistä varten Reactissa käytetään komponentin tilaa ja ominaisuuksia (React 2022g). Yleensä rajapintakutsu eli datan hakeminen taustasta tehdään komponentin alustuksessa tai kun käyttäjä tekee jotakin, esimerkiksi painaa painiketta.

UseEffect-hook kirjaimellisesti käyttää efektiä ja se on käytännöllistä, kun haluamme tehdä käyttäjän puolesta asioita. UseEfectit ottavat kaksi argumenttia, ensimmäinen on varsinainen funktio ja toinen vapaaehtoinen riippuvuustaulukkoparametri (engl. dependency arrays). Riippuvuustaulukot ovat yleisimmin meillä komponentin tiloja, joiden muutoksia riippuvuustaulukot seuraavat. Mikäli riippuvuustaulukoissa oleva tila muuttuu, funktio suoritetaan uudestaan. Näin voidaan hakea vaikkapa listadata aina uudestaan, kun yksittäistä listariviä muokataan. Riippuvuustaulukkoargumentti on tyhjä taulukko, jos useEffect halutaan suorittaa vain komponenttia alustaessa.

GET:llä saatu data yleensä tallennetaan komponentin tilaan, joka lopulta palautetaan komponentin palautuslausekkeessa React-komponentteina ja HTML-elementteinä, tai annetaan komponentin lapsikomponenteille ominaisuuksina. Yleisimmin käytetään JS:n map()-metodia, joka voi sisältää huomattavan määrän muita taulukkofunktioita ja muita JS:n korkean tason funktioita.

4.7 Seurantaviikko 7: REST-rajapinnan kehittäminen

Raportointiviikkojen välissä on jälleen väliä, tällä kertaa kuitenkin vain pari viikkoa. Viime raportointiviikon analyysissa keskityttiin REST-rajapinnan käyttämiseen projektissa. API:ssa huomattiin kuitenkin puutteita, ja sitä oli jatkokehitettävä. Tällä raportointiviikolla keskitytään REST-rajapinnan kehittämiseen projektissa. Sitä varten tarvitaan .NET-kehitysympäristöä ja C#-ohjelmointikieltä.

Ma 2.5.2022, Sprint 9

Maanantaina aloitan rajapinnan muokkausyritykset. Olen aiemmin pyytänyt ohjeistusta rajapintamuutosten tekemiseen, mutta koska se ei ole FE-tiimin vastuulla ja kehittäjillä on kiire, en ole saanut toistaiseksi siihen apua. FE-tiimin muut kehittäjät eivät ole aiemmin tehneet näitä muutoksia.

Tiimiä mentoroiva vanhempi FE-kehittäjä antaa tänään kuitenkin ohjeet API-muutosten tekemiseen, ja teen niiden pohjalta PR:n, mutta BE vaatii korjauksia PR:iin. Rajapintojen kehittäminen on muuttunut saamani ohjeistuksen ajoista entistä automatisoidummaksi. Teen uudesta käytännöstä dokumentaation FE-kehittäjille.

Ti 3.5.2022, Sprint 9

Tiistaina olen sairaana. Tällä kertaa pysyttelen myös pois työviestimistä. Nukun koko päivän, mutta seuraavana päivänä palaan jo töihin levon voimistamana.

Ke 4.5.2022, Sprint 9

Keskiviikkona jatkan kesken jäänyttä rajapinnan muokkausta. Saan tänään apua BE-kehittäjältä .NET-ympäristön konfiguroinnissa. Seuraan ensin Microsoftin ohjeita VS Coden .NET-ympäristön pystytykseen ja sitten jatkamme BE-kehittäjän kanssa konfigurointia, mm. tarvittavien käyttöoikeusasetuksien suhteen. Lopulta kaiken pitäisi olla oikein asennettuna, ja .NET ”hello world” toimii. Kuitenkin Swagger-tiedostojen generoiminen epäonnistuu. .NET-ympäristön pystytys ei tuota täten tulosta, koska varsinainen pihvi, Swagger-tiedostojen autogenerointi ei toimi. Kuitenkin opin periaatteen ja pystyn tekemään tiedostot, joiden pohjalta BE generoi Swagger-tiedostot.

To 5.5.2022, Sprint 9

Laskelmalistauksen alustus tehtiin jo projektin ensisprinteillä. Alustus ei sisältänyt todellista lukutoiminnallisuutta vielä, vain apukirjastosta tuodun kehikon, johon varsinainen lista myöhemmin upotettaisiin. Ensimmäinen CRUD-toiminnallisuus “Create” luotiin puolestaan edellisessä sprintissä. Tänään avaan PR:n toiselle CRUD:n toiminnallisuudelle, oikean muotoisen datan lukemiselle. Kuten aiemmilla raportointiviikoilla todettua, oma listakomponenttimme on kokonainen erillinen apukirjasto, jonka alustus laskelmalistaukseen vaatii työtä.

Avaan lisäksi PR:n rajapintamuutoksia hyödyntävään toiminnallisuuteen, joka kertoo mikä rakennuskokonaisuuden simuloitu prosenttiarvo on käyttäjälle. Pidämme myös yhteisen tiketöintisession designin kanssa uusien tikettien luomiseksi prototyyppimuutosten pohjalta.

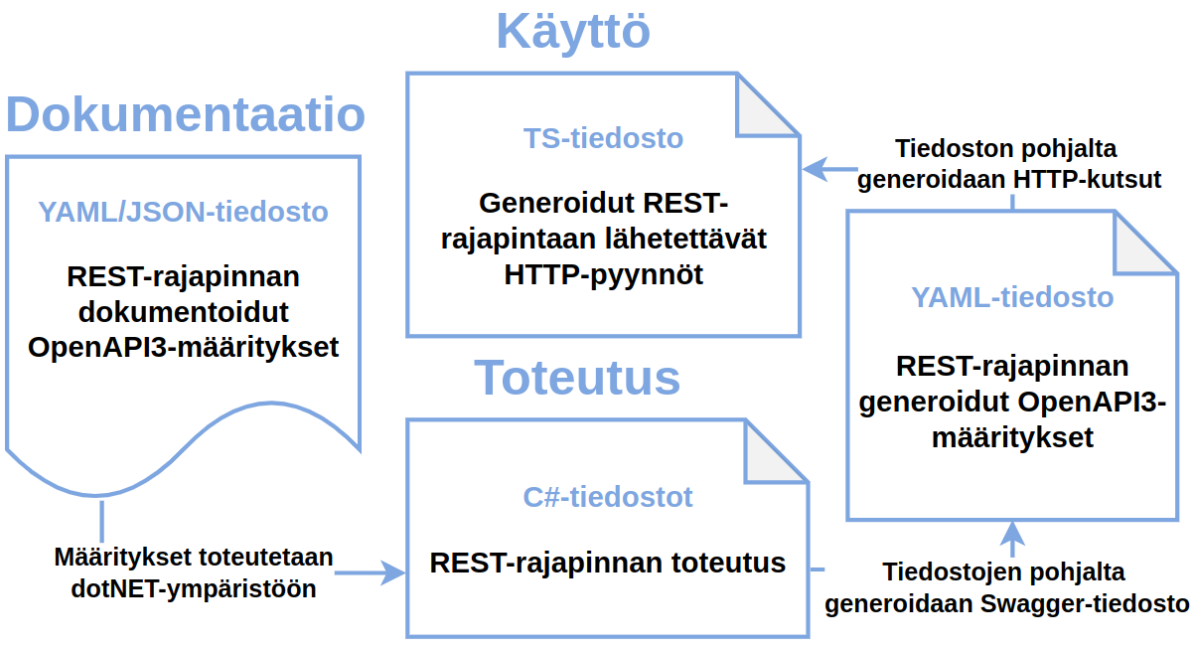
Pe 6.5.2022, Sprint 9

Perjantaina on toinen FE-kehittäjien yhteinen palaveri. Keskiviikkona pidetyssä edeltävässä palaverissa on tarkoitus käydä läpi projekteihin tehtyjä PR:iä kooditasolla yhteisesti. Tässä jälkimmäisessä puolestaan keskustella yleisemmistä kehitystavoista tai jatkaa keskustelua aiemmassa tapaamisessa käsitellyistä teknologioista. Käytännössä ensimmäisessä tapaamisessa käydään läpi

projekteihin yhdistetyt PR:t ja valitaan niistä yksi lähempään tarkasteluun. Tätä jälkimmäistä tapaamista varten kehittäjät ehdottavat keskustelunaiheita, jotka he sitten esittelevät. Ehdotin PR-käytäntöjen pohdintaa päivän aiheeksi. Tänään oli myös SignalR-tapaaminen Timeline-komponenttia varten. Olen mukana tapaamisessa kuunteluoppilaana.

Viikkoanalyysi 7

Edellisviikon analyysissa mainittiin Swagger-tiedostot. Käyttöliittymäprojektin konfiguraatio generoi REST-rajapintaan lähetettävät HTTP-pyynnöt automaattisesti Swagger-tiedostojen pohjalta. Generoinnin etuna on myös kirjoitusvirheiden minimoiminen kutsuissa. Generoituja rajapintakutsuja oli vajaa 800 riviä projektin lopussa. Toinen vaihtoehto olisi kirjoittaa Axiosilla (kts. viikkoanalyysi 6) tehtävät HTTP-kutsut käsin. Näin olikin tehtävä muihin kuin projektin omaan rajapintaan tehtäville HTTP-pyynnöille. Projektille oli konfiguroitu vain yksi REST-rajapinta, projektin uusi oma ohjelmarajapinta. Muihin ohjelmakokonaisuuden REST-rajapintoihin lähetettävät HTTP-pyynnöt oli ohjelmoitava käsin, eikä niihin voinut odottaa kehitysvaiheessa mock-API:sta onnistuneita vastauksia. (Stoplight 2022b.)



Kuva 9. REST-rajapinnan käyttö- ja kehitysprosessi

Swagger-tiedostot haetaan MFE:iin API-repositorysta, jossa Swagger-tiedostot myös generoidaan ja johon REST-rajapintaa kehitetään (kuva 9). Aiemmin Swagger-tiedostoja saattoi muokata käsin, mutta nykyisin myös Swagger-tiedostojen generoiminen on automatisoitu C#-tiedostojen pohjalta. Swagger-tiedostojen generoimiseen C#-tiedostoista tarvitsee .NET-ohjelmointiympäristön.

Swagger-tiedostot voivat sisältää myös esimerkkidataa HTTP-pyyntöihin liittyviin vastauksiin. Käyttöliittymäkehittäjät ovat yleensä kiinnostuneita tästä esimerkkidatasta, sen pohjalta generoidaan komponentin elementtejä. YAML-muodossa olevan esimerkkidatan pohjalta voi kääntää JSON-tiedostoja käytettäväksi kehitystyössä.

API-repositoryssa sijaitsevat C#-tyypit, joiden pohjalta Swagger-tiedostot generoidaan debug-komennolla. Sieltä ne tuodaan puolestaan MFE:iin. Swagger-tiedostojen pohjalta MFE:n generaattoriskripti generoi rajapintakutsut projektiin. Tässä käytettävä teknologia ja syntaksi jäi itselleni vieraaksi, enkä joutunut muokkaamaan generaattoriskriptin koodia rajapintaa kehittäessäni. Rajapintakutsuja on projektissa jo tällä hetkellä kymmeniä ellei satoja. On hyvä, ettei niitä tarvitse käsipelin koodata.

4.8 Seurantaviikko 8: Komponenttikokonaisuus ja CRUD-operaatiot

Projekti rupeaa olemaan päätöksessään ja suuria muutoksia sen lähdekoodiin ei ole tarkoitus enää tehdä ennen viimeistä demopäivää. Edellisen raportointiviikon ja viimeisen raportointiviikon välissä on yksi raportoimaton viikko. Päädyin projektin loppuun mennessä kehittämään kaikkia CRUD-operaatioiden toiminnallisuuksia käyttöliittymän laskelmalistauskomponentille.

Ma 16.5.2022, Sprint 10

Maanantaina avasin CRUD:n ”Delete”-toiminnallisuuteen liittyvän PR:n. Laskelmalistauskokonaisuudesta puuttuu edelleen käännökset ja niiden toteutustapa pohdituttaa. Ymmärrän tarpeen laskelmalistauskomponentin käännösten yhtenäisyydelle. Tällä hetkellä jokainen MFE on implementoinut oman käännöstiedostonsa laskelmalistaukselle. Erityistä kummastusta aiheuttaa tänään myös useEffectin käyttö sisarprojekteissa, ja päädyn tutkimaan asiaa. Tavallisesti käyttäjien tekemät toiminnallisuudet ovat omissa funktioissaan, mutta sisarprojekteissa on päädytty muokkaamaan tiloja, joita riippuvuustaulukot seuraavat. Aiemmilta raportointiviikoilta tuttu super-projektin tarve korostuu tänään jälleen. Seuraavassa projektissa super-projekti tulee olemaan välttämättömyys.

Ti 17.5.2022, Sprint 10

Tiistaina julkaistaan uudet projektijaot. Pääsen mukaan omasta mielestäni mielenkiintoisempaan projektiin, mihin olen tyytyväinen. Uusissa suunnitelmissa on jättää päättyneiden projektien jatkokehitys kaikkien kehittäjien vastuulle kiertävästi. Tästä sain hieman esimakua projektin alkupuolella. Tämän rotaation huono puoli on ongelmallisuus, mitä kuvasin jo aiemmin päiväkirjassa. Kehittäjät joutuvat kesken oman projektinsa eri teknologiaratkaisujen ja sisältöjen pariin, joita he eivät tunne. Samalla he putoavat kärryiltä oman projektinsa kehityksestä. Asialla on myös hyviä puolia. Kehittäjät tulevat tutuiksi laajempien kokonaisuuksien kuin pelkkien omien meneillä olevien projektiensa kanssa.

Tänään saan myös viimein super-projektin pystytettyä. Ongelma oli Python-versiossani. Siihen nähden kuinka yksinkertaisena kielenä Pythonia mainostetaan, sisältää sen asentaminen kuitenkin useamman askeleen kuin toivoisi.

Ke 18.5.2022, Sprint 10

Keskiviikkona vanhempi FE-kehittäjä esittelee meille Timeline-komponentille tulevan WebSocketin käyttöä. Tiedän toimintaperiaatteen ja tarpeen, mutta tarkempi tekninen tutustuminen asiaan tulee jäämään projektin jälkeiseen aikaan itselleni. Vaikka CRUD:n tärkeimmät toiminnallisuudet rupeavatkin olemaan paikallaan, puuttuu kokonaisuudesta edelleen testejä ja käännöksiä. Laskelmalistaukseen on tullut nyt kaksi muutosta: oikea data ja komponenttikokonaisuuden refaktorointi. Testit hajoavat ja vuotavat muihin komponentteihin, mikä tekee korjaamisesta työlästä. Lopulta palaaminen askel kerrallaan takaisin päin auttaa korjaamaan testit.

To 19.5.2022, Sprint 10

Torstaina on jälleen kehittäjien yhteinen tapaaminen. Tapaamisessa keskiöön nousee Timeline-komponentti ja WebSocketin käytön esittely muille projektitiimeille. Siirryn kesken työpäivän toiseen työskentelypisteeseen ja päivä on hieman rikkonainen. Päivitän auki olevan PR:ni ja rupean työskentelemään viimein puuttuvien käännösten kanssa. Päivästä ison osan haukkaa myös eilen aloitettu rikkinäisten testien jäljittäminen.

Pe 20.5.2022, Sprint 10

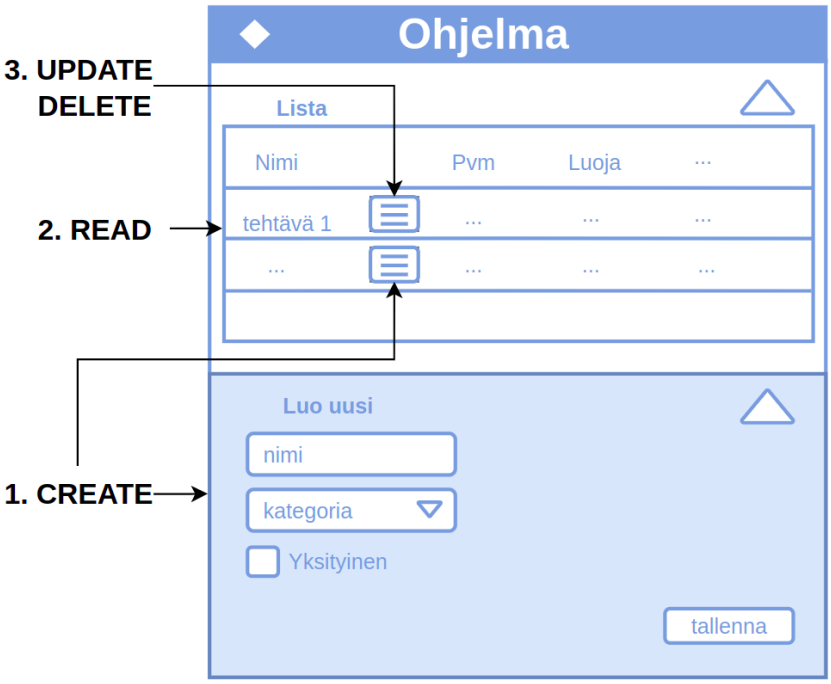
Tänään avaan PR:n, jossa laskelmalistauskomponentti refaktoroidaan suurelta osin, ja jossa samalla korjataan luomiseen liittyviä bugeja. Ensi viikolla toteutetaan projektin viimeinen demo yleisessä demopäivässä. Uusia muutoksia ensi maanantain jälkeen ei projektiin oteta kuin pakosta.

Laskelmalistauskomponentissa olisi mielestäni vielä paljon kehittämistä, mutta tulevan demon kannalta se ei ole juuri nyt oleellista. Komponentin kehittäminen jää hiukan kesken, ja olisin halunnut jatkaa sen työstämistä täysin valmiiksi, joka ei ole enää mahdollista.

Ensi viikolla on viimeinen demo ja tuotteen julkaisu, joten seuraamme listaa tarvittavista asioista. Laskelmalistauskomponentti, jonka parissa olen työskennellyt, ei ole siinä. Tämä johtuu siitä, että laskelmalistauskomponentti ei ole ainutlaatuinen vain tälle MFE:lle ominainen toiminnallisuuskokonaisuus, eikä sitä esitellä demossa, ja siitä että sen ensimmäisen prioriteetin toiminnallisuudet ovat jo toiminnassa.

Viikkoanalyysi 8

CRUD on kirjainlyhenne sanoista Create, Read, Update and Delete. Se on yleinen rajapintoihin liittyvä operaatiokokoelma (Mursu 2016, 13). CRUD:n update-osuus oli laskelmalistauskomponentissa ehdottomasti vaativin ja vastaisi laajuudeltaan koko muuta komponenttia valmistuessaan. Se sisälsi useita komponentteja ja rajapintakutsuja, ja se jäi kesken projektin päättyessä. Se ei kuitenkaan ollut projektin ensimmäisen prioriteetin toiminnallisuusvaatimuksien listassa, toisin kuin laskelmalistauskomponentin Create ja Read, jotka tulivat onnistuneesti valmiiksi. Create sisälsi laskelman kopioimisen ja uuden laskelman luomisen. Delete eli laskelman poistaminen tuli myös valmiiksi projektin päätökseen mennessä.



Kuva 10. Esimerkki käyttöliittymästä siihen liittyvine CRUD-operaatioineen

Komponentin kehitys ja sille CRUD-operaatioiden toiminnallisuuksien toteuttaminen kävi läpi useita vaiheita. Projektin alkuvaiheessa komponentin ulkoiset kehykset siirrettiin sisäiseen komponenttikirjastoomme. Sieltä komponentti implementoitiin osaksi projektia ja se sisälsi alkuun täysin kuvitteellisen muotoista dataa. Seuraavaksi komponenttikokonaisuudelle kehitettiin komponentti uuden laskelman luomiseen. Myös muut CRUD:iin liittyvät toiminnallisuudet sijaitsivat omissa komponenteissaan.

Sivulla 42 olevassa kuvassa 10 on kuvattu kuvitteellinen ohjelmakomponentti. Numero yksi on oma sinisellä sävyllä kuvattu haitarikomponenttinsa, joka sisältää tekstikentän, pudotusvalikon, checkboxin ja tallennapainikkeen. Numero kaksi eli ohjelmalistan lukeminen tapahtuu luonnin yllä olevassa haitarikomponentissa, joka on kuvattu valkealla taustavärillä. Haitarit avautuvat ja sulkeutuvat oikean yläkulman nuolipainikkeista. Ohjelmalistan päivittäminen, poistaminen ja kopiointi tapahtuvat ohjelmalistauksessa sijaitsevasta hampurilaisvalikosta valitsemalla. Hampurilaisvalikon valinnat avaavat uusia näkymiä, joita ei ole tarkemmin kuvattu tässä esimerkissä.

Projektin lopussa päädyin refaktoroimaan laskelmalistauskomponenttia ja erityisesti sen komponenttihierarkiaa ja tiedostorakennetta (React 2022f). Koska kaikki laskelmalistaan kehitetyt komponentit kuuluivat yksinomaisesti sille, jaoin sen komponentit loogisiin vanhempi- ja lapsikomponentteihin (kuva 11).

Kuva 11. Laskelmalistauskomponentin refaktoroitu tiedostorakenne

Keskustelimme kehittäjäpalaverissa tällaisen rakenteen hyvistä ja huonoista puolista. Hyvä puoli oli, että se lisäsi komponenttien luettavuutta. Aiemmin kaikki liittyvät komponentit kirjoitettiin samalle tasolle. Huono puoli puolestaan, jos komponentteja tarvittaisiin muualla ohjelmistossa, sisäkkäinen tiedostorakenne olisi epälooginen. Reactin dokumentaatio suosittelee välttämään liiallista sisentämistä tiedostorakenteessa (React 2022h).

Jokaiseen nimettyyn komponenttiin kuuluu TS:llä kirjoitettu komponenttitiedosto ja yksikkötestitiedosto. Tämä paljastui ongelmalliseksi yksikkötestejä ylemmällä tasolla kohdentaessa. LaskelmatKomponenttia testatessa kaikki sen alla olevat komponentit tulevat testatuiksi. Tämä ei ollut tavoiteltavaa, kun halusin testata yksittäistä komponenttia. Parempi olisi luoda Laskelmat-kansio ja sijoittaa varsinainen LaskelmatKomponentti samalle tasolle Luo-, Poista- ja PäivitäLaskelmaKomponenttien kanssa.

5 Pohdinta

Ohjelmistoprojekti ja sen mukana päiväkirjaraportointi päättyivät suunnitellun aikataulun mukaisesti. Ohjelmistokehitys on ammattitaitona haastavaa, monimutkaista ja aikaa vievää puuhaa. Ohjelmistoprojektin puoli vuotta kestänyt aikahaitari ja raportoidut kahdeksan seurantaviikkoa sen varrelta muodostivat varsin soveliaan opinnäytetyön laajuuden tutkimusongelman käsittelylle. Ohjelmistokehittäjänä kehittymisessä kestää aikaa. (Disney Codeillusion 2020.)

Sekä ohjelmistoprojekti että siihen liittynyt päiväkirjaraportointi onnistuivat pääosin tavoitteissaan. Seuraavissa luvuissa käydään läpi yksityiskohtaisemmin ohjelmistoprojektin onnistuminen (5.1 Ohjelmistoprojektin post mortem), opinnäytetyön onnistuminen tekijän ammatilliselle kehittymiselle asetetuissa tavoitteissa (5.2 Henkilökohtainen kehitys) ja jatkopohdinta (5.3 Jatko ja tulevaisuuden näkymät).

5.1 Ohjelmistoprojektin post mortem

Ohjelmistoprojekti päättyi alkuperäisen aikataulun mukaisesti. Ohjelmistoprojektit asettavat monesti tavoitteita, jotka eivät lopulta kuitenkaan toteudu aikataulussa tai lainkaan (Savolainen 2011, 9-11; Massessi 2021). Ensimmäisen tärkeysasteen ominaisuuksista oli projektin päättyessä valmiina 24:stä 22 kappaletta. Kaksi valmistumatonta ensimmäisen tärkeysasteen ominaisuutta oli työn alla projektin päättyessä. Kokonaisuudessaan 44 määritellystä toiminnallisuusvaatimuksesta 31 oli projektin päätyttyä valmiina.

Projektin alkuvaiheessa tunnistettiin halutuissa ominaisuuksissa teknisiä ongelmia, jotka otettiin huomioon myöhemmässä jatkosuunnittelussa. Nk. “feature creep”, missä uusia toiminnallisuusvaatimuksia lisätään jatkuvasti meneillä olevaan projektiin maalitolppia siirtäen, vaikutti pysyvän hyvin hallinnassa halki projektin (Elliot 2007). Vaatimukset eivät muuttuneet kesken projektin kuin jo viimeistään projektin kickoff-vaiheessa haasteeksi tunnistetun Timeline-komponentin osalta. Savolaisen (2011, 41) tutkimus epäonnistuneista ohjelmistoprojekteista nostaa projektien käynnistysvaiheen kriittiseen asemaan projektien onnistumisen kannalta. Uuden moduulin vastaanotto oli positiivinen organisaation sisäisesti varsinaisen käyttäjäpalautteen vielä odottaessa.

Projektia varten PR:iä tehtiin sadottain eri repositoryihin. PR:ien koot kuitenkin vaihtelevat huomattavasti, toisten ollessa yhden rivin muutoksia ja toisten kymmenien tiedostojen ja satojen rivien pituisia. Oleellista ei ole PR:ien määrä eikä koko, vaan PR:ien ratkaisemat ongelmat. Voi pohtia, onko PR hyvä, jos se tuo ohjelmistoon paljon toiminnallisuutta mutta samalla myös paljon ongelmia?

Sisäisesti todettiin, että projekti on toteutettu projektisuunnitelman mukaisesti ajallaan. Projektiryhmä oli selvästi oikeansuuntaisesti mitoitettu kooltaan ja osaamiseltaan, ja se oli suunniteltu ja johdettu onnistuneesti. Savolaisen (2011, 32-34) määrittelemistä ohjelmistoprojektin onnistumisen kriteereistä, jotka ovat asiakastyytyväisyys sekä lyhyen ja pitkän aikavälin liiketoimintahyöty, ei ole vielä tietoa.

5.2 Henkilökohtainen kehitys

Henkilökohtaisen kehityksen alkuun anekdootti: eräs kokenut ohjelmistokehittäjä totesi, että harvat ohjelmistokehittäjistä kykenevät lukemaan koodia ja ymmärtämään jokaisen askeleen, jonka ohjelma suorittaa, suoraan päässään. Hänen mukaansa heitäkin on, mutta muiden on käytettävä ammattilaisohjelmia ja -menetelmiä. Tärkeimmiksi näistä hän nosti ammattilaistyökalujen kuten versionhallinnan, debuggauksen ja ohjelmointiympäristön hallinnan. Nämä olivat pitkälti tässä opinnäytetyössä toissijaisiksi ammatillisiksi kehitystavoitteiksi nostettuja teemoja.

Ensisijaiset tavoitteet opinnäytetyössä liittyivät käyttöliittymäkehitykseen. Opinnäytetyö kasvoi nopeasti ensisijaisia tavoitteita laajemmaksi kokonaisuudeksi. Reactia, TypeScriptiä, Jestiä ja Testing librarya, neljää ensisijaisissa tavoitteissa mainittua teknologiaa, on seurantajakson kaikissa viikkoanalyyseissa ensimmäistä ja seitsemättä lukuun ottamatta (taulukko 5). Ensimmäinen ja seitsemäs viikkoanalyysi ovat puolestaan tärkeimmät viikot toissijaisten tavoitteiden täyttymisen kannalta.

Opinnäytetyön ensisijainen ja toissijainen tavoite kietoutuvat yhteen seurantajakson aikana jokaisen raportointiviikon kohdalla jossakin määrin. Ensisijaista tavoitetta 1a käsitellään vähintään joka toisen seurantaviikon viikkoanalyyseissa. Tärkein seurantaviikko toissijaisten tavoitteiden täyttymisen kohdalta on ensimmäinen viikkoanalyysi. Viikko seitsemän on puolestaan tärkein opinnäytetyön tavoitteen 2b osalta. Voisi sanoa, että opinnäytetyö on kokonaisuutena vastaus opinnäytetyön tavoitteisiin, kehittymiseen käyttöliittymä- ja ohjelmistokehittäjänä. Molempiin saatiin opinnäytetyöntekijän mielestä kattava vastaus.

Taulukko 5. Asetetut tavoitteet ja niiden käsittely seurantajaksolla

Tavoitenro.	**Oman ammatillisen kehittymisen tavoitteet**	**Viikkoanalyysi**
1a	Kehittyminen käyttöliittymien ohjelmoimisessa Reactilla ja TypeScriptillä	2, 3, 5, 6, 8

1b	Kehittyminen yksikkötestien kirjoittamisessa Jestillä ja Testing Libraryllä	4, 8
2a	Kehittyminen ohjelmistokehityksen yleisten ammattilaisohjelmien ja -menetelmien, kuten versionhallinnan, ohjelmointiympäristön, debuggauksen ja ketterän kehityksen hallitsemisen saralla	1, 4, 5
2b	Oppiminen ja kehittyminen muissa ohjelmistokehitykseen liittyvissä teknologioissa ja toimintatavoissa	1, 5, 6, 7

Yllä olevasta taulukosta 5 käy hyvin ilmi, että ensisijaisen tavoitteen 1b eli yksikkötestaamisen käsittely jäi lopulta vähäiseen osaan opinnäytetyössä ja seurantajakson viikkoanalyyseissa. Ennen opinnäytetyön kirjoittamista opinnäytetyöntekijä jopa pohti opinnäytetyön aiheeksi TDD-filosofiaa ja yksikkötestaamista. Tästä huolimatta opinnäytetyöntekijä kokee, että suurimmat harppaukset henkilökohtaisessa kehityksessä tapahtuivat juuri yksikkötestien kirjoittamisen saralla. Yksikkötestaamisessa kehittymisen raportointi jäi yksikkötestauskurssien suorittamisen jalkoihin.

Viiden ensimmäisen seurantaviikon ja viimeisen kolmen seurantaviikon välillä olleen raportointikatkoksen aikana kasvua käyttöliittymäkehittäjänä on hankalampi arvioida kuin seurantajakson aikana. Raportointikatkoksen varrelle mahtui suvantovaiheita ja pidempi sairaslomajaksokin. Kasvua tuolla välin kuitenkin tapahtui ja se on nähtävissä projektin loppupuolen viikkoanalyysien teemoissa 6, 7 ja 8. Viikon 6 teema vaikutti tekijälle tärkeimmältä jo mainitun yksikkötestaamisen lisäksi. Raportointiviikot 7 ja 8 olivat puolestaan eräitä tekijälle inspiroivimpia vaiheita projektissa. Tämä ei näy lopulta niiden viikkoanalyyseissa samalla tavalla kuin viikon 1 kohdalla. Syitä tähän on varmasti monia, kuten alun innostus ja lopun kiire.

5.3 Jatko ja tulevaisuuden näkymät

Ohjelmistoalalla ei voi olla koskaan valmis ja sen teknologiat edistyvät nopeaa tahtia. On pysyttävä kiinni kiihtyvässä kehityksessä ja käytävä jatkuvasti ammattitaitoa parantavia kursseja. Opinnäytetyön yhteydessä aloitettu Helsingin Yliopiston “Full Stack open 2022”-kurssi mainitsee “JavaScript fatiguen”, JavaScript-väsymyksen osana tätä jatkuvan kehittymisen vaatimusta (Luukkainen 2022). Opinnäytetyön tekeminen auttoi tekijää osaltaan JS-väsymykseltä suojautumisessa, sillä opinnäytetyössä tuli käsiteltyä ja pohdittua käyttöliittymäkehitystä laajemmasta perspektiivistä kuin vain uusimpien JS-kirjastojen kautta (Maida 2017).

Nykyisissä töissä eniten sisäistettävää tuntuu olevan projektin päätöksen jälkeen ja uuden projektin alkaessa React-projektien konfiguroimisessa ja ohjelmistoarkkitehtuurin kokonaisvaltaisessa ymmärtämisessä. Myös API-kehittäminen .NET-ympäristössä jätti paljon kysymysmerkkejä. Lisäksi testaamisessa riittää valtavasti jatko-opiskeltavaa jo pelkästään yksikkötestaamisen saralla puhumattakaan muista testaamisen muodoista (Pittet s.a.).

Tulevaisuus

Selainpohjaisten käyttöliittymäkehittäjien, erityisesti React-kehittäjien, tulevaisuudennäkymät ohjelmistoalalla vaikuttavat erittäin valoisilta. React on tämän hetken käytetyin ja kysytyin FE-teknologia. (Bolt 2022; Statista 2022). Kehittäjien markkina-arvoon ja palkkaan vaikuttavat kehittäjän kokemus ja taidot sekä työpaikan koko ja maantieteellinen sijainti (Bolt 2022; Burak 2022). Toivottavasti tästä päiväkirjatyyppisestä opinnäytetyöstä on osaltaan apua alasta kiinnostuneelle alan vaatimusten ja toimintatapojen esittelyn muodossa.

Lähteet

A Few Minutes Of Code 2019. How to convert an array into an object in javascript. Luettavissa: https://afewminutesofcode.com/how-to-convert-an-array-into-an-object-in-javascript. Luettu: 23.1.2022.

Atlassian 2022a. Making a Pull Request. Luettavissa: https://www.atlassian.com/git/tutorials/making-a-pull-request. Luettu: 16.1.2022.

Atlassian 2022b. Git Feature Branch Workflow. Luettavissa: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow. Luettu: 16.1.2022.

Bister, T. 2019. Tietojenkäsittelyn opinnäytetyö: Viittoja ja karttoja tutkimisen ja kehittämisen teille. 2019. Jyväskylän Ammattikorkeakoulu. Jyväskylä.

Bolt, C. 2022. The Complete React Developer Salary Guide for 2022. Luettavissa: https://bluelight.co/blog/react-developer-salary-guide. Luettu: 22.7.2022.

Bose, T. 2021. Redux vs Context API: When to use them. Luettavissa: https://dev.to/ruppysuppy/redux-vs-context-api-when-to-use-them-4k3p. Luettu: 23.1.2022.

Burak, A. 2022. Global React Developer Salary and Hourly Rate Report. Luettavissa: https://relevant.software/blog/react-js-developer-salary-and-hourly-rate/. Luettu: 22.7.2022.

Cepe, C. 2019. Managing data with modern JavaScript: map, filter, reduce, find. Luettavissa: https://chcepe.medium.com/managing-data-with-modern-javascript-map-filter-reduce-find-6659a69c08d9. Luettu: 30.1.2022.

Chacon, S. & Straub, B. 2014. Pro Git. 2nd Edition. Apress. New York. E-kirja. Luettu: 16.1.2022.

Codeacademy 2022. Red, Green, Refactor. Luettavissa: https://www.codecademy.com/article/tdd-red-green-refactor. Luettu: 6.2.2022.

Copes, F. 2018. Write JavaScript loops using map, filter, reduce and find. How to perform common operations in JavaScript where you might use loops, using map(), filter(), reduce() and find(). Luettavissa: https://flaviocopes.com/javascript-loops-map-filter-reduce-find/. Luettu: 30.1.2022.

Disney Codeillusion 2020. How Long Does It Take To Become A Computer Programmer? Luettavissa: https://codeillusion.io/blogs/blog-collection/how-long-does-it-take-to-become-a-computer-programmer. Luettu: 22.7.2022.

DV, R. R. 2022. Objects in Java World VS JavaScript World. Luettavissa: https://rajaraodv.medium.com/is-class-in-es6-the-new-bad-part-6c4e6fe1ee65. Luettu: 16.7.2022.

Elliott, B. 2007. Anything is possible: Managing feature creep in an innovation rich environment. 2007 IEEE International Engineering Management Conference, s. 304-307.

Erikson, M. 2018. Redux - Not Dead Yet! Luettavissa: https://blog.isquaredsoftware.com/2018/03/redux-not-dead-yet/#the-new-context-api-can-replace-redux. Luettu: 23.1.2022.

GeeksforGeeks 2021. Difference between Fetch and Axios.js for making http requests. Luettavissa: https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests/. Luettu: 17.4.2022.

Grace, N. 2018. How to read your way to becoming a better developer. Luettavissa: https://www.freecodecamp.org/news/how-to-read-your-way-to-becoming-a-better-developer-b6432fa5bc0c/. Luettu: 5.6.2022.

Hora, A. 2021. Googling for Software Development: What Developers Search For and What They Find. 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), s. 317-328.

Hunt, A. & Thomas, D. 2019. The Pragmatic Programmer: your journey to mastery. 20th Anniversary Edition, 2nd Edition. Addison-Wesley Professional. Boston. E-kirja. Luettu: 23.1.2022.

Jaya, T. 2018. The Minimum JavaScript You Should Know When You Code React & Redux. Itsenäinen kustantaja. Yhdysvallat.

Jest 2022. Jest CLI Options. Luettavissa: https://jestjs.io/docs/cli. Luettu: 6.2.2022.

Luukkainen, M. 2022. JavaScript fatigue. Luettavissa: https://fullstackopen.com/en/part0/fundamentals_of_web_apps#java-script-fatigue. Luettu: 22.7.2022.

Maida, K. 2017. How to Manage JavaScript Fatigue. Luettavissa: https://auth0.com/blog/how-to-manage-javascript-fatigue/. Luettu: 22.7.2022.

Martin, R. C. 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson. London. E-kirja. Luettu: 6.2.2022.

Massessi, D. 2021. Why Software Development Teams Keep Missing Project Deadlines. Luettavissa: https://medium.com/geekculture/why-software-development-teams-keep-missing-project-deadlines-6e5d733da1e2. Luettu: 26.6.2022.

McMinn, K. 2015. How to write the perfect pull request. Luettavissa: https://github.blog/2015-01-21-how-to-write-the-perfect-pull-request/. Luettu: 16.7.2022.

MDN 2022a. Details of the object model. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model. Luettu: 16.7.2022.

MDN 2022b. Arrow function expressions. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions. Luettu: 30.1.2022.

Morcov, S. 2021. Managing Positive and Negative Complexity: Design and Validation of an IT Project Complexity Management Framework. Väitöskirja. Arenberg Doctoral School, Faculty of Engineering Science. Luettavissa: https://lirias.kuleuven.be/retrieve/637007. Luettu: 4.6.2022.

Mursu, S. 2016. REST-tietokantarajapinta mobiilisovellukselle ja web-sivustolle. AMK-opinnäytetyö. Oulun ammattikorkeakoulu, tietotekniikan koulutusohjelma. Luettavissa: https://www.theseus.fi/bitstream/handle/10024/110124/Mursu_Sakari.pdf. Luettu: 22.5.2022.

Myers, N. Starliper, M. Summers, S. L. & Wood, D. A. 2017. The Impact of Shadow IT Systems on Perceived Information Credibility and Managerial Decision Making. Accounting Horizons, 31, 3, s. 105–123.

Nagey, S. 2019. React Hooks is the functional paradise you've been waiting for. Luettavissa: https://medium.com/capbase-engineering/react-hooks-is-the-functional-paradise-youve-been-waiting-for-994e53f65f94. Luettu: 16.7.2022.

Pavlutin, D. 2019. 6 Ways to Declare JavaScript Functions. Luettavissa: https://dmitripavlutin.com/6-ways-to-declare-javascript-functions/. Luettu: 30.1.2022.

Piispanen, E. 2021. Komponenttikirjaston rakentaminen Storybookin avulla. AMK-opinnäytetyö. Haaga-Helia ammattikorkeakoulu, tietojenkäsittelyn koulutusohjelma. Luettavissa: https://www.theseus.fi/bitstream/handle/10024/505277/Piispanen_Elina.pdf. Luettu: 23.1.2022.

Pittet, S. s.a. The different types of software testing. Luettavissa: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. Luettu: 22.7.2022.

Pountney, M. 2021. Pull Request Etiquette. Luettavissa: https://gist.github.com/mike-pea/863f63d6e37281e329f8#pull-request-etiquette. Luettu: 5.6.2022.

Ramirez, C. 2021. Typescript vs Flow. Which static type checker should you use on your project? Luettavissa: https://medium.com/geekculture/typescript-vs-flow-b4eb3778cf6b. Luettu: 20.2.2022.

React 2022a. Function and Class Components. Luettavissa: https://reactjs.org/docs/components-and-props.html#function-and-class-components Luettu: 23.1.2022.

React 2022b. Introducing Hooks. Luettavissa: https://reactjs.org/docs/hooks-intro.html Luettu: 23.1.2022. Luettu: 23.1.2022.

React 2022c. Adoption Strategy. Luettavissa: https://reactjs.org/docs/hooks-faq.html#adoption-strategy. Luettu: 23.1.2022.

React 2022d. Mocking a rendering surface. Luettavissa: https://reactjs.org/docs/testing-environments.html#mocking-a-rendering-surface. Luettu: 6.2.2022.

React 2022e. Converting a Function to a Class. Luettavissa: https://reactjs.org/docs/state-and-lifecycle.html#converting-a-function-to-a-class. Luettu: 13.2.2022.

React 2022f. Thinking in React, Step 1: Break the UI Into A Component Hierarchy. Luettavissa: https://reactjs.org/docs/thinking-in-react.html#step-1-break-the-ui-into-a-component-hierarchy. Luettu: 13.2.2022.

React 2022g. What is the difference between state and props? Luettavissa: https://reactjs.org/docs/faq-state.html#what-is-the-difference-between-state-and-props. Luettu: 17.4.2022.

React 2022h. File Structure. Avoid too much nesting. Luettavissa: https://reactjs.org/docs/faq-structure.html#avoid-too-much-nesting. Luettu: 22.5.2022.

Räisänen, T. & Virkkala, R. 2020. Päiväkirjamuotoinen opinnäytetyö tietojenkäsittelyn tutkinto-ohjelmassa. Luettavissa: https://blogi.oamk.fi/2020/04/17/paivakirjamuotoinen-opinnaytetyo-tietojenkasittelyn-tutkinto-ohjelmassa/. Luettu: 16.7.2022.

Schwaber, K. & Sutherland, J. 2020. Scrum-opas: Scrumin määritelmä ja pelisäännöt. Luettavissa: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Finnish.pdf. Luettu: 16.7.2022.

Savolainen, P. 2011. Why Do Software Development Projects Fail? Emphasing the Supplier’s Perspective and the Project Start-Up. Väitöskirja. Jyväskylän yliopisto, tietojärjestelmätiede. Luettavissa: https://jyx.jyu.fi/bitstream/handle/123456789/36970/9789513944681.pdf. Luettu: 22.7.2022.

Stack Overflow 2022. Methods in ES6 objects: using arrow functions. Luettavissa: https://stackoverflow.com/questions/31095710/methods-in-es6-objects-using-arrow-functions. Luettu: 30.1.2022.

Stoplight 2022a. Prism is a set of packages for API mocking and contract testing with OpenAPI v2 (formerly known as Swagger) and OpenAPI v3.x. Luettavissa: https://meta.stoplight.io/docs/prism/674b27b261c3c-overview. Luettu: 17.4.2022.

Stoplight 2022b. Serving Multiple OpenAPI Documents. Luettavissa: https://meta.stoplight.io/docs/prism/4bf78efbf9dd2-serving-multiple-open-api-documents.

Suvisuo, J. 2020. Google Analytics -rajapinnan integrointi Web-sovellukseen. AMK-opinnäytetyö. Hämeen ammattikorkeakoulu, tietojenkäsittelyn koulutusohjelma. Luettavissa: https://www.theseus.fi/handle/10024/345064. Luettu: 23.1.2022.

Swagger 2022. Swagger Editor. Luettavissa: https://editor.swagger.io/. Luettu: 17.4.2022.

Testing Library 2022. Debugging, screen.debug(). Luettavissa: https://testing-library.com/docs/queries/about#screendebug. Luettu: 6.2.2022.

TypeScript 2022a. TypeScript for JavaScript Programmers. Luettavissa: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. Luettu: 30.1.2022.

TypeScript 2022b. The Primitives: string, number, and boolean. Luettavissa: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#the-primitives-string-number-and-boolean. Luettu: 30.1.2022.

TypeScript 2022c. Differences Between Type Aliases and Interfaces. Luettavissa: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#differences-between-type-aliases-and-interfaces. Luettu: 30.1.2022.

Vuohijoki, P. 2018. REST-rajapinnan suunnittelu ja toteutus. AMK-opinnäytetyö. Turun ammattikorkeakoulu, tietotekniikka. Luettavissa: https://www.theseus.fi/bitstream/handle/10024/148038/Vuohijoki_Pasi.pdf. Luettu: 16.7.2022.

W3Schools 2022. JavaScript Arrow Function. Luettavissa: https://www.w3schools.com/js/js_arrow_function.asp. Luettu: 30.1.2022.