LAB University of Applied Sciences

Opintojenseurantatyökalun järjestelmäpäivitys

Moderni web-ohjelmointi ja käyttäjäkeskeinen suunnittelu

LAB-ammattikorkeakoulu
Tradenomi (AMK), tietojenkäsittely
2022
Iida Laukkanen, Aki Tiainen

Tiivistelmä

Tekijät	Julkaisun laji	Valmistumisaika
Iida Laukkanen	Opinnäytetyö, AMK	2022
Aki Tiainen	Sivumäärä	
	49	

Työn nimi

Opintojenseurantatyökalun järjestelmäpäivitys

Moderni web-ohjelmointi ja käyttäjäkeskeinen suunnittelu

Tutkinto ja koulutusala

Tradenomi (AMK), tietojenkäsittely

Toimeksiantaja

Saimaan tiimiyrittäjäakatemia ry

Tiivistelmä

Opinnäytetyön tarkoitus oli tuottaa järjestelmäpäivitys kehityksessä olevaan web-sovellukseen, joka on opintojenseurantatyökalu LAB-ammattikorkeakoulun markkinointiopiskelijoille ja -valmentajille. Heillä ilmeni tarve modernille ja helppokäyttöiselle opintojenseurantatyökalulle tehostamaan osuuskuntamuotoista opiskelua. Kehityksessä oleva työkalu tarvitsi päivityksen, jotta se olisi valmis käyttöönotettavaksi. Päivitys piti sisällään valmentajalle oman käyttöliittymän.

Opinnäytetyön teoriaosassa selvitettiin, mitä teknologioita modernin web-sovelluksen kehitykseen kannattaa valita, ja mitä käyttöliittymän suunnittelussa tulee ottaa huomioon. Opinnäytetyön toiminnallinen osa toteutettiin sekä käyttäjäystävällisen suunnittelun että modernin web-ohjelmoinnin periaatteita noudattaen. Toiminnallisen osan viimeistelemä sovellus toteutettiin yksisivuisen sovelluksen (SPA), REST API -rajapinnan ja Microsoft SQL -relaatiotietokannan muodostamana kokonaisuutena. Sovellusta kehitettäessä korkeimpana prioriteettina oli käyttäjien tarpeet ja toiveet.

Päivitetty opintojenseurantatyökalu täytti loppukäyttäjien tarpeet niin hyvin, että markkinointiosuuskunta Vola päätti ottaa sovelluksen pidempään testikäyttöön. Sen aikana opiskelijat tekevät järjestelmään oikeita opintojaan vastaavia kirjauksia ajankohtaisesti, ja valmentaja seuraa opiskelijoidensa edistymistä. Loppukäyttäjien tyytyväisyydestä voidaan päätellä, että käyttäjäkeskeisessä suunnittelussa on onnistuttu, ja että sovellus vastaa loppukäyttäjien tarpeeseen modernista ja helppokäyttöisestä opintojenseurantatyökalusta. Käyttäjäkeskeinen suunnittelu on tehokas väline käyttäjäystävällisten ohjelmistojen ja positiivisten käyttäjäkokemuksien luomisessa.

Asiasanat

ohjelmistokehitys, web-ohjelmointi, SPA, REST, käyttöliittymä, käyttäjäkokemus, käyttäjäkeskeinen suunnittelu

Abstract

Authors	Type of Publication	Published
Iida Laukkanen	Thesis, UAS	2022
Aki Tiainen	Number of Pages	
	49	

Title of Publication

System Update to Study Tracking Application

Modern Web Development and User Centered Designs

Degree and field of study

Bachelor of Business Administration (UAS), Business Information Technology

Client Organisation

Saimaan Tiimiyrittäjäakatemia ry

Abstract

The purpose of the thesis was to design and implement a system update for a web application. The application is designed to support marketing students and teachers of LAB University of Applied Sciences to track their studies in a modern and easy way. The application needed an update before it could be deployed for the end users. The update covered teacher's user interface. The theoretical part of the thesis aimed to answer the following questions: Which technologies should be chosen when creating a modern web application and which aspects should be taken in consideration when designing a user-friendly user interface?

User-centered design was the core principle in the designing process of the system update. Using the principles of modern web development was the second aspect of the thesis. The result of the functional part of the thesis was an updated system consisting of the following stack of technologies: Single Page Application, REST API, and Microsoft SQL database. The design decisions during the development of the application were based on the needs and wishes of the end users.

From the result of the thesis, it can be concluded that user-centered design is an effective way to design and create user-friendly web applications. The result of the update was so successful that marketing team Vola decided to start using the application for real right after testing it. The positive reception from the end users implies that the updated application serves its purpose and that the need for modern and easy study tracking application was met.

Keywords

software development, web development, SPA, REST, user interface, user experience, user-centered design

Sisällys

Käsitteet ja lyhenteet

API	*Application Programming Interface*: ohjelmointirajapinta, jonka avulla ohjelmat voivat keskustella ja jakaa tietoa keskenään.
Backend	Järjestelmän palvelinpuoli.
CSS	*Cascading Style Sheet*: HTML-sivuston tyyliohjeet.
Frontend	Järjestelmän selainpuoli.
HTML	*Hypertext Markup Language*: hypertekstin merkintäkieli, jolla kuvataan hyperlinkkejä sisältävää tekstiä.
HOPS	Henkilökohtainen opintosuunnitelma: dokumentoitu suunnitelma siitä, mitä, miten, milloin ja missä järjestyksessä opiskelija suorittaa opintonsa (eLab 2022).
Ikonografia	Visuaalisten elementtien ja symbolien käyttöä tietyn viestin välittämiseksi käyttäjälle (Ritter & Winterbottom 2017).
JS	*JavaScript*, (web-)ohjelmointikieli, joka mahdollistaa dynaamisen toiminnallisuuden verkkosivulla.
Konstruktori	engl. *Constructor*: olio-ohjelmoinnissa luokan erityisfunktio, jonka vastuulla on luokan muuttujien alustaminen (Tutorialspoint)
Ohjelmistokehys	Ohjelmistorunko, jota täydentämällä rakennetaan uusia sovelluksia tai sovelluksen osia.
SQL	*Structured Query Language*, relaatiotietokantojen standardoitu kyselykieli.
TS	*TypeScript,* ohjelmointikieli, joka suunniteltiin paikkaamaan JavaScriptin puutteita isoja ohjelmistoja rakennettaessa.
Typografia	Tekstiin, kirjainten asetteluun, fonttiin ja väritykseen liittyvää suunnittelua ja sommittelua (Graafinen 2015).

1 Johdanto

Digitalisaation myötä LAB-ammattikorkeakoulun markkinointiosuuskunnilla ilmeni tarve modernille, nopealle ja helppokäyttöiselle järjestelmälle, jolla opiskelijat voivat kirjata ylös opintosuorituksiaan, ja valmentajat voivat tarkastaa ja arvostella niitä. Idea opintojenseurantatyökalusta lähti markkinointiopiskelijoilta, koska he kokivat käytössä olevan järjestelmän, eli Excel-taulukkoon perustuvan opintojenseurannan, hitaaksi ja vanhanaikaiseksi. Kyseisen Excel-taulukon ongelmana oli tiedonvälityksen monimutkaisuus: opiskelijoiden tekemät muutokset eivät päivittyneet reaaliaikaisesti valmentajille – ja toisinpäin – vaan se vaati Excel-tiedostojen lähettämistä manuaalisesti opiskelijalta valmentajille ja heiltä yhä eteenpäin. Excel-taulukon käyttämisessä ilmeni myös käytettävyysongelma sen ollessa opiskelijan tietokoneeseen sidonnainen – opiskelija ei aina kanna tietokonettaan mukana esimerkiksi projektitöissä.

Markkinointiosuuskuntien tarpeeseen lähdettiin vastaamaan kesällä 2019, jolloin perustettiin projekti HOPS2.0. Projektissa digitradenomiopiskelijoista koostuva ohjelmistokehitysryhmä lähti luomaan opintojenseurantatyökalun ensimmäistä versiota markkinointiopiskelijoiden toiveiden ja tarpeiden pohjalta. Matkan varrella markkinointiopiskelijat ovat päässeet testaamaan sovelluksen eri ominaisuuksia ja vaikuttamaan sovelluskehitykseen antamalla palautetta ja ideoita. Sovellukseen on kehitetty myös valmentajien puolen ensimmäinen, alkeellinen versio.

Sovelluksen nykytila ja samalla opinnäytetyön lähtötilanne pitää sisällään kahtia jaetun alustan, joka koostuu opiskelijan ja valmentajan puolista. Lisäksi sovelluksessa on kirjautumis- ja rekisteröitymissivut. Opiskelijan puoli sisältää opintosuoritusten ajanoton sekä kirjaamisen viiteen eri opintokategoriaan, kirjausten tarkastelun, edistymisen seurannan ja kirjausten viennin Exceliin. Sovelluksen tulostama Excel-tiedosto vastaa tiedoiltaan osuuskunnilla aiemmin käytössä ollutta Excel-tiedostoa. Valmentajan puoli sisältää listat osuuskunnan opiskelijoista ja kunkin opiskelijan kirjauksista sekä opiskelijan edistymisen seurannan ja kirjausten hyväksymisen.

Opintojenseurantatyökalu ei ole sellaisenaan valmis käyttöönotettavaksi osuuskunnissa. Valmentajien puolelta puuttuu vielä toiminnallisuuksia, ja se tarvitsee selkeän käyttöliittymän helpottamaan opintojenseurantaa. Opiskelijan puolelta puuttuu enää pieniä yksityiskohtia, ja se vaatii hieman virheenkorjausta.

Opinnäytetyön tavoitteena on suunnitella ja toteuttaa opintojenseurantatyökaluun päivitys noudattaen modernin web-sovelluskehityksen periaatteita. Päivityksen tulee olla käyttäjäystävällisesti suunniteltu, eli vastata kohderyhmänsä tarpeeseen huomioiden opiskelijoiden ja

valmentajien toiveet. Päivityksen tavoitteena on viimeistellä osuuskuntalaisten arkea tehostava sovellus, joka olisi helposti saavutettavissa ja reaaliaikaisesti käytettävissä. Sovelluksen tavoitteena on tehostaa osuuskuntalaisten arkea esimerkiksi vähentämällä opintojen edistymisen seurantaan käytettävää työmäärää ja nopeuttamalla tiedonvälitystä.

Opinnäytetyön toiminnallisen osan tavoitteena on lisätä valmentajan käyttöliittymään ne ominaisuudet, jotka vielä tarvitaan sovelluksen käyttöönottoa varten. Näihin ominaisuuksiin kuuluu esimerkiksi kirjauksen arvostelu, osuuskunnan luominen ja hallinta sekä opiskelijan tietojen vieminen Exceliin. Lisäksi projektissa voidaan kehittää osuuskunnan arkea tehostavia lisäominaisuuksia valmentajien ja opiskelijoiden toiveiden pohjalta.

Opinnäytetyö keskittyy sovelluksen päivityksen suunnitteluun ja toteutukseen. Päivityksen julkaiseminen rajataan ulos, vaikka sovelluksesta on jo julkaistu testiversio internetiin. Teoriaosuudessa käsitellään vain opintojenseurantatyökalun kehityksessä käytettäviä teknologioita, eikä kaikkia moderneja web-teknologioita. Käyttäjäkokemusta ja käyttäjäkeskeistä suunnittelua käsitellään digitaalisen tuotteen kehityksen näkökulmasta. Kehittäjän kokemus (engl. *Developer Experience*) sovelluksen kehityksestä ja kehitettävästä sovelluksesta rajataan opinnäytetyön ulkopuolelle.

Opinnäytetyön toimeksiantajina toimivat LAB-ammattikorkeakoulun markkinoinnin opetuksen kehittämisestä vastaavat lehtorit sekä opintojenseurantatyökalun tilaaja Saitemia eli Saimaan tiimiyrittäjäakatemia ry. Toimeksiantajana toimivat lehtorit toimivat Saitemian osuuskuntien opettajina eli valmentajina.

Saitemia on LAB-ammattikorkeakoulun Lappeenrannan kampuksella markkinointia opiskelevien tradenomien yhdistys, joka koostuu jokaisen vuosikurssin perustamista osuuskunnista. Osuuskunnat tarjoavat yrityksille markkinointiin liittyviä palveluita markkinoinnin eri osa-alueilta. Saitemia toimii osuuskuntiensa kattojärjestönä kehittäen niiden toimintaa sekä yhteistyötä eri alojen toimijoiden kanssa. Sen tarkoitus on myös edistää tiimiyrittäjyyden oppimismuotoa. (Saimaan tiimiyrittäjäakatemia ry 2022.)

Opinnäytetyössä tarkastellaan sovelluskehitystä keskittyen modernin web-ohjelmoinnin periaatteisiin sekä käyttäjäystävälliseen suunnitteluun. Työssä haetaan vastausta päätutkimuskysymykseen: Miten luodaan käyttäjäystävällinen ja moderni web-sovellus?

Päätutkimuskysymystä tarkentavat apukysymykset:

- Mitkä teknologiat sovelluskehitykseen kannattaa valita ja miksi?
- Mitä käyttöliittymän suunnittelussa tulee ottaa huomioon?
- Miten käyttäjäkokemusta voidaan parantaa jo kehitysvaiheessa?

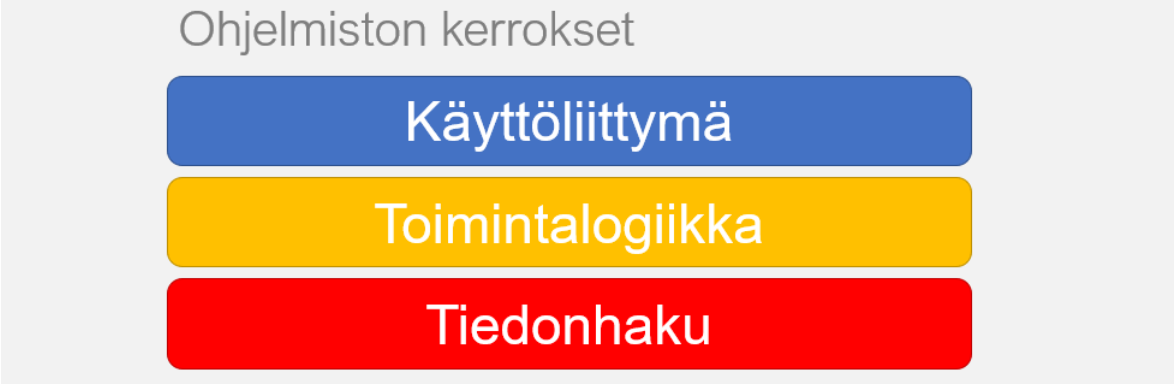
2 Modernin web-ohjelmoinnin periaatteita

2.1 Järjestelmän arkkitehtuuri

Tietokoneiden ohjelmoinnissa on tapahtunut hyvin vähän muutosta 50 vuodessa. Ohjelmointikielet ovat hieman kehittyneet, ja ohjelmistotyökaluihin on tullut huomattavia parannuksia, mutta kaikesta huolimatta ohjelmoinnin perusperiaatteet eivät ole muuttuneet. (Martin 2017.)

Ohjelmistojen arkkitehtuurin suunnittelun ja toteutuksen perustana pitäisi aina olla ylläpidon ja huollettavuuden mahdollistaminen. Yksi sovelluskehitystä ohjaavista toimintaperiaatteista on Separation of Concerns (SoC) eli huolenaiheiden erottaminen. SoC:n perusperiaatteen mukaan kehitettävä ohjelmisto pitäisi jakaa osiin vaadittavien ominaisuuksien perusteella. (Microsoft 2021a.)

Ohjelmiston arkkitehtuuri voidaan loogisesti rakentaa noudattamaan SoC:tä erottamalla liiketoimintalogiikka infrastruktuurista ja UI-logiikasta. Tällainen erottelu auttaa varmistamaan liiketoimintamallin helpon testattavuuden ja kehittymisen ilman, että se on tiukasti kytketty ohjelmiston toteutuksen yksityiskohtiin. SoC on avaintekijä kerroksittaisen arkkitehtuurin (Kuva 1) toteuttamisessa sovelluskehityksessä. (Microsoft 2021b.)



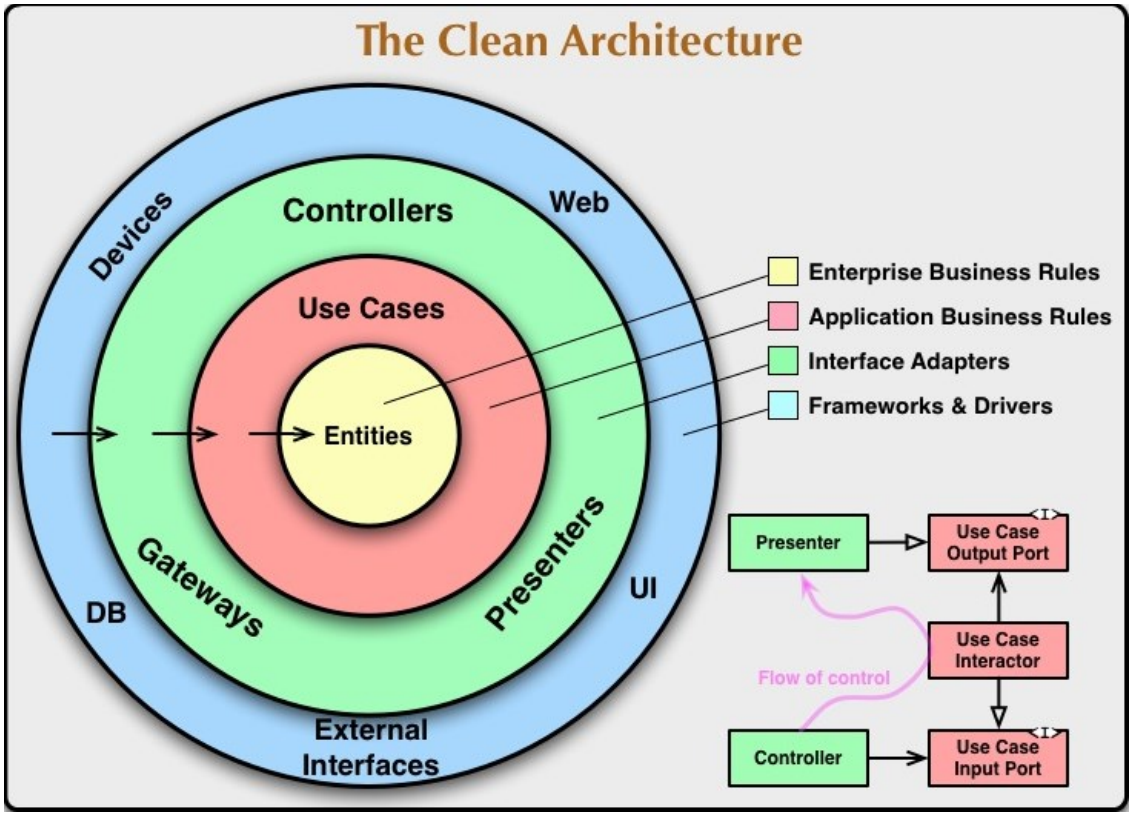
Kuva 1. Kerroksittainen arkkitehtuuri (mukailtu Microsoft 2021b)

Viime vuosina on nähty järjestelmäarkkitehtuurista lukuisia keskenään samankaltaisia ideoita – arkkitehtuurimalleja, jotka kuitenkin yksityiskohdiltaan eroavat toisistaan. Yhteinen tekijä näille arkkitehtuurimalleille on SoC, ja sen myötä ohjelmiston jakaminen kerroksiin. Tällaisia arkkitehtuurimalleja ovat mm. Alistair Cockburnin kehittelemä Heksagoninen arkkitehtuuri, Jeffrey Palermon sipuliarkkitehtuuri sekä DCI (*Data, Context and Interaction*), jonka ovat kehittäneet James Coplien ja Trygve Reenskaug. (Martin 2012.)

Edellä mainitut arkkitehtuurimallit tuottavat järjestelmiä seuraavanlaisilla ominaisuuksilla:

- Ohjelmistokehyksistä riippumaton: Arkkitehtuuri ei ole riippuvainen mistään ohjelmiston sisältävästä kirjastosta. Riippumattomuus mahdollistaa ohjelmistokehysten käyttämisen työkaluina sen sijaan, että järjestelmä pitäisi sulloa niiden rajoituksiin.
- Testattava: Liiketoimintasääntöjä voi testata ilman käyttöliittymää, tietokantaa, web-palvelinta tai muuta ulkopuolista elementtiä.
- Käyttöliittymästä riippumaton: Käyttöliittymää voi muuttaa vaivatta muuttamatta järjestelmän muita osia.
- Tietokannasta riippumaton: Tietokanta on vaihdettavissa, koska ohjelmiston liiketoimintasäännöt eivät ole sidonnaisia tietokantaan.
- Riippumaton kaikesta järjestelmän ulkopuolisesta toiminnasta. (Martin 2012.)

Kuvassa 2 esitetään puhdas arkkitehtuurimalli, joka pyrkii yhdistämään kaikki edellä mainitut arkkitehtuurit yhdeksi toimivaksi ideaksi. Luonnostaan testattavissa oleva järjestelmä luodaan jakamalla ohjelmisto kerroksiin ja noudattamalla riippuvuussääntöä. Kun jokin järjestelmän ulkoisista osista vanhenee, sen voi korvata mahdollisimman pienellä vaivalla. (Martin 2012.)



Kuva 2. Puhdas arkkitehtuurimalli (Martin 2012)

Puhtaan arkkitehtuurimallin noudattaman riippuvuussäännön mukaan yksikään ohjelmiston sisempi kehä ei voi olla tietoinen sitä ulommista kehistä. Mitään asiaa, joka on esitelty ulommalla kehällä, ei saa mainita koodissa sisemmällä kehällä. Näihin lukeutuvat funktiot, luokat, muuttujat ja muut ohjelmistossa nimetyt osat. Yhtä lailla sisempi kehä ei saa käyttää ulommalla kehällä esiteltyjä datamuotoja etenkään, jos ne ovat jonkun ulommalla kehällä käytetyn ohjelmistokehyksen generoimia. (Martin 2017.)

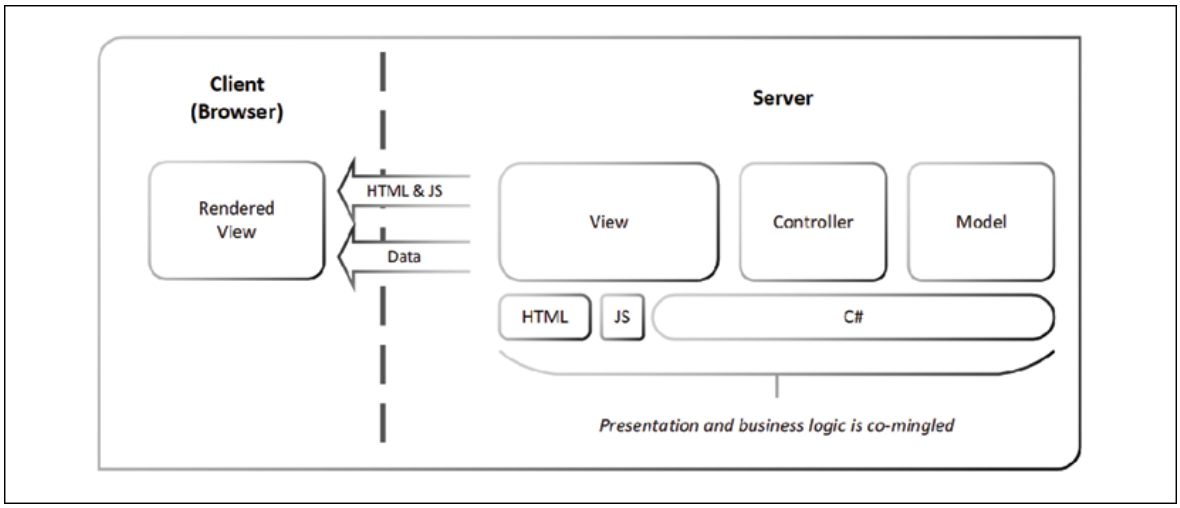
2.2 Teknologioita

Ohjelmistokehitykseen kehitetään jatkuvasti uusia teknologioita, joilla on keskenään erilaisia ominaisuuksia eri käyttötarkoituksiin. Projektiin sopivia teknologioita valittaessa täytyy huomioida useita seikkoja, jotta niiden ominaisuudet täyttävät kehitettävän ohjelmiston tarpeet mahdollisimman mutkattomasti. Siksi esimerkiksi koneoppimista varten rakennettava ohjelmisto kannattaa toteuttaa eri teknologioilla kuin web-sovellus. (Silk 2021).

Usein myös yksittäisen järjestelmän eri kerrokset, kuten frontend ja backend, toteutetaan eri teknologioita hyödyntäen. Esimerkiksi järjestelmän backend voi olla toteutettu C++-ohjelmointikielellä ja frontend JavaScriptillä (Kenzie Academy 2020). Stack Overflow:n (2022) tekemän tutkimuksen mukaan JavaScript on ollut suosituin ohjelmointikieli web-ohjelmoinnissa jo kymmenen vuotta (Ansari 2022).

2.2.1 Yksisivuinen sovellus

2000-luvun alussa web-applikaatioita kehitettiin noudattamaan edestakaista sovellusmallia. Tässä mallissa jokainen käyttäjän tekemä toiminto aloittaa selaimessa uuden kutsun palvelimelle, josta selain saa täysin uuden HTML-sivun näytettäväksi käyttäjälle – toisin sanoen jokainen toiminto suorittaa uuden sivun latauksen. Edestakaisella mallilla toteutetussa ohjelmistossa selain toimii ikään kuin HTML-sisällön luku- ja esityslaitteena, ja kaikki sovelluksen logiikka ja data sijaitsee palvelimessa. Kuviossa 1 esitetään edestakainen malli toteutettuna ASP.NET-ohjelmistokehyksellä. (Uluca 2018; Freeman 2020.)



Kuvio 1. Edestakainen malli (Uluca 2018)

Edelleen osa uusista web-sivustoista toteutetaan edestakaisella mallilla, koska ne vaativat hyvin vähän suorituskykyä ja toimintoja selaimelta. Tämän ansiosta näillä web-sivustoilla on kattavin mahdollinen selaintuki. Edestakaisessa mallissa on kuitenkin haittapuolensa etenkin käyttäjäkokemuksen näkökulmasta: koska jokainen käyttäjän tekemä toiminto aloittaa uuden sivulatauksen palvelimelta, on sivustoa hitaampi käyttää odotellessa sivulatauksia. Lisäksi palvelimelta vaaditaan enemmän tehoja ja suurempaa infrastruktuuria kaikkien kutsujen prosessoimiseen, ja käyttäjän internetyhteydeltä vaaditaan isompaa kaistanleveyttä, koska ladattavat HTML-dokumentit sisältävät usein keskenään samaa dataa. (Freeman 2020.)

Yksisivuinen sovellus eli SPA (*Single Page Application*) on moderni vaihtoehto perinteiselle edestakaiselle mallille. SPA:ssa selain tekee ensimmäisen kutsun palvelimelle ja saa vastaukseksi pohjimmaisen HTML-dokumentin ja mahdollisen JavaScript-koodin sen mukana. Kaikki seuraavat kutsut, jotka käyttäjän selaimesta lähetetään, ovat Ajax-kutsuja (*Asynchronous JavaScript And XML*) eli kutsuja, joilla selain voi kommunikoida palvelimen kanssa ilman sivulatauksia. Toisin sanoen kutsuissa pyydetään palvelimelta vain pieniä palasia dataa sen sijaan, että ladattaisiin kokonainen HTML-dokumentti jokaisella kutsulla. Datapalaset voidaan päivittää olemassa olevaan HTML:ään käyttämällä DOM-manipulaatiota (*Document Object Model*). (Freeman 2020.)

Web-applikaation rakentamiseen kannattaa valita SPA, kun ohjelmiston liiketoimintamalli vaatii paljon toiminnallisuuksia selaimessa ja käyttäjäkokemus on korkealla tärkeysjärjestyksessä. SPA:ssa on perinteistä web-sivustoa nopeammat sivunvaihdokset, ja käyttäjän yksittäiset toiminnot ovat responsiivisempia, koska täysiä sivunlatauksia tapahtuu harvoin tai ei ollenkaan. SPA:t tukevat ohjelmiston ominaisuuksia, kuten applikaation käyttämistä ilman nettiyhteyttä tai tietojen tallennusta väliaikaisesti ilman niiden lähettämistä

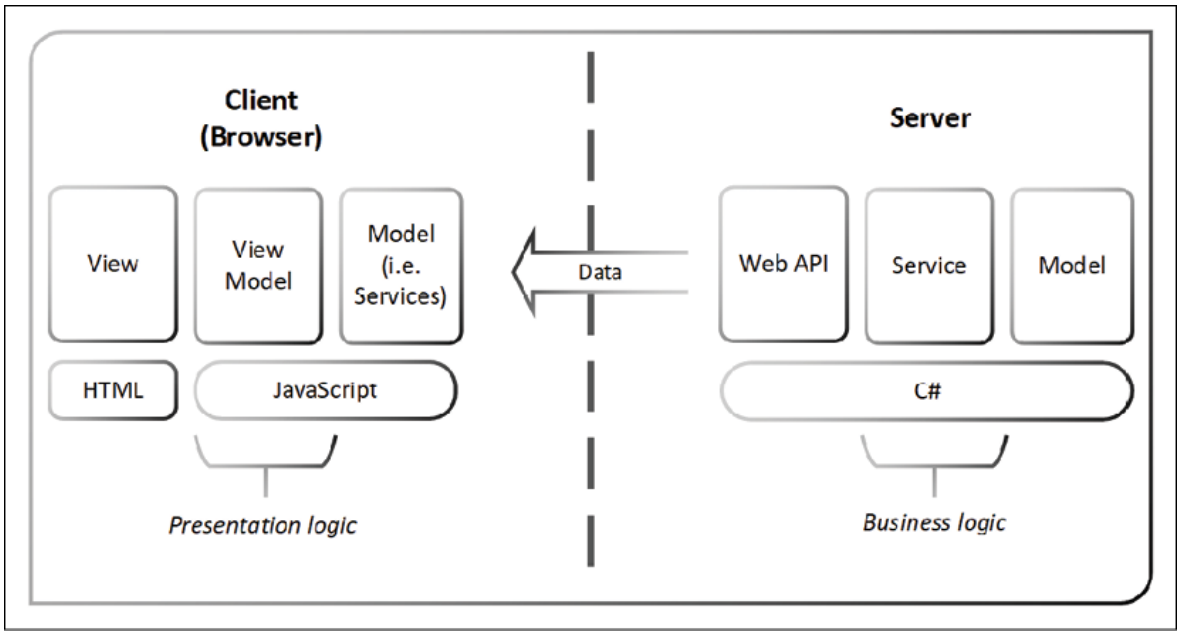
backendille. SPA:lla voi siis toteuttaa enemmän funktionaalisuuksia selaimessa, mutta tämä tekee herkästi sovelluksesta raskaamman pyörittää selaimessa. (Microsoft 2022.)

Angular

Angular on Googlen kehittämä avoimen lähdekoodin TypeScript-pohjainen ohjelmistokehys web-sovellusten kehittämiseen. Yhdessä Microsoftin kanssa Google kehitti TypeScriptistä Angularin oletuskielen alkuperäisen JavaScriptin tilalle. Angular on kehitetty etenkin tehokkaiden ja hienostuneiden SPA-sovellusten luomiseen. (Uluca 2018; Angular 2022a.) Angular on yksi vuoden 2022 trendikkäimmistä web-ohjelmoinnin teknologioista (Goel 2022).

Angularilla kehitetyille sovelluksille ominaisia piirteitä ovat helppo laajennettavuus yhtenäisten komponenttien ja kirjastojen avulla, ylläpidettävyys jatkuvien päivitysten myötä, kattava testattavuus sekä web-selainten kykenevyyteen perustuva standardoituneisuus (Freeman 2020). Angularista voi hyötyä niin yksittäisen sovelluskehittäjän projektit kuin yritystason järjestelmät sen laajan skaalautuvuuden vuoksi. Se on suunniteltu siten, että sovelluksen päivittäminen on mahdollisimman suoraviivaista. Angular-ekosysteemi koostuu monimuotoisesta yli 1.7 miljoonan sovelluskehittäjän, sovelluskirjaston tekijän ja sisällöntuottajan ryhmästä. (Angular 2022b.)

Toimintorikkaiden ja natiivisovellusten tuntuisten web-sovellusten kehittämisestä on tullut helpompaa ja kustannustehokasta Angularin ja muiden modernien web-ohjelmistokehysten yleistymisen myötä 2010-luvulla. Kuviossa 2 kuvataan Angularille tyypillinen asiakassovellus sekä palvelimelle toteutettu REST API -rajapinta. Kuvanmukainen erottelu asiakassovellukseen ja erilliseen API-rajapintaan vahvistaa SoC:n toteutumista. Teoriassa tällainen REST-arkkitehtuuria mukaileva järjestelmien toteutus mahdollistaa asiakassovellusten uusimisen ja muuttamisen ilman, että koko järjestelmää tarvitsee uusia. (Uluca 2018.)



Kuvio 2. Asiakassovellus ja REST API (Uluca 2018)

2.2.2 REST-arkkitehtuurityyli

Suurella osalla nykypäivän API-rajapinnoista on ongelma: kun ne ovat julkaistu, niitä on hyvin vaikea muuttaa. Tästä syystä lukuisat nimekkäätkin API-rajapinnat pysyvät muuttumattomina vuosia teknologioiden ja alan muuttuessa niiden ympärillä. *World Wide Web* koostuu miljoonista nettisivuista, jotka pyörivät tuhansilla servereiden eri toteutuksilla ja kokevat määräajoittaisia uudelleensuunnitteluja. REST-arkkitehtuuria mukailevat järjestelmät ovat suunniteltu kestämään näitä muutoksia. (Richardson ym. 2013.)

REST (*Representational State Transfer*) on hajautettujen hypermediajärjestelmien ja ohjelmointirajapintojen arkkitehtuurityyli, jonka Roy Fielding esitteli kuuluisassa väitöskirjassaan vuonna 2000 (REST API Tutorial 2022). Niin kuin kaikilla arkkitehtuurityyleillä, RESTillä myös on sille ominaiset rajat ja ohjaavat periaatteet. Näiden periaatteiden täytyy toteutua, jotta rajapintaa voidaan kutsua RESTfuliksi. (Doglio 2018.)

Doglion (2018) mukaan REST-arkkitehtuurin kuusi ohjaavaa perusperiaatetta:

- Asiakas – Palvelin (engl. *client – server*): Asiakas- ja palvelinkerrokset on eriytetty SoC:n mukaan toisistaan riippumattomiksi kokonaisuuksiksi, jotka ovat toistensa kanssa vuorovaikutuksessa.
- Tilaton (engl. *stateless*): Jokaisen asiakkaan tekemän pyynnön täytyy sisältää kaikki tarvittava informaatio niin, että palvelin pystyy suorittamaan pyynnön ilman aiemmin tallennettuja tietoja.

- Yhdenmukainen rajapinta: Pitämällä käyttöliittymä yhtenäisenä komponenttien välillä, asiakkaan helpompi olla vuorovaikutuksessa rajapinnan kanssa.
- Välimuistiin tallennettavissa (engl. *cacheable*): Asiakkaan pyyntöjä tai palvelimen vastauksia voi tallentaa selaimen tai palvelimen välimuistiin asiakkaan uudelleenkäytettäväksi.
- Kerroksittaisuus: Ohjelmiston jakaminen kerroksiin tehtävien mukaan helpottaa ohjelmiston ylläpitoa ja auttaa toteuttamaan SoC:ta.
- Koodi pyynnöstä (engl. *Code on Demand*): Asiakas voi ladata ja suorittaa palvelimen toimittaman koodin omassa selaimessaan. (Tämä periaate on ainut valinnainen RESTin kuudesta periaatteesta.)

2.2.3 Relaatiotietokannat ja SQL

Vuonna 1970 E. F. Codd julkaisi tutkimuksen *A Relational Model of Data for Large Shared Data Banks*, jossa hän ehdotti datan esittämistä taulukkomuotoisesti. Coddin ehdotuksessa ylimääräistä dataa käytetään linkittämään taulukkoihin tallennettu data toisiinsa sen sijaan, että käytettäisiin osoittimia navigointiin entiteettien välillä. Tätä ajatusta hän kutsui relaatiomalliksi. (Beaulieu 2020.)

Relaatiomallin lisäksi Codd ehdotti DSL/Alpha-nimistä kieltä relaatiomallin datan manipuloimiseksi. IBM (*International Business Machines Corporation*) kehitti Coddin ehdotuksen pohjalta kielen, josta muovautui ajan saatossa SQL. Yli 40-vuotias SQL on muuttunut historiansa aikana, ja sitä varten on luotu standardeja vuodesta 1980 lähtien. Uusimpien standardien myötä SQL-kieleen on lisätty muun muassa tuki olio-ohjelmoinnin funktionaalisuuksille, minkä takia se on edelleen pysynyt relevanttina. (Beaulieu 2020.)

Koska SQL on standardoitu kieli, se on yleistynyt relaatiotietokantojen käytetyimpänä ohjelmointikielenä. Yleisyytensä ja standardoituneisuutensa takia ohjelmistokehittäjien kannattaa hallita SQL ja relaatiotietokannat tänäkin päivänä. (Zapanta 2022.)

2.3 Versionhallintatyökalut

Ohjelmistokehitysympäristöjen kehittyessä niiden rinnalle on kehitetty versionhallinnan järjestelmiä eli työkaluja, jotka auttavat ohjelmistokehittäjiä hallitsemaan lähdekoodissa tapahtuvia muutoksia. Versionhallinnan työkalut ovat erityisen tärkeitä ohjelmistokehitysryhmille, joissa moni henkilö saattaa tehdä ohjelmistokoodiin muutoksia samanaikaisesti. Versionhallintajärjestelmät auttavat ohjelmistokehitysryhmiä työskentelemään nopeammin ja älykkäämmin. (Atlassian a.)

Versionhallintaohjelmisto pitää kirjaa kaikista lähdekoodissa tapahtuvista muutoksista eräänlaisessa tietokannassa. Versionhallinnan avulla ohjelmoijat voivat verrata aikaisempia versioita ohjelmistosta keskenään ja palauttaa aiemman version ohjelmistosta, jos uusimmissa muutoksissa on mennyt jotain pieleen. (Atlassian a.)

Git on aktiivisesti ylläpidetty avoimen lähdekoodin projekti, jonka perusti Linus Torvalds vuonna 2005. Git on yleisesti eniten käytetty moderni versionhallintaohjelmisto maailmassa. Gitin vahvuus on sen hajautettu arkkitehtuuri. Hajautetussa versionhallinnan järjestelmässä täysi versionhallinnan historia on jokaisen projektiin osallistuvan kehittäjän omalla laitteella yhden keskitetyn palvelimen sijaan. Tämä tekee Gitistä tehokkaan, turvallisen ja joustavan versionhallinnan järjestelmän. (Atlassian b.)

2.4 Projektinhallintatyökalut

Ohjelmistoprojektinhallinta on ohjelmistoprojektien suunnittelua, toteutusta, valvontaa ja ohjausta. Sen tavoitteena on täyttää vaatimukset tietokoneohjelmistojen kehittämiseksi hallitsemalla, kohdistamalla ja aikatauluttamalla resursseja, jotta projektille asetetut tavoitteet saavutetaan kriteerien mukaisesti. (JavaTpoint; Association for Project Management.)

Investoimalla tehokkaaseen projektinhallintaan varmistetaan resurssien tehokas ja arvoa tuottavin käyttö, suurempi todennäköisyys saavuttaa haluttu tulos sekä projektin sidosryhmien tarpeiden tyydyttäminen. Projektinhallinnan ydinkomponentteja ovat muun muassa projektin resurssien ja aikataulujen arvioiminen, vaatimusmäärittely, lopputuotteen laadun määrittely, projektin edistymisen seuranta sekä projektitiimin johtaminen ja motivointi. (Associaton for Project Management.)

Digitaaliset projektinhallintatyökalut on suunniteltu auttamaan monien ihmisten välisessä yhteistyössä. Niiden avulla kaikki projektiin liittyvä tieto löytyy yhdestä paikasta, tehtävät ovat järjestyksessä, työn raportoiminen sekä aikataulun ja budjetin seuranta on helppoa ja resurssit voi kohdistaa järkevästi. Projektinhallintatyökalut voidaan karkeasti jakaa kolmeen kategoriaan: tehtävienhallintaan ja viestintään keskittyvät työkalut, kokonaisvaltaisemmat työkalut ja projektiportfolioiden hallintatyökalut. (Pulkkanen.)

3 Moderni käyttäjäkokemus ja käyttäjäkeskeinen suunnittelu

3.1 Käyttöliittymä

Käyttöliittymäksi eli UI:ksi (*User Interface*) kutsutaan käyttäjän ja tuotteen välistä keskustelua, jonka tarkoituksena on suorittaa tehtävät, joilla käyttäjä saavuttaa tavoitteensa. Käyttöliittymä on se, mitä käyttäjä näkee ja koskettaa käyttäessään tuotetta. Se yhdistää käyttäjän tuotteen perustana olevaan teknologiaan. (McKay 2013.)

Käyttöliittymiä on joka puolella. Esimerkiksi auton käyttöliittymä sisältää ohjauspyörän, polkimet, kojelaudan, avaimet, penkit ja sisustuksen. Jokainen käyttöliittymän elementti voidaan arvioida sen mukaan, kuinka tehokkaasti se kommunikoi käyttäjälle. (McKay 2013.)

McKayn (2013) mukaan hyvin suunniteltu käyttöliittymä kiteytyy käyttäjälle kommunikointiin luonnollisella, asiantuntevalla, ystävällisellä, helposti ymmärrettävällä ja tehokkaalla tavalla. Käyttöliittymän suunnittelu ei ole vain subjektiivista visuaalista taidetta, vaan periaatteellinen objektiivinen kommunikaatiotaito selittää tehtäviä käyttäjälle.

3.2 Käyttäjäkokemus

Käyttäjäkokemus eli UX (*User Experience*) on yksi tärkeimmistä tekijöistä digitaalisen tuotteen luomisessa. Termin keksijän, Donald Normanin, mukaan se kattaa kaikki näkökulmat henkilön kokemuksesta tuotteen käytöstä: teollisen suunnittelun, grafiikat, käyttöliittymän, fyysisen vuorovaikutuksen sekä käsikirjan. (Ritter & Winterbottom 2017.)

Käyttäjäkokemus ei koske ainoastaan digitaalisia tuotteita, vaan se näkyy myös yksinkertaisissa arkipäivän toimissa, kuten ruokakaupassa navigoidessa tai julkisessa WC:ssä asioidessa. Tällaisissakin tilanteissa palvelun käyttäjä voi kohdata haasteita, joiden seurauksena syntyy huono käyttäjäkokemus. Jokainen käyttäjäkokemus voidaan suunnitella uudestaan positiiviseksi, mikä voi vaikuttaa merkittävästi liiketoimintaan etenkin verkkoyrityksistä puhuttaessa. Ymmärtämällä liiketoiminnan ja käyttäjän tarpeet voidaan keskittyä siihen, mitä tarkalleen pitäisi korjata. (Maioli 2018.)

Käyttäjäkokemukseen vaikuttaa käyttöliittymän lisäksi tuotteen sisäiset osat, joiden kanssa käyttäjä ei ole suoraan vuorovaikutuksessa, sekä sen ulkoiset osat. Esimerkiksi auton käyttäjäkokemukseen vaikuttavia sisäisiä osia ovat moottori, kori ja toimintavarmuus, ja ulkoisia auton osto, toimitus ja takuu. Digitaalisessa tuotteessa vastaavia ominaisuuksia ovat järjestelmän tietokanta (sisäinen) ja järjestelmän tekninen tuki (ulkoinen). (McKay 2013.)

Käytettävyys on yksi miellyttävän käyttäjäkokemuksen avaintekijöistä, ja sillä on keskeinen merkitys markkinamenestyksen saavuttamiseksi. Useimmissa organisaatioissa tehdään

virhe miellyttävän käyttäjäkokemuksen suunnittelussa, kun työntekijät viittaavat henkilökohtaiseen kokemukseensa tai havainnointiinsa, ja tekevät sitä kautta oletuksia loppukäyttäjän tarpeista ja mieltymyksistä koskien tuotteen ominaisuuksia. Tätä näkökulmaa ei voida pitää luotettavana, koska näiden oletusten esittäjä ei välttämättä edusta kohderyhmää, jolloin hänen tarpeensa eivät vastaa loppukäyttäjän tarpeita. Ominaisuuden todellinen tarve selviää ainoastaan loppukäyttäjän testattua sitä. (Ritter & Winterbottom 2017.)

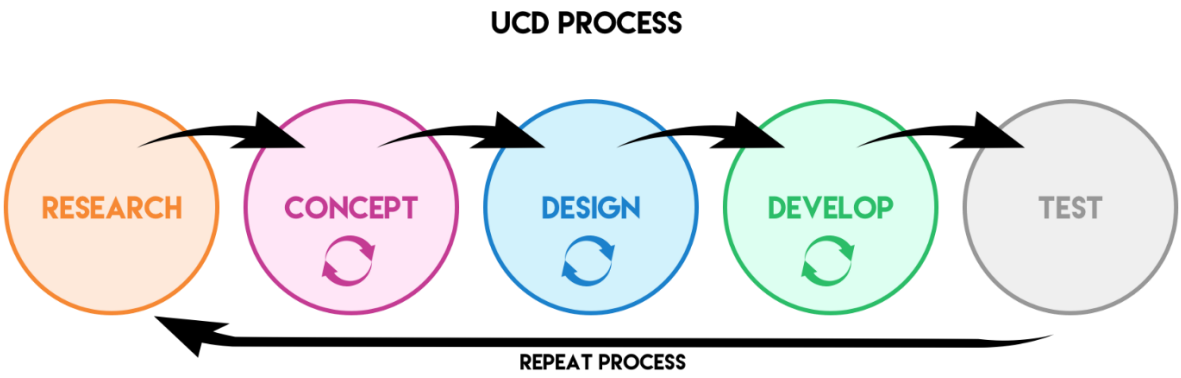
3.3 Käyttäjäkeskeinen suunnittelu

Sovelluksen suunnittelussa keskitytään usein organisaation liiketoimintatavoitteisiin, hienoihin toiminnallisuuksiin ja laitteiden tai ohjelmistotyökalujen teknisiin ominaisuuksiin, mutta prosessin tärkein osa, eli loppukäyttäjä, jää huomiotta (Usability First 2015). Sovelluksen suunnittelussa korkeimpana prioriteettina tulisi aina olla käyttäjän tarpeisiin vastaaminen (Ustwo 2014, 5).

Käyttäjäkeskeinen suunnittelu eli UCD (*User-Centered Design*) on suunnittelufilosofia, jonka periaatteet sovelluskehityksessä keskittyvät käyttäjän haluihin, tarpeisiin ja rajoitteisiin parhaan mahdollisen lopputuotteen luomiseksi käyttäjälle. Dreyfuss kertoi jo vuonna 1955 tekstissään *Designing for People*, että suunnittelussa on onnistuttu, jos tuote tuo käyttäjälleen turvallisuutta, mukavuutta, ostohalukkuutta, tehokkuutta tai iloa. (Ritter & Winterbottom 2017.) Käyttäjäkeskeinen suunnittelu vaikuttaa positiivisesti käyttäjäkokemukseen mahdollistaen samalla myynnin ja asiakasuskollisuuden kasvun (Usability First 2015).

3.3.1 Käyttäjäkeskeisen suunnittelun vaiheet

Käyttäjäkeskeinen suunnittelu on laaja-alainen ja muovautuva prosessi, mutta se noudattaa pohjimmiltaan aina samaa kaavaa (Kuvio 3): tutkimus, konsepti, suunnittelu, kehittäminen ja testaus. Jokaisen syklin jälkeen tehdään iteraatio käyttäjän palautteen perusteella, ja jokaisen syklin myötä tuote on yhä enemmän oikeilla jäljillä. (Ritter & Winterbottom 2017.)



Kuvio 3. Käyttäjäkeskeisen suunnittelun vaiheet (Ritter & Winterbottom 2017)

Käyttäjäkeskeinen suunnittelu alkaa käyttäjien ja kilpailijoiden laajalla tutkimuksella, jolla saadaan kokonaisvaltaista ymmärrystä käyttäjän tarpeista. Konseptivaiheessa määritellään projektin laajuus ja vaatimukset tutkimuksessa kerätyn datan pohjalta, listataan käytettävyystavoitteet ja tehdään paperiprototyyppejä käyttäjäskenaarioista. Suunnitteluvaiheessa täsmennetään teknisiä ja toiminnallisia vaatimuksia yksityiskohtaisemmiksi luomalla tehtävävirtaa, käyttäjäpolkuja, rautalankamalleja ja prototyyppejä sekä visuaalisia suunnitelmia. Kehittämisvaiheessa suunnitelmat viedään toteutukseen keskittyen parhaisiin käytäntöihin ja saavutettavuusstandardeihin. Toteutuksen jälkeen lopputuotetta testataan muun muassa kohderyhmillä, kenttätutkimuksilla ja suorituskykyanalyyseillä. Näillä käytettävyystesteillä selvitetään mahdolliset käytettävyysongelmat, joita lähdetään korjaamaan seuraavan syklin aikana. (Ritter & Winterbottom 2017.)

3.3.2 Suunnitteluvalinnat

Suunnitteluvaiheessa päätösten tulee perustua siihen, että lopputuotetta käyttäessä käyttäjä voi suorittaa haluamansa toiminnot mahdollisimman nopeasti. On tärkeää ottaa huomioon ulkoasun sopivuus käyttäjäryhmälle; esimerkiksi nuorison silmään sopii trendikkäät ratkaisut, kun taas vanhempi väestö arvostaa tuttua ja turvallista. (Ustwo 2014, 5.) Siksi on tärkeää tehdä tutkimusta kohderyhmästä ja lopulta testata, käyttäytyykö tuote käyttäjän toiveiden ja tarpeiden mukaisesti.

Tutkimus- ja konseptivaiheet ovat oleellisia tuotteen perustan luomisessa, mutta käyttöliittymän visuaalinen suunnittelu herättää konseptin henkiin. Nettisivusto voi olla kaunis, mutta sen täytyy olla myös helppokäyttöinen. Käyttämällä alan standardinmukaisia käytäntöjä, kuten Googlen Material Design tai iOS:n Human Interface Guidelines, käyttöliittymä ei ole pelkästään esteettisesti miellyttävä, vaan se täyttää myös kaikki käytettävyysvaatimukset. (Ritter & Winterbottom 2017.)

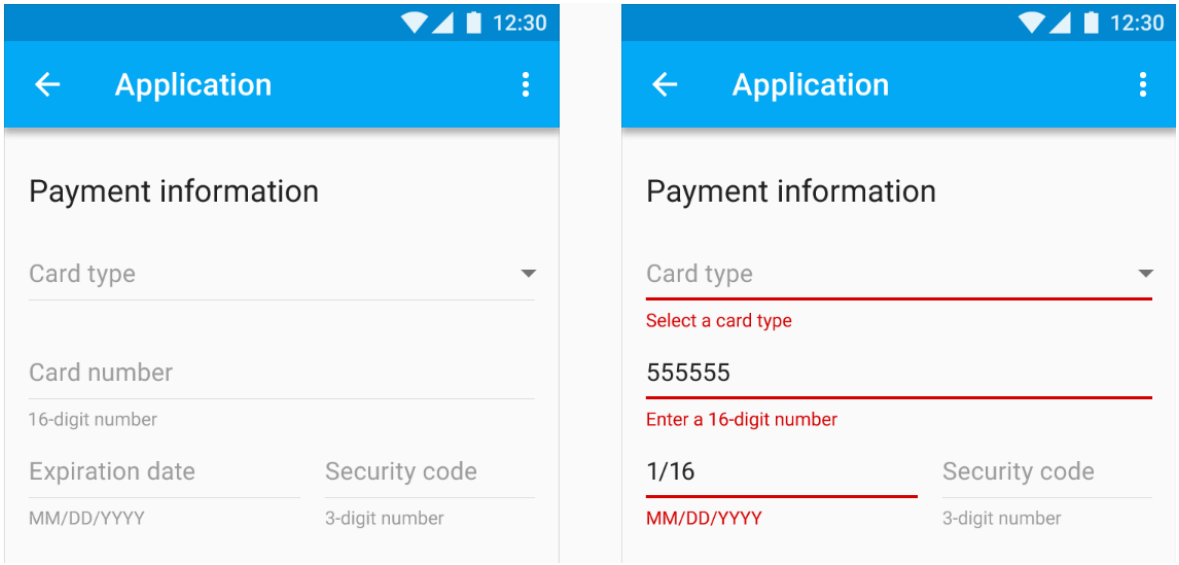
Saavutettavuus

Saavutettavuus (engl. *accessibility*) on olennaista jokaiselle käyttäjälle: se ei tarkoita ainoastaan kompromisseja rajoitteellisille, vaan hyvää, kattavaa suunnittelua kaikille. Saavutettavuusstandardien noudattamiseksi ei tarvita valtavia uhrauksia ajasta, rahasta tai visuaalisesta houkuttelevuudesta, kun saavutettavuus otetaan huomioon jo suunnitteluvaiheessa. (Ustwo 2014, 7, 35.)

Saavutettavuuden kannalta on tärkeää pitää sovelluksen muotoilu mahdollisimman selkeänä ja ytimekkäänä, jotta liika sisältö ei kuormita käyttäjää. Liikaa kuormitusta voidaan välttää näyttämällä käyttäjälle vain tilanteessa relevanttia tietoa, ja paljastaa yksityiskohtia sitä mukaa, kun niitä tarvitaan. Yhtä lailla on tärkeää tehdä sisällöstä mobiililaitteille sopivaa.

Kosketusnäytöllisille laitteille suunnitellessa on huomioitava elementtien koko ja sijainti suhteessa sormien kokoon. Muun muassa painikkeilla olisi hyvä olla vähintään 7 neliömillimetrin kosketusala, ja niiden väliin on hyvä jättää vähintään 2 millimetrin väli, jotta vältetään virhepainallukset. (Ustwo 2014, 37, 48.)

Tuotteen käytön aikana mahdollisesti ilmeneviä virheitä (engl. *errors*) on syytä yrittää minimoida. Käyttäjää kannattaa huomauttaa tehtävien päätöksien vakavuudesta varoituksilla ja vahvistuksilla, jotta hän ei vahingossa tee peruuttamattomia toimintoja. Käyttäjälle on hyvä osoittaa, että syötetty data on oikeassa muodossa. Jos data on väärässä muodossa (esimerkiksi puhelinnumero on liian lyhyt), käyttäjää pyydetään tarkistamaan data täytettävän kentän läheisyyteen sijoitetulla virheilmoituksella (Kuva 3). Paras tapa välttyä syötevirheeltä on tarjota käyttäjälle oletusarvot vapaan syöttämisen sijaan. (Ustwo 2014, 51–53.)



Kuva 3. Vääränmuotoisesta datasta johtuvia virheilmoituksia (mukailtu Material Design 2014)

Yhtenäisyys tyylin, navigoinnin, typografian ja kielenkäytön suhteen tulee säilyä siirryttäessä sivulta toiselle. Käyttöliittymäelementtien tulee käyttäytyä samalla tavalla aina näyttäytyessään, ja ne kannattaa sijoittaa samaan paikkaan kaikilla näytöillä; jos tietty painike suorittaa tietyn toiminnon tietyllä sivulla, tulee sen suorittaa vastaava toiminto kaikilla muillakin sivuilla. Tämä auttaa käyttäjää ennakoimaan tuotteen käyttäytymistä tuotteesta saatujen aiempien kokemusten perusteella. (Ustwo 2014, 38–39.)

Navigoinnissa on syytä välttää ylimääräisiä vaiheita. Käyttäjän on helpompi saada lisää sisältöä näkyviin selaamalla sivua, kuin siirtymällä linkin perässä yhä seuraavalle sivulle. Linkin tulee aina kertoa määränpäänsä, jotta käyttäjän on helpompi navigoida etsimäänsä

sisältöön. Linkin tai painikkeen kuvauksen tulee kuitenkin olla mahdollisimman lyhyt ja selkeä. (Ustwo 2014, 41, 43, 46.)

Visuaalisten apujen, kuten kuvakkeiden ja värien, tarjoaminen käyttäjälle auttaa sivun tärkeän sisällön tunnistamisessa. Värikoodaus helpottaa tunnistamaan tiettyyn kontekstiin kuuluvat asiat. Värin lisäksi on syytä kuitenkin käyttää myös vaihtoehtoista viestintätapaa, koska eri laitteilla ja erityyppisessä valaistuksessa värit voivat näyttäytyä eri tavoin. Tekstin ja taustavärin välisen kontrastin huomioiminen varmistaa, että tärkeä informaatio on saatavilla käyttäjälle kaikissa olosuhteissa. (Ustwo 2014, 74–77.)

Responsiivisuus

Käyttöliittymää suunnitellessa täytyy huomioida muun muassa se, missä ympäristössä ja millä laitteilla tuotetta tullaan käyttämään (Ustwo 2014, 6). Maailmanlaajuisesti yhä useammat ihmiset käyttävät mobiililaitetta päästäkseen internettiin kuluttamaan sisältöä ja tekemään liiketoimia. Responsiivinen suunnittelu on pakollista, jotta verkkosivuston voi optimoida mobiililaitteille. (Maioli 2018.)

Responsiivisen suunnittelun tarkoitus on luoda sivuston ulkoasu, jonka design mukautuu näyttämään hyvältä ja yhtenäiseltä kaikilla laitteilla. Mobiiliteknologian räjähdysmäisen leviämisen myötä alettiin suunnittelemaan web-sivustoja myös mobiililaitteille, mutta ennen responsiivisen suunnittelun mahdollisuutta jokaiselle laitteelle piti luoda erillinen versio samasta sivustosta. Responsiivisuuden kehittämisen myötä sivusto saadaan reagoimaan eri näyttökokoihin, jolloin design näyttää yhtenäiseltä ja kauniilta käytettävästä laitteesta riippumatta. (Beaird & George 2014, 40–42).

Esimerkiksi televisio ja mobiililaite ovat erilaisia ominaisuuksiltaan, ja siksi niihin suunniteltuissa käyttöliittymissä toimivat eri asiat; on syytä harkita esimerkiksi fonttikokoa ja rivin leveyttä sen mukaan, että teksti on varmasti hyvin luettavissa käytettävällä laitteella (ustwo 2014, 6). Pöytätietokonetta ja kannettavaa käytetään eri tavoin ja erilaisissa tilanteissa kuin puhelinta ja tablettia, joten on otettava huomioon myös konteksti ja ympäristö, joissa laitetta käytetään (Maioli 2018).

On tärkeää testata designia niillä laitteilla, joilla kohdekäyttäjän oletetaan käyttävän tuotetta. Mobiililaitteen näytön resoluutiolla ja teknologialla voi olla merkittävä ero tietokoneeseen, sillä esimerkiksi datan syöttötapa vaihtelee käytettävän laitteen mukaan. Nykyään on saatavilla laaja valikoima reaaliaikaisia esikatselutyökaluja, joiden avulla design voidaan sijoittaa kohdelaitteelle testausta varten. (Ustwo 2014, 15.)

Typografia

Typografiaa usein aliarvostetaan digitaalisissa tuotteissa, vaikka teksti ilmaisee suurimman osan informaatiosta. Siksi typografian tulee olla korkealla suunnittelun prioriteettilistalla. Hyvä typografia on elintärkeää digitaalisessa suunnittelussa ja se vaikuttaa suuresti myös tuotteen saavutettavuuteen. Käyttäjän on helppo siirtyä riviltä toiselle tekstin ollessa kevyttä ja lukukelpoista, ja muotoilun mahdollisimman yksinkertaista. Rivivälin tulee olla tarpeeksi väljä ja rivien leveyden tarpeeksi tiivis. Liikkuvaa ja välkkyvää tekstiä on syytä välttää. (Ustwo 2014, 13, 65–66, 70–71.)

Parhaan luettavuuden takaamiseksi fonttikoon olisi hyvä olla vähintään 12 pistettä. Kuitenkin mobiililaitteelle ihanteellinen fonttikoko saattaa näyttää pieneltä tietokoneen näytöllä ja päinvastoin, sillä mobiililaitetta pidetään yleensä lähempänä kasvoja. Hyvä lähtökohta tietokoneelle on 16 pisteen fonttikoko, kun taas mobiililaitteella teksti voi olla pienempää, kun käyttäjä voi itse säätää näytön ja silmien välistä etäisyyttä. (Maioli 2018.)

Vaikka typografia on paljon muutakin kuin pelkkä fontti, on fontilla tärkeä rooli siinä, millaiseksi potentiaalinen käyttäjä mieltää tuotteen olemuksen. Täydellisen fontin etsiminen on hyvä aloittaa määrittelemällä tunteet, joita halutaan herättää kohderyhmässä. Suunnitteluvaiheessa kannattaa kokeilla muutamia eri vaihtoehtoja, jotta nähdään, mikä fontti sopii kontekstiin parhaiten ja luo halutun mielikuvan käyttäjälle. (Beaird & George 2014, 151–155.)

Fontteja, fonttikokoja ja kontrastia vaihtelemalla voidaan määrittää hierarkiat ja erottaa leipäteksti otsikoista. Tässäkin tärkeintä on johdonmukaisuus. (Maioli 2018.) Kuvassa 4 esitetään, miten fonttien avulla voi muotoilla tekstiä niin, että esimerkiksi otsikot saadaan erottumaan leipätekstistä. Kuvassa havainnollistetaan myös fontin valinnan vaikutusta luettavuuteen, jolla on suora yhteys käyttäjäkokemukseen.

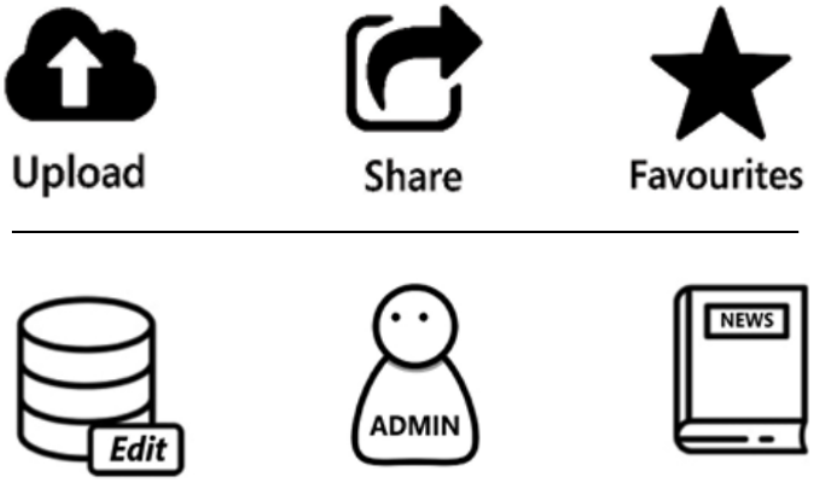
Kuva 4. Tekstin muotoilua fonteilla (Maioli 2018)

Ikonografia

Ikonografia on käyttöliittymäsuunnittelun peruskäytäntö, jolla voidaan parantaa huomattavasti käyttökokemusta (Ritter & Winterbottom 2017). Kuvakkeet (engl. *icons*) ovat tapa lyhentää ja yksinkertaistaa tietoa ja merkityksiä helposti muistettaviksi kuviksi. Oikein käytettynä ne ovat tehokas työkalu monimutkaisten käyttöliittymien yksinkertaistamiseen, mutta epäjohdonmukainen kuvaketyyli voi jopa hämmentää käyttäjää ja näyttää epäammattimaiselta. (Grant 2022.)

Kuvakkeet ovat vastuussa käyttäjän navigoinnin ohjaamisesta ja sujuvoittavat käyttäjän ja käyttöliittymän välistä vuorovaikutusta. Kuvake voi edustaa objektia, prosessia tai toimintoa, jolloin käyttäjä tunnistaa visuaalisen metaforan avulla nopeasti ja helposti toiminnon, jonka haluaa suorittaa järjestelmässä. Laitteilla, joilla on vähemmän tilaa vuorovaikutukselle, kuvakkeet auttavat tiivistämään informaation pienempään tilaan. (Maioli 2018.)

Käyttöliittymäsuunnittelussa kannattaa käyttää sellaisia kuvakkeita, joiden merkitys on jo tuttu käyttäjälle. Ei ole syytä keksiä ja luoda uutta kuvaketta, jos on jo olemassa tarpeisiin sopiva. Kuvaketta valittaessa tulee pohtia, onko sen tarjoama visuaalinen metafora tuttu esimerkiksi eri ikäryhmissä ja kulttuureissa. Nykyaikana ei välttämättä tunnisteta kuvakkeita, jotka kuvaavat vanhentuneita teknologioita, kuten tallentamista kuvaava levyke. Kuvakkeeseen on hyvä yhdistää myös sanallinen vihje (Kuva 5), jotta käyttäjän ei tarvitse arvailla, mitä kyseinen kuvake juuri kyseisessä kontekstissa tarkoittaa. Tekstin ei kuitenkaan tule olla kuvakkeessa itsessään. Jotta voidaan selvittää, ovatko kuvakkeet ymmärrettäviä käyttäjille, niitä on testattava kohderyhmällä. (Grant 2022.)



Kuva 5. Teksti kuvakkeessa (mukailtu Grant 2022)

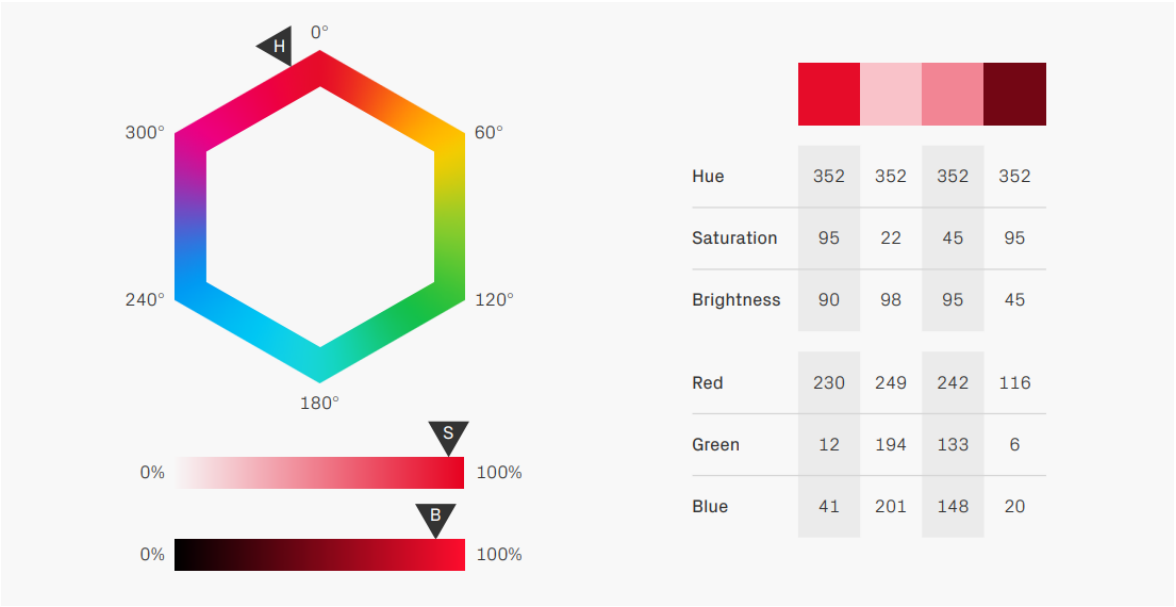
Värimalli

Väri on vaikutusvaltainen ja voimakas työkalu käyttöliittymäsuunnittelussa, ja sillä voidaan vaikuttaa käyttäjän ja käyttöliittymän väliseen vuorovaikutukseen. Väripsykologian mukaan värit eivät vaikuta jokaiseen ihmiseen samalla tavalla, vaan eri tekijät vaikuttavat siihen, miten ihminen mieltää tietyn värin. Tällaisia tekijöitä ovat esimerkiksi ikä, kulttuuri ja sukupuoli. Tietyn kohderyhmän kanssa työskennellessä onkin tärkeää selvittää, minkälaisia värien hahmottamiseen mahdollisesti vaikuttavia ominaisuuksia käyttäjillä on. Kuvassa 6 esitetään yleisimmät värit ja niihin yhdistettävät tunteet ja mielleyhtymät. (Ritter & Winterbottom 2017.)

Kuva 6. Värien mielleyhtymät (Ritter & Winterbottom 2017)

Vaikka väripsykologia vaikuttaa suuresti värien valintaan, värimaailman menestys riippuu lopulta valittujen värien välillä vallitsevasta harmoniasta. Jotta löydetään hyvin yhdessä toimivat värit, täytyy luoda värimalli. Värimallit (engl. *color schemes*) ovat peruskaavoja harmonisten ja tehokkaiden väriyhdistelmien luomiseen. (Beaird & George 2014, 59, 65.)

Kun lähdetään luomaan värimallia, valitaan ensin perusväri. Muut värit valitaan sen mukaan, minkälainen värimalli halutaan luoda. Esimerkiksi monokromaattinen värimalli koostuu ainoastaan yhden perusvärin eri sävyistä, eli sen tummemmista ja/tai vaaleammista muunnoksista, kun taas analogisessa värimallissa valitaan väriympyrässä vierekkäin sijaitsevat värit. (Beaird & George 2014, 59, 65–79.) Värimallia luotaessa HSB-malli (Kuva 7) on tehokas tapa luoda erilaisia perusvärin sävyjä: sävy (engl. *Hue*) pidetään samana, mutta kylläisyyttä (engl. *Saturation*) ja kirkkautta (engl. *Brightness*) säädetään muunnelmien luomiseksi (Ustwo 2014, 23).

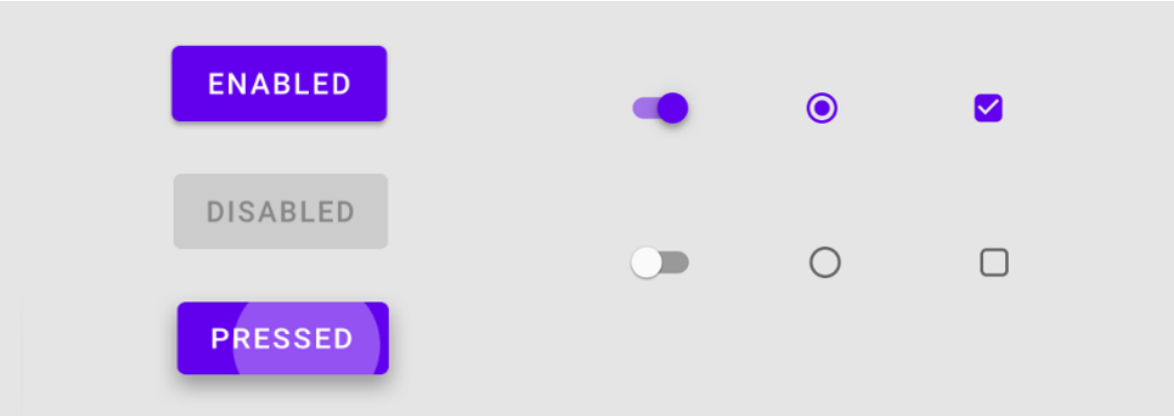


Kuva 7. HSB:n avulla luotu värimalli (Ustwo 2014, 23)

Riittävä kontrasti tekstin ja taustavärien välillä on välttämätöntä interaktiiviselle suunnittelulle. Siksi lopulliseen väripalettiin kannattaa valita myös kontrastia lisääviä värejä, joita voidaan käyttää esimerkiksi taustan ja tekstin väreinä. Tätä varten on saatavilla värityökaluja, jotka auttavat valitsemaan palettiin värit luodun värimallin rinnalle. (Beaird & George 2014.)

Tilat

Interaktiivisille elementeille kannattaa määrittää oletustilan lisäksi ylimääräiset tilat (engl. *states*) (ustwo 2014, 29). Tilat ovat visuaalisia esityksiä, joita käytetään viestimään komponentin tai interaktiivisen elementin, kuten painikkeen, status käyttäjälle. Tilojen avulla käyttäjälle selviää muun muassa, onko painike käytettävissä ja kumpi ’päällä’ vai ’pois’ -vaihtoehdoista on valittu (Kuva 8). Tilat voidaan ilmaista esimerkiksi värin, varjon tai kursorin avulla. (Material Design 2018a.)



Kuva 8. Elementtien tilat (mukailtu Material Design 2018a)

Tilat voivat vaihdella alustan ja syöttötavan mukaan, mutta ylimääräiset tilat kannattaa suunnitella jo ennen toteutusvaihetta, jotta varmistetaan tilojen vuorovaikutuksen taso ja tyylin sopivuus muuhun designiin (Ustwo 2014, 29). Jokaisen tilan tulee olla visuaalisesti samantyylinen, mutta siinä on oltava selkeät ominaisuudet, jotka erottavat sen muista tiloista (Material Design 2018a).

Visuaalinen tyyliopas

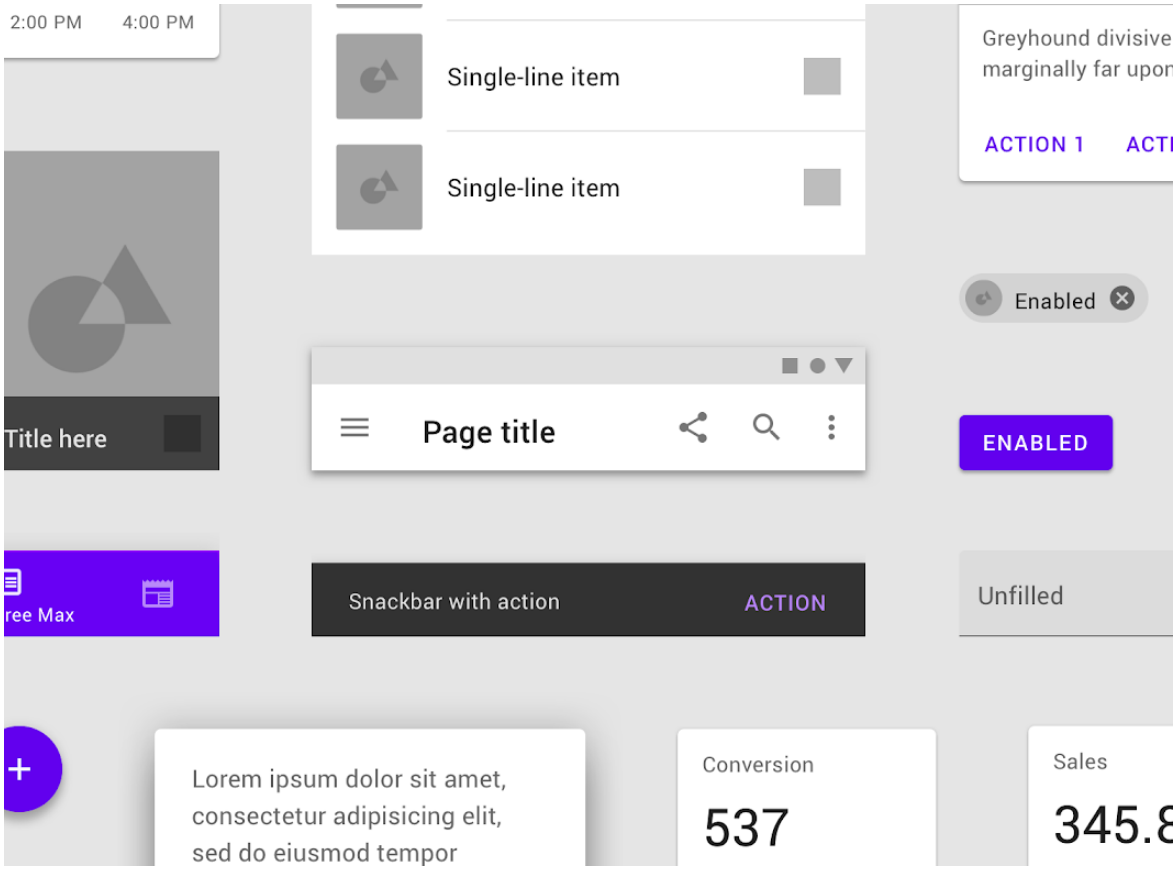
Visuaalinen tyyliopas on projektissa kätevä resurssi, joka kokoaa kaikki projektin tyylit yhteen paikkaan. Tyylioppaan avulla kaikki projektin elementit saadaan näyttämään yhtenäisiltä, ja sekä projektin suunnittelija että kehittäjä pysyvät kartalla projektin tyyliperiaatteista. (Ustwo 2014, 100.)

Material Design on Googlen luoma visuaalinen tyyliopas, jonka tarkoituksena on ylläpitää käyttöliittymän suunnitteluelementtien yhtenäisyyttä eri alustoilla. Se perustuu designin kymmeneen perinteiseen sääntöön, jotka korostavat neutraalia, läpinäkyvää, yksinkertaista ja ajatonta designia. Tyyliopas sisältää useita käyttöliittymäelementtityylejä, jotka noudattavat Material Designin kolmea pääperiaatetta:

- Material (suom. *aineellinen*) on metafora: Suunnitteluopas saa inspiraationsa fyysisestä maailmasta ja sen tekstuureista.
- Selkeä, havainnollinen, tarkoituksellinen: Painatuspohjaisen suunnittelun peruselementit (typografia, ikonografia, mittakaava, kuvien käyttö, jne.) ohjaavat oppaan tyyliä.
- Liike antaa merkityksen: Liike kiinnittää huomion ja ylläpitää jatkuvuutta hienovaraisen palautteen ja johdonmukaisten siirtymien avulla. (Ritter & Winterbottom 2017.)

Käyttöliittymäelementtityylit voidaan jakaa kategorioihin, jotka antavat selkeät suuntaviivat kutakin elementtiä koskeviin parhaisiin käytäntöihin (Ritter & Winterbottom 2017). Kuvassa

9 esitellään Material Design -tyylioppaan eri elementtejä ja niille luotua tyyliä. Elementit noudattavat keskenään yhtenäistä ulkoasua, joka perustuu edellä läpikäytyihin periaatteisiin.



Kuva 9. Material Design elementtejä (Material Design 2018b)

4 Järjestelmäpäivityksen toteutus

4.1 Vaatimusmäärittely

Opintojenseurantatyökalun jatkokehitysprojekti alkoi tutkimuksella, jossa valmentajilta kerättiin tarpeita ja toiveita siitä, mitä heidän tulee pystyä sovelluksella tekemään. Konseptivaiheessa tarpeiden ja toiveiden pohjalta koottiin vaatimuslista niistä ominaisuuksista, jotka sovelluksen käyttöönottoa varten täytyi projektin aikana toteuttaa. Päivityksen päävaatimuksena oli valmentajan puolelle toimiva käyttöliittymä. Sivuvaatimuksena oli opiskelijan puolen virheenkorjaukset, jotta sovellus olisi valmis käytettäväksi.

Valmentajan puolen käyttöliittymän toiminnalliset vaatimukset:

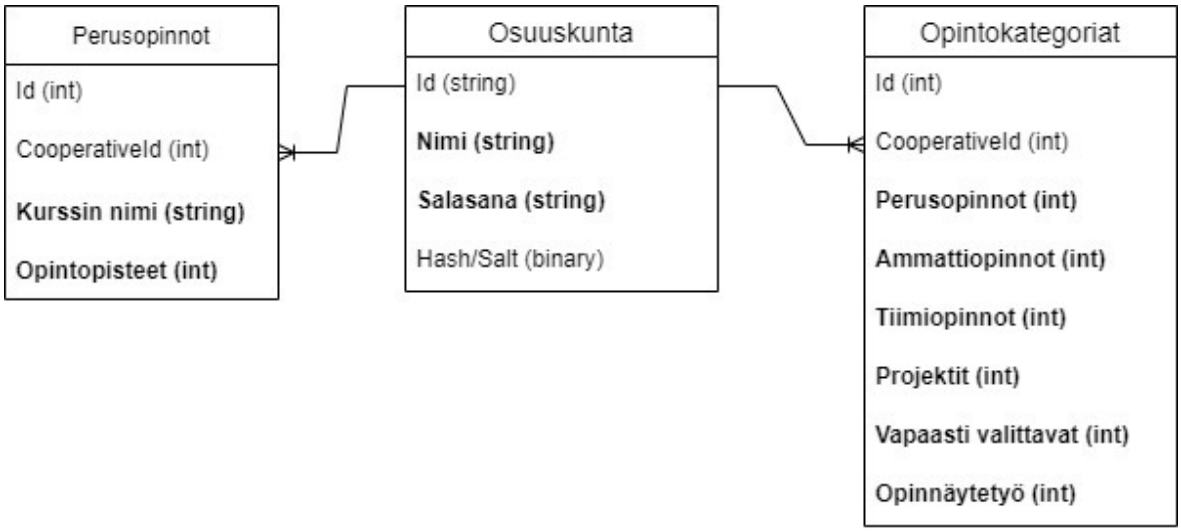
- Uuden osuuskunnan luominen
- Osuuskunnan asetusten muokkaaminen
- Opiskelijan poistaminen osuuskunnasta
- Opiskelijan edistymisen seuranta
- Opiskelijan kirjauksen arviointi
- Opiskelijan kirjausten vieminen Exceliin
- Kirjalista, johon voi lisätä ja poistaa kirjoja.

Valmentajien toivomia ei-toiminnallisia vaatimuksia olivat nopea- ja helppokäyttöisyys sekä tietokoneella käytettävyys. Lisäksi kapasiteettivaatimuksena oli skaalautuvuus vähintään kaikille Saitemian alaisten osuuskuntien opiskelijoille.

Sovellukselle oli myös keksittävä uusi työnimi, joka myös käyttöliittymässä esiintyisi. Koska sovellusta kehitetään markkinointiopiskelijoille, päätettiin heidän osaamistaan hyödyntää nimen valinnassa. Markkinointiopiskelijoille järjestettiin nimi-innovointi, jonka tuloksena valittiin parempi työnimi edellisen HOPS2.0:n tilalle. Uudeksi nimeksi muovautui S-Pace, joka tulee sanoista *Study Pace* eli opintotahti, ja sillä nimellä sovellusta kutsuttiin projektin aikana.

4.2 Tietokantasuunnittelu

Järjestelmän tietokantaan tehtyjä muutoksia lähdettiin suunnittelemaan valmentajilta saatujen vaatimusten perusteella. Uusien haluttujen ominaisuuksien pohjalta tietokantaan täytyi luoda kaksi uutta taulua: osuuskuntakohtaiset kategorioiden pisteet ja perusopinnot (Kuva 10). Lisäominaisuutena toteutettiin kirjalista, jonka toimiminen vaati toiset kaksi taulua: kirja ja teema.



Kuva 10. Suunnitelma tietokantaan lisättävistä tauluista

Lisäksi tietokantaa tuli yksinkertaistaa liittämällä opiskelija- ja valmentajataulut yhdeksi *User*-tauluksi. Tämän muutoksen tarkoitus oli yksinkertaistaa kirjautumisen ja käyttäjienhallinnan logiikkaa. Muutos vähentää ohjelmistokoodin määrää ja monimutkaisuutta, kun kaikki käyttäjiin liittyvät toiminnot suoritetaan samalla tavalla käyttöliittymässä ja backendissä. Ohjelmistokoodin yksinkertaistaminen myös lisää järjestelmän turvallisuutta.

4.3 Visuaalinen ja toiminnallinen suunnittelu

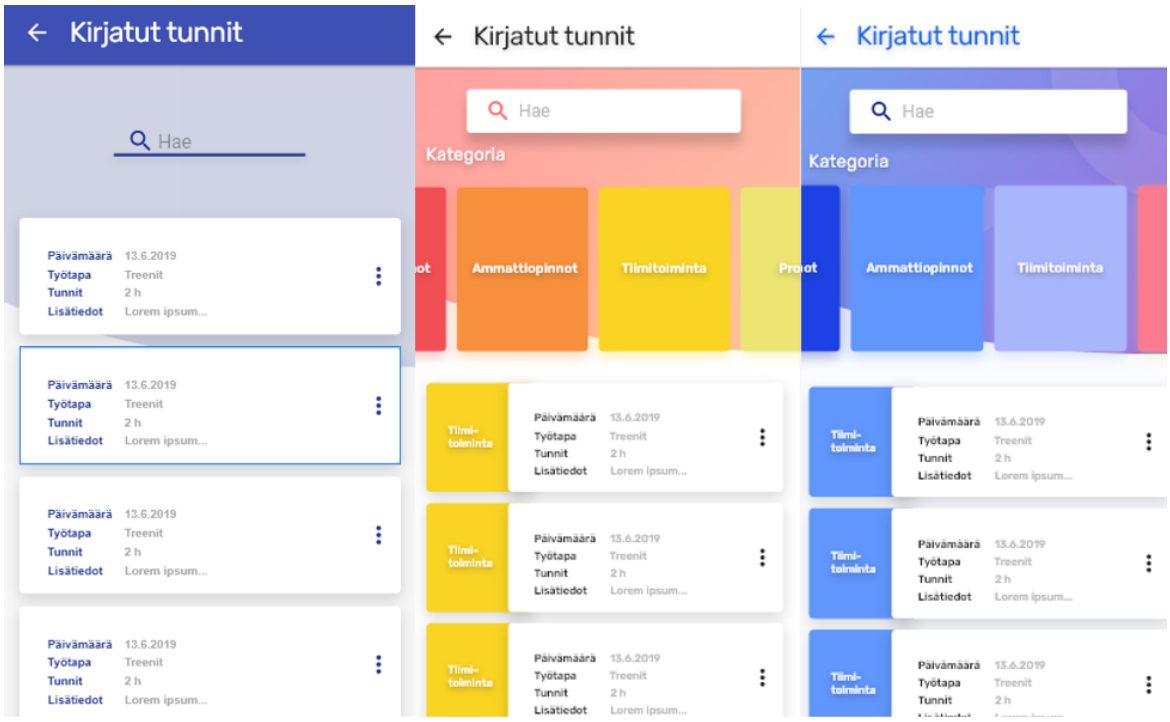
Valmentajan puolen käyttöliittymän visuaalinen ja toiminnallinen suunnittelu aloitettiin kerättyjen vaatimusten pohjalta. Suunnittelussa keskityttiin käyttöliittymän selkeään ja sujuvaan toiminnallisuuteen, jossa vältetään ylimääräisiä vaiheita. Valmentajille oli tärkeää, että opiskelijoiden etenemisen ja kirjaukset voi nähdä nopeasti yhdellä silmäyksellä, mikä sujuvoittaa osuuskuntien tiimipalaverien kulkua.

Vaikka opiskelijan puoli on suunniteltu pääasiassa mobiilissa käytettäväksi, valmentajien tarvetta vastaa paremmin tietokoneella käytettävä käyttöliittymä. Jotta valmentajan puolen käyttöliittymä sopisi yhteen opiskelijan puolen kanssa, sen suunnittelussa painotettiin ulkoasujen yhtenäisyyttä. Yhtenäisyys vaikuttaa positiivisesti sovelluksen käytettävyyteen ja saavutettavuuteen.

4.3.1 Värimallin luominen

Iso valmentajan ja opiskelijan puolten yhtenäisyyteen vaikuttava tekijä on yhteinen värimalli. Opiskelijan puolta kehitettäessä värimalli muovautui sitä mukaa, kun valittujen värien havaittiin herättävän sovelluksen tarkoitukseen ja kohderyhmään nähden ristiriitaisia mielikuvia. Kuvassa 11 esitetään opintojenseurantatyökalun värimallin historia: Ensimmäisen

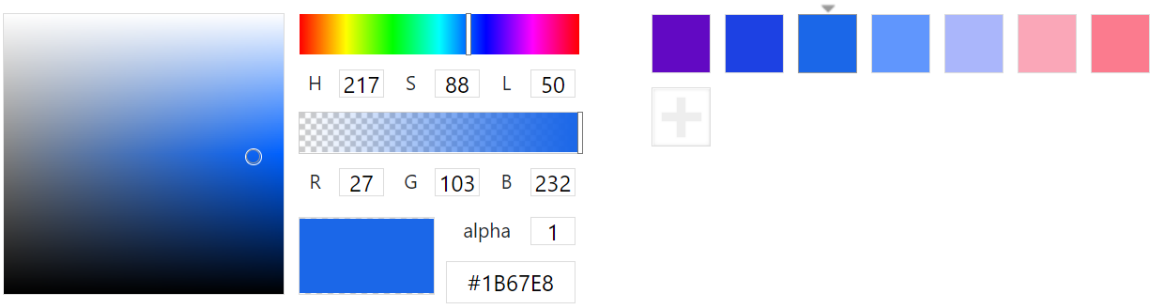
version designissa käytettiin monokromaattista sinistä värimallia, mutta sininen väri synnytti liian virallisen ja monotonisen mielikuvan. Pelkistetty sininen värimalli vaihdettiin värikkäämpään punaisen, oranssin ja keltaisen pohjalta luotuun värimalliin, mutta se tuntui liian huomiota herättävältä ja räikeältä, eikä ollenkaan harmoniselta. Lopulta rauhallinen sininen väri tuotiin takaisin, ja sen ympärille koottiin analoginen värimalli, jossa sinistä pehmennetään violetin sävyillä. Kontrastia tuomaan valittiin hieman vaaleanpunaista. Tämä värimalli todettiin hyväksi opiskelijan puolella, joten siitä syntyvien mielikuvien ja yhtenäisyyden vuoksi se valittiin myös valmentajan puolen käyttöliittymään.



Kuva 11. Sovelluksen värimallin historia

Beairdin ja Georgen (2014) mukaan sinistä pidetään yleisesti rakastettavana värinä. Sininen symboloi avoimuutta, älykkyyttä ja uskoa, ja sillä on todettu olevan rauhoittavia ja jännitystä vähentäviä vaikutuksia. Violetti sen sijaan tasapainottaa punaisen stimulaatiota ja sinisen rauhoittavaa vaikutusta. Patrick McNeil kertoo violetin olevan yksi web-suunnittelussa vähiten käytetyistä väreistä, joten sen käyttäminen on hyvä keino erottua joukosta.

Värien valinnassa hyödynnettiin erästä HSB-värityökalua (Kuva 12). Sen avulla näki nopeasti, sointuvatko värit hyvin yhteen. Lisäksi työkalu antoi värien HEX-värikoodit, joita tarvittiin sovelluskehityksen myöhemmissä vaiheissa.

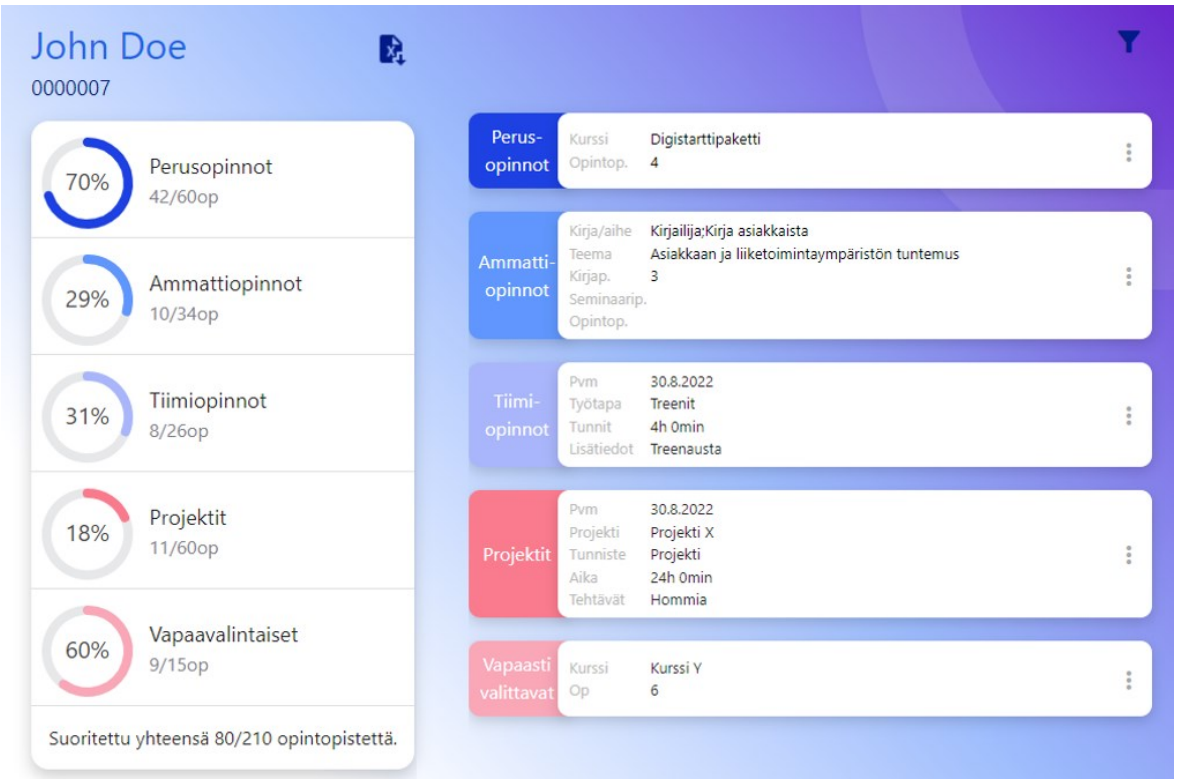


Kuva 12. Värimallin luominen HSB-värityökalulla

Värikoodaus

Opintokategorioiden tunnistamiseksi hyödynnettiin värikoodausta. Väreiksi valittiin värimalliin sopivia sinisen, violetin ja vaaleanpunaisen sävyjä sillä perusteella, että ne ovat harmoniassa keskenään, mutta erottuvat kuitenkin toisistaan selkeästi.

Kuvassa 13 esitetään, kuinka värikoodausta hyödynnetään valmentajan näkymässä sivun eri osioissa. Opiskelijan edistymisen seurannassa käytetään samaa kategorioiden värikoodausta kuin yksittäisissä opiskelijan tekemissä kirjauksissa. Värikoodauksen avulla sekä opiskelijan että valmentajan on helppo huomata nopealla silmäyksellä, mistä opintokategoriasta on kyse.



Kuva 13. Opintokategorioiden värikoodaus

4.3.2 Prototyyppi

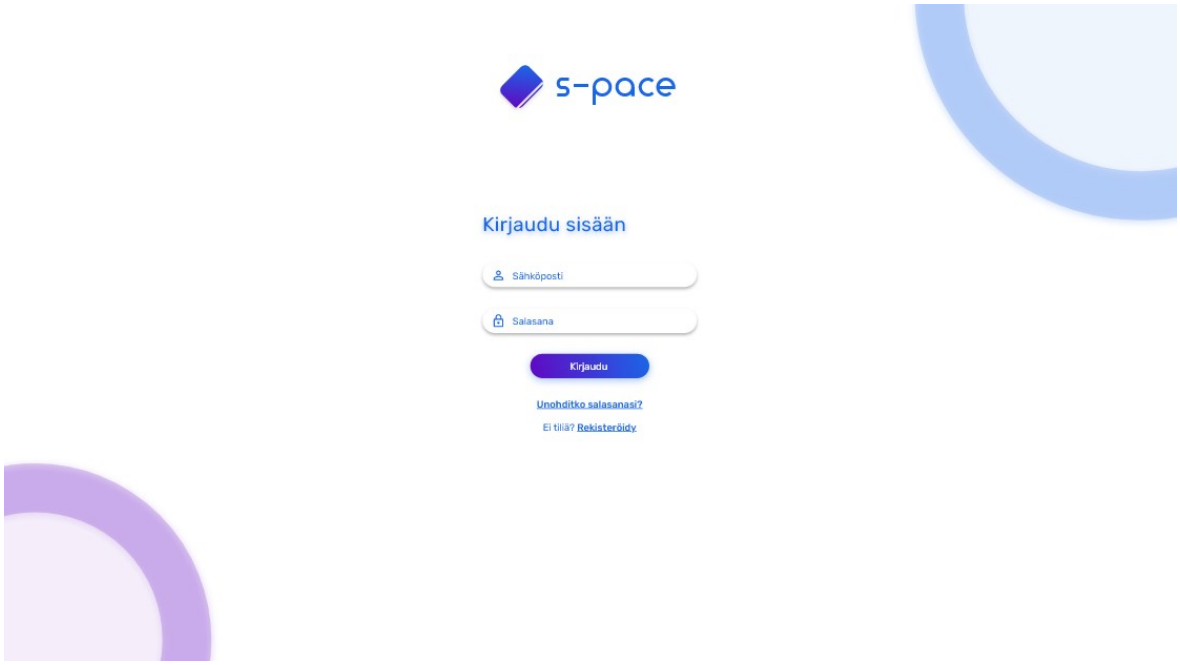
Opintojenseurantatyökalun prototyyppi luotiin Adobe XD:llä. Rae (2020) kuvailee Adobe XD:tä tehokkaaksi ja helppokäyttöiseksi vektoripohjaiseksi suunnittelualustaksi, joka sisältää työkalut maailman parhaiden käyttäjäkokemusten luomiseen. Se mahdollistaa suunnittelijoiden ja suunnittelutiimien ripeän yhteistyön suunnitteluprosessin aikana aina ideoinnista kehittäjille viemiseen asti. Adobe XD:llä luodaan todellisen näköisiä ja tuntuisia klikattavia prototyyppejä.

Klikattavat prototyypit ovat hyödyllisiä tehtävävirtojen ja toimintojen testaamiseen ja validointiin. Klikattavan prototyypin avulla alustava idea voidaan esitellä asiakkaille ja sitä voidaan käyttää dokumenttina ohjelmistokehitysryhmälle helpottamaan virtojen ymmärtämistä. Prototyyppien ei tarvitse olla kokonaisia, vaan riittää, että ne havainnollistavat tärkeimmät tehtävävirrat. (Maioli 2018.) Valmentajan puolen klikattavaan prototyyppiin koottiin kaikki tärkeimmät ominaisuudet, jotta idea käyttöliittymän toiminnallisuuksista ja ulkoasusta tulee esille.

Koska valmentajat tulevat käyttämään opintojenseurantatyökalua pääasiassa tietokoneella, täytyi aluksi suunnitella valmentajan puolen käyttöliittymän peruselementit (Kuva 14), eli tausta, navigointipalkki ja yläpalkki, desktop-versioon sopiviksi. Peruselementit pysyvät samoina valmentajan navigoidessa eri komponentteihin. Yläpalkista käy ilmi, mitä sovellusta käytetään. Navigointipalkki sisältää tiedon siitä, millä käyttäjällä sovellukseen on kirjauduttu, sekä painikkeet, joiden avulla navigoidaan sisällöstä toiseen. Nämä elementit suunniteltiin opiskelijan käyttöliittymän valmiiden elementtien pohjalta, jotta valmentajan käyttöliittymä olisi opiskelijan puolen kanssa yhtenäinen.

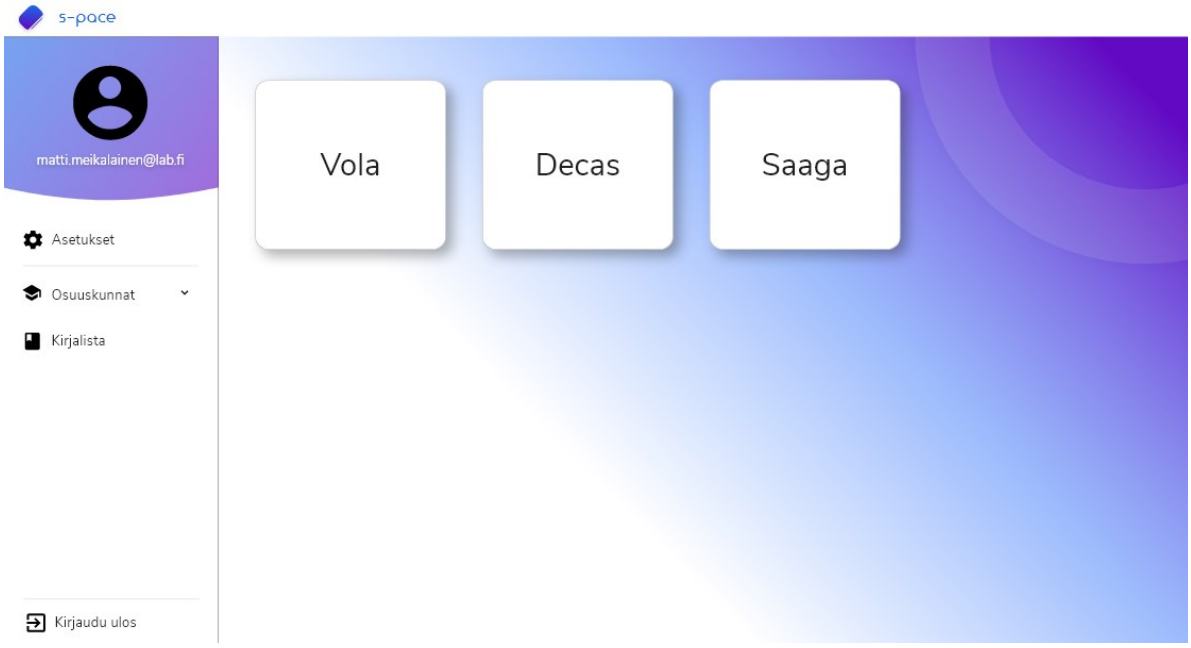
Kuva 14. Valmentajan käyttöliittymän peruselementit

Aluksi suunniteltiin myös kirjautumissivun desktop-versio (Kuva 15). Vaikka valmentaja kirjautuu sovellukseen samalta sivulta kuin opiskelija, se vaati responsiivista suunnittelua, jotta sivu mukautuu ja näyttää hyvältä myös tietokoneen näytöllä.



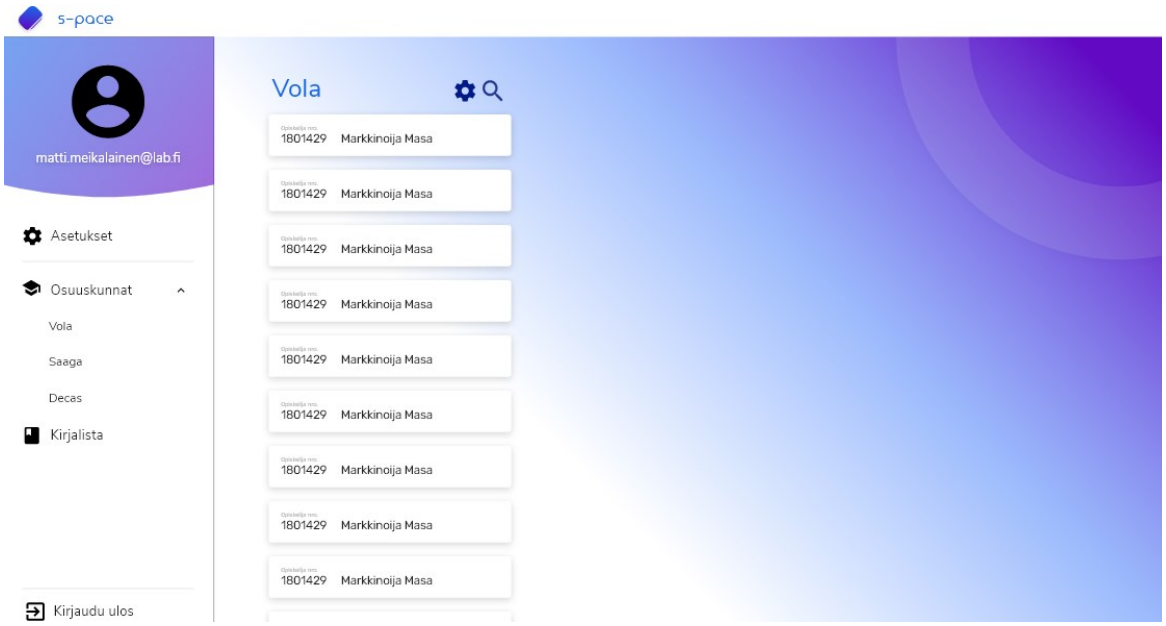
Kuva 15. Kirjautumissivun desktop-versio

Seuraavaksi lähdettiin koostamaan valmentajan puolelle sisältöä. Kun valmentaja on kirjautunut sisään sovellukseen, avautuu yksinkertainen kotinäkymä (Kuva 16), josta valmentaja näkee yhdellä silmäyksellä kaikki järjestelmään lisätyt osuuskunnat.



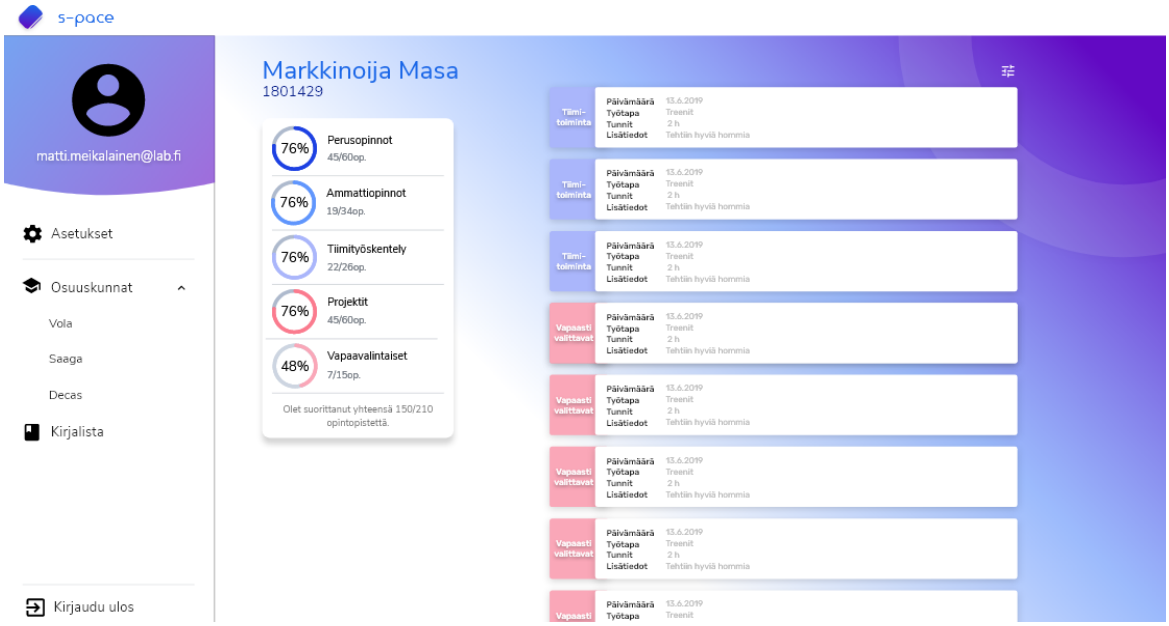
Kuva 16. Valmentajan kotinäkymä

Kun valmentaja on valinnut osuuskunnan, avautuu osuuskuntanäkymä (Kuva 17), jossa on lista kaikista osuuskunnan opiskelijoista. Näkymästä voi navigoida yhä eteenpäin Osuuskunnan asetukset -sivulle, jossa valmentaja voi muokata osuuskunnan opintokategorioita ja perusopintokursseja. Osuuskuntanäkymään voi navigoida myös navigointipalkin kautta.



Kuva 17. Osuuskuntanäkymä

Kun osuuskuntanäkymästä on valittu opiskelija, jonka opintoja halutaan tarkastella, avautuu opiskelijan eteneminen sekä lista kaikista opiskelijan tekemistä kirjauksista värikoodattuna opintokategorian mukaan (Kuva 18). Kyseisestä näkymästä valmentaja näkee kaikki opiskelijan tärkeimmät tiedot ja jo suoritetut opinnot.



Kuva 18. Opiskelijan etenemisen seuranta

Opintojenseurantatyökalun kehityksessä on jo ennen tätä projektia käytetty Material Design -tyylioppaan elementtejä tai elementit on muotoiltu tyylioppaan perusperiaatteita noudattaen. Siksi muun muassa prototyypissä käytetyt painikkeet, ikonografia ja kortit noudattavat Material Designin muotoilua. Prototyyppiin valittiin selkeä ja koruton fontti, koska sen ei ole tarkoitus herättää suuria mielikuvia. Fontin valinnassa kiinnitettiin huomiota myös sen moderniin olemukseen.

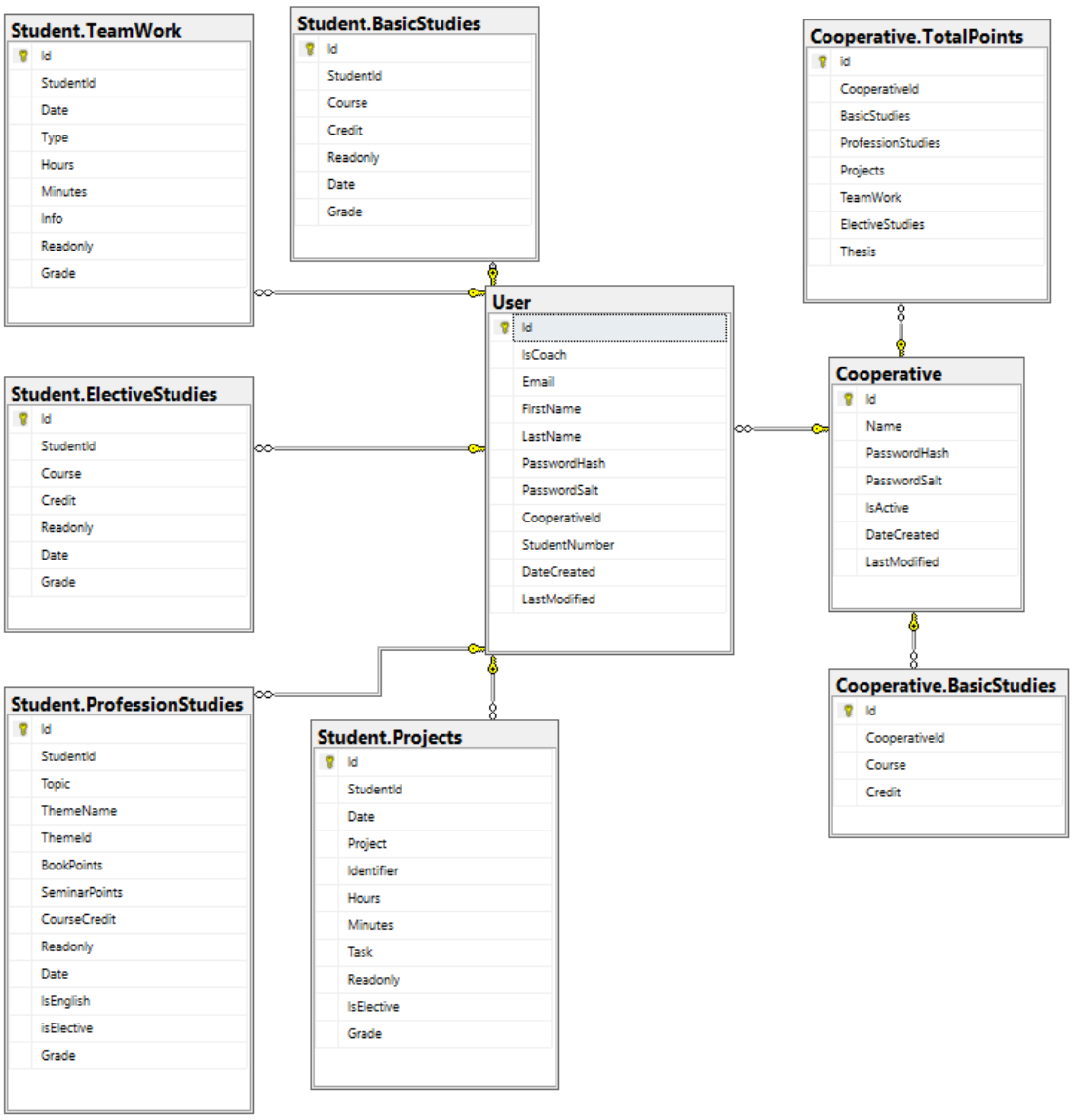
4.4 Tekninen toteutus

Opintojenseurantatyökalun päivityksen tekninen toteutus pyrittiin toteuttamaan mahdollisimman vähäisillä järjestelmän muutoksilla, minkä vuoksi tietokannan, backendin ja frontendin teknologiat pidettiin samoina kuin järjestelmän tähänastisessa kehityksessä. Tietokanta pysyi Microsoft SQL -relaatiotietokantana, backend pysyi ASP.NET:llä toteutettuna REST API -rajapintana ja frontend pysyi Angular-web-sovelluksena.

Järjestelmäpäivityksen edellyttämät muutokset tehtiin ensimmäisenä tietokantaan, jonka jälkeen uusille tietokannan tauluille toteutettiin toiminnallisuudet backendissä. Lopuksi suunnitteluvaiheessa luodun prototyypin pohjalta toteutettiin frontendissä käyttöliittymän ulkoasu sekä toiminnot, jotka kutsuvat backendin uusia päätepisteitä (engl. *endpoints*), joista käyttäjän haluama tieto haetaan tai johon se lähetetään.

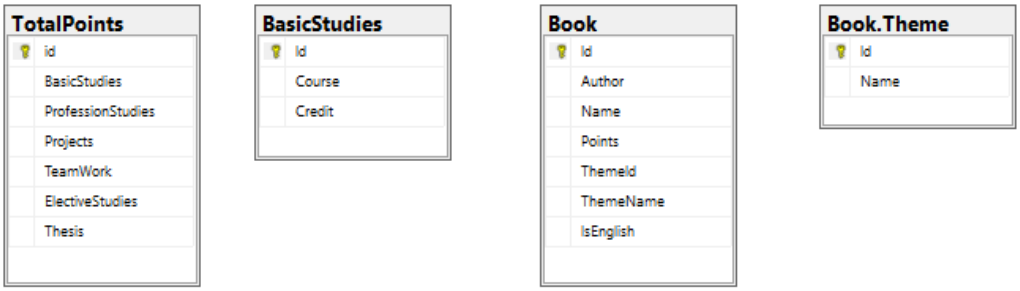
4.4.1 Tietokanta

Järjestelmän tietokannan rakenteeseen tehtiin muutoksia vaatimusmäärittelyn ja tietokantasuunnittelun perusteella. Tietokannan erilliset taulut valmentajien ja opiskelijoiden käyttäjille yhdistettiin *User*-tauluksi, johon lisättiin *isCoach* -sarake indikoimaan käyttäjän tyyppiä, eli sitä, kuuluuko käyttäjätunnus opiskelijalle vai valmentajalle. Tietokantaan lisättiin suunnitelman mukaiset taulut osuuskuntakohtaisille kategorioiden pisteille (*Cooperative.TotalPoints*) sekä perusopinnoille (*Cooperative.BasicStudies*). Kuvassa 19 esitetään relaatiot tietokannan taulujen välillä. Relaatioiden mukaan jokainen käyttäjä kuuluu osuuskuntaan, jokaisella osuuskunnalla on omat asetuksensa ja jokainen opintokirjaus kuuluu käyttäjätunnukselle.



Kuva 19. Tietokannan taulujen yhteydet

Lisäksi tietokantaan luotiin kaksi uutta taulua kirjalistaominaisuutta varten: *Book* ja *Book.Theme* (Kuva 20), jotka sisältävät listassa näytettävät kirjat sekä kirjoille valittavat teemat. Kuvassa 20 näkyy myös tietokantaan lisätyt *TotalPoints*- ja *BasicStudies* -taulut, joihin tallennetaan osuuskunnan luomisessa käytettävät oletusarvot.



Kuva 20. Tietokannan loput uudet taulut

4.4.2 REST-rajapinta

Opintojenseurantatyökalun backend eli REST-rajapinta toteutettiin .NET-kehitysympäristöllä, joka mukautuu monien erityyppisten ohjelmistojen rakentamiseen monipuolisten ominaisuuksiensa sekä kattavan ohjelmointikieli- ja alustatukensa ansiosta (.NET 2022a). ASP.NET on .NETin lisäosa, joka on suunniteltu erityisesti web-ohjelmistojen luomiseen. ASP.NETillä on helppo rakentaa REST API -rajapintoja sen monipuolisten ja kattavien kirjastojen avulla. (.NET 2022b.)

Järjestelmän backend noudattaa REST-arkkitehtuurin ja SoC:n mukaista erottelua, ja se on jaettu kolmeen eri kerrokseen: *controller*, *service*, ja *repository*. Jokaisella kerroksella on sille ominaiset piirteet ja tehtävät, joita se suorittaa. Näiden kerrosten lisäksi backendissä määritettiin tietokantayhteyksien asetukset kontekstitiedostoihin ja tietokannan tauluja vastaavat luokat.

Jokaisella tietokannan taulua vastaavalla luokalla on omat *controller*-, *service*- ja *repository*-luokkansa sekä niiden metodit. C*ontroller*-, *service*- ja *repository*-luokat muodostavat backendissä reittejä, joita pitkin data kulkee tietokannasta frontendille.

Controller-kerroksen tehtävänä on toimia API-rajapinnan uloimpana kerroksena ja vastaanottaa kutsuja ulkopuolisilta sovelluksilta, kuten järjestelmän frontendiltä. Jokaisessa *controller*-luokassa määritettiin päätepisteet, joista rajapinnan eri metodeja voidaan kutsua. Kuvassa 21 on osuuskuntiin liittyvien metodien *controller*-luokka, jossa määritettiin sen reitti, eli osoite, jossa se sijaitsee.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class CooperativesController : Controller
{
    private IMapper _mapper;
    private readonly ICooperativeService _cooperativeService;
    private readonly IUserService _studentService;
    private readonly ICoachService _coachService;
    private readonly OpsSettings _opsSettings;

    0 references
    public CooperativesController(ICooperativeService cooperativeService, IUserService studentService,
    {
        _cooperativeService = cooperativeService;
        _studentService = studentService;
        _coachService = coachService;
        _mapper = mapper;
        _opsSettings = opsSettings.Value;
    }
```

Kuva 21. Osuuskuntien *controller*-luokka

Kuvassa 22 on kyseiseen *controller*-luokkaan luotuja *GET*-metodeja. Luokassa määritettiin metodien tarkemmat osoitteet. Samasta osoitteesta voi kutsua vain yhtä saman tyyppistä metodia, minkä takia metodeille piti määrittää omat osoitteensa.

```
[HttpGet()]
0 references
public IActionResult GetCooperatives()
{
    var result = _cooperativeService.ReadActive();
    return new JsonResult(result);
}

[HttpGet("getall")]
0 references
public IActionResult GetAllCooperatives()
{
    var result = _cooperativeService.Read();
    return new JsonResult(result);
}

// GET: {cooperativeId}/students
[HttpGet("{id}/students")]
0 references
public IActionResult GetStudents(string id)
{
    var result = _coachService.Read(id);
    List<UserDto> dtoList = new List<UserDto>();
    foreach (var item in result)
    {
        dtoList.Add(_mapper.Map<UserDto>(item));
    }
    return new JsonResult(dtoList);
}
```

Kuva 22. *Controller*-luokan metodeja

Service-kerroksen tehtävänä on toimia liiketoimintalogiikan suorittajana ja datan käsittelijänä *controller*- ja *repository*-kerrosten välillä. Puhtaan arkkitehtuurimallin mukaan *service*-kerros ei voi olla tietoinen *controller*-kerroksesta, eikä näin ollen voi kutsua *controller*-luokkien metodeja. Saman periaatteen mukaan *controller*-kerros voi kutsua vain *service*-luokkien metodeja, eikä kutsua suoraan *repository*-luokkien metodeja ohittaen *service*-kerroksen kokonaan. Kuvassa 23 on osuuskuntiin liittyvän *service*-luokan metodeja, jotka kutsuvat osuuskuntiin liittyvän *repository*-luokan metodeja.

```
public List<CooperativeDto> ReadActive()
{
    var list = _cooperativeRepository.ReadActive();
    var cooperatives = new List<CooperativeDto>();
    foreach (var item in list)
    {
        cooperatives.Add(new CooperativeDto
        {
            Id = item.Id,
            Name = item.Name
        });
    }
    return cooperatives;
}

2 references
public Cooperative Update(Cooperative cooperative)
{
    var cooperativeToUpdate = _cooperativeRepository.Read(cooperative.Id);
    if(cooperativeToUpdate == null)
    {
        throw new OpsException("Cooperative not found, cannot update");
    }
    cooperativeToUpdate.IsActive = cooperative.IsActive;
    cooperativeToUpdate.LastModified = DateTime.Now;
    _cooperativeRepository.Update(cooperativeToUpdate);
    return cooperative;
}
```

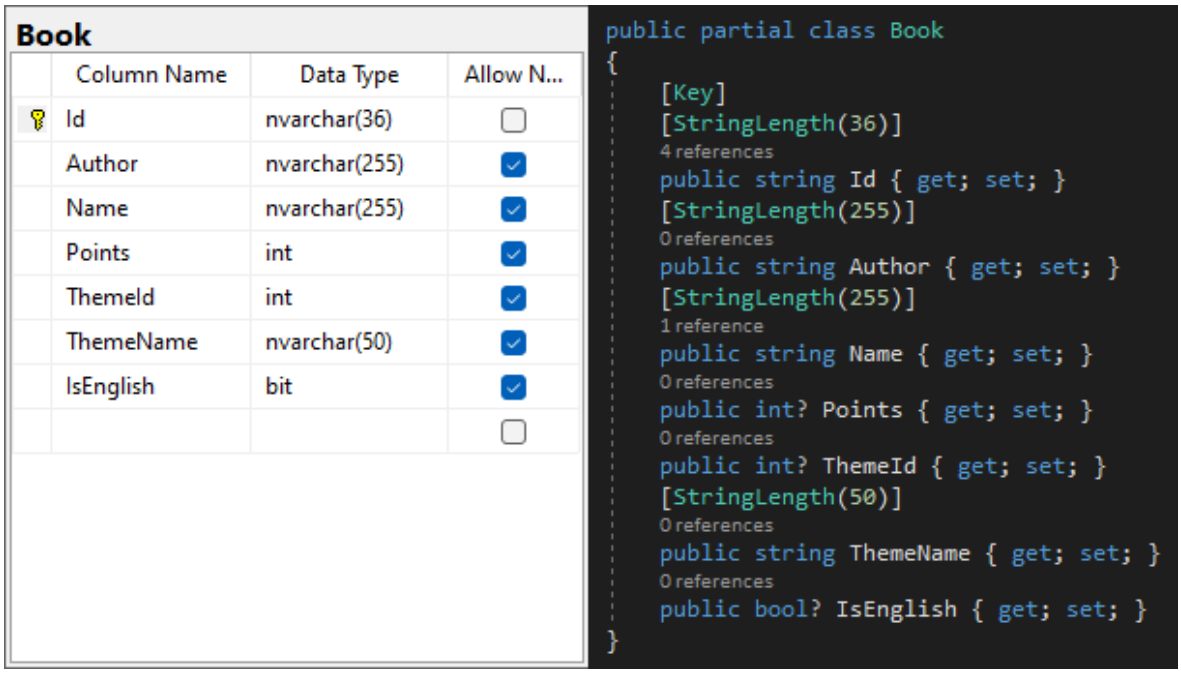
Kuva 23. *Service*-luokan metodeja

REST API:n sisimmäisen kerroksen eli *repository*-kerroksen tehtävänä on tehdä kutsuja tietokantaan. *Repository*-luokka käyttää kontekstitiedostoissa määritettyjä tietokantayhteyksiä, joihin kutsut tehdään. ASP.NET sisältää valmiita kirjastoja, joiden avulla voidaan luoda SQL-kyselyitä ilman, että ohjelmoijan tarvitsee osata SQL:n syntaksia. Kuvassa 24 näkyy osuuskuntiin liittyvän *repository*-luokan metodeja, jotka suorittavat SQL-kyselyitä ASP.NETin LINQ-kirjaston (*Language-Integrated Query*) avulla.

```
6 references
public Cooperative Read(string id)
{
    return _context.Cooperative.AsNoTracking().FirstOrDefault(c => c.Id == id);
}

2 references
public Cooperative Update(Cooperative cooperative)
{
    _context.Update(cooperative);
    _context.SaveChanges();
    return cooperative;
}

2 references
public List<Cooperative> ReadActive()
{
    return _context.Cooperative.AsNoTracking().Where(c => c.IsActive == true).ToList();
}
```

Kuva 24. *Repository*-luokan metodeja

EF Core Power Tools on .NET-ohjelmistokehykseen kehitetty takaisinmallinnustyökalu. Työkalulla voidaan luoda .NET-projektiin luokat, kontekstitiedosto ja kartoitukset SQL-tietokannalle takaisinmallinnuksella. Työkalussa on helppokäyttöinen UI, jossa voidaan konfiguroida takaisinmallinnuksen asetukset. (Jensen 2022.) Tietokannan tauluista mallinnetut luokat mallinnettiin käyttäen hyväksi EF Core Power Tools -työkalua. Kuvassa 25 on yksi työkalun backendiin mallintamista luokista. Tällä tekniikalla varmistettiin, että tietokantaan tallennettava data on oikeassa muodossa.



Kuva 25. Tietokannan *Book*-taulu ja takaisinmallinnuksella luotu *Book*-luokka

4.4.3 SPA-sovellus

Opintojenseurantatyökalun käyttöliittymä eli sovelluksen frontend toteutettiin SPA:na Angular-ohjelmistokehyksellä. SPA:n toimintalogiikka toteutuu Angularissa reitittimien (engl. *router*) ja komponenttien avulla. Komponentit koostuvat yhdessä toimivista HTML-, CSS- ja TypeScript-tiedostoista. Reititin vaihtaa sovelluksen näkymää näyttämällä käyttäjän toimintojen perusteella valittua komponenttia.

Suunnitteluvaiheessa luodun prototyypin pohjalta toteutettiin valmentajien puoli, jonka toiminta perustuu navigointipalkin ja reitittimen sisältävään *CoachHome*-komponenttiin. Navigointipalkin vieressä on alue näkymille, joita reititin vaihtaa käyttäjän painaessa jotain navigointipalkin painiketta. *CoachHome*-komponentin navigointipalkki (*coach-sidenav*) ja reititin (*router-outlet*) näkyvät kuvassa 26 HTML-tiedostossa.

```
<div class="page-container">
  <section class="coach-sidenav">
    <img src="../../../../../assets/img/coach%20sidenav.png" class="sidenav-image" alt="">
    <mat-icon class="account-icon">account_circle</mat-icon>
    <div class="account-text">{{currentCoach.email}}</div>
    <div class="sidenav-buttons-container">
      <button mat-button class="sidenav-button" [routerLink]="['./settings']">
        <mat-icon class="sidenav-icon">settings</mat-icon>
        <span class="font">Asetukset</span>
      </button>
      <mat-divider></mat-divider>
      <button mat-button class="sidenav-button" [routerLink]="['./cooperatives']">
        <mat-icon class="sidenav-icon">school</mat-icon>
        <span class="font">Osuuskunnat</span>
      </button>
      <button mat-button class="sidenav-button" [routerLink]="['./books']">
        <mat-icon class="sidenav-icon">book</mat-icon>
        <span class="font">Kirjalista</span>
      </button>
      <button mat-button class="sidenav-logout-button" (click)="logout()">
        <mat-icon class="sidenav-icon">logout</mat-icon>
        <span class="font">Kirjaudu ulos</span>
      </button>
    </div>
  </section>
  <main class="content-container">
    <router-outlet></router-outlet>
  </main>
</div>
```

Kuva 26. Navigointipalkki, painikkeet ja reititin HTML-koodissa

Prototyypissä määritetty muotoilu jokaiselle näkymälle toteutettiin käyttämällä CSS:ää yhdessä HTML:n kanssa. CSS-tiedostoon (Kuva 27) luotiin jokaista sivulla näkyvää elementtiä varten oma CSS-luokkansa, johon kirjoitettiin elementille määritettävät tyylit. Usean samankaltaisen elementin muotoilussa käytettiin samaa CSS-luokkaa, mikä vähensi kirjoitettavan koodin toistoa (kuvan 26 koodissa kaikki navigointipalkin painikkeet käyttävät samaa *sidenav-button* -luokkaa). Kuvassa 27 näkyy myös responsiivisuuden mahdollistavaa CSS-koodia. Mediakyselyiden (engl. *media query*) avulla sovellus tunnistaa käytettävän laitteen näytön resoluution. CSS-tiedostoon määritettiin näytön leveydelle pysäytyspisteitä (engl. *breakpoint*), joihin kirjoitettiin erilaista CSS-koodia resoluution mukaan. Esimerkiksi navigintipalkki päätettiin piilottaa kokonaan alle 450 pikseliä leveiltä näytöiltä käytön helpottamiseksi.

```css
.account-text{
  color: #FFFFFF;
  position: relative;
  top: -80px;
  margin-bottom: -40px;
}

.sidenav-buttons-container {
  flex: 1;
  width: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
}

.sidenav-button {
  background-color: transparent;
  border: none;
  margin: 10px;
  width: 90%;
  display: flex;
  flex-direction: row;
  align-items: center;
}
```

```css
@media only screen and (max-width: 1600px){
  .account-icon{
    transform: scale(4);
    position: relative;
    top: -120px;
  }
}

@media only screen and (max-width: 1350px){
  .account-icon{
    transform: scale(3);
    position: relative;
    top: -110px;
  }
}

@media only screen and (max-width: 450px) {
  .coach-sidenav{
    width: 0;
    min-width: 0;
    visibility: hidden;
  }
}

@media only screen and (max-height: 790px){
  .coach-sidenav{
    height: 93vh;
  }
```

Kuva 27. CSS-luokan koodia

Valmentajien puolen käyttöliittymän koodi jakautuu eri komponentteihin ja luokkiin SoC:n periaatteiden mukaisesti. Jokaisella komponentilla ja luokalla on omat tehtävänsä, esimerkiksi osuuskuntien näyttäminen listassa tai kutsun lähettäminen backendille. Kuvassa 28 on osa *CooperativeList*-komponentin TypeScript-luokasta, jossa näkyy luokan muuttujat, konstruktori sekä kaksi funktiota. *CooperativeList*-komponentin luokan konstruktorissa injektoitiin eri *service*-luokkia, joiden funktioita komponentti voi kutsua: esimerkiksi *LoadCooperatives*-funktiossa kutsutaan *CooperativeService*-luokan *GetCooperatives*-funktiota.

```
export class CooperativeListComponent implements OnInit {
  currentCoach: User;
  cooperatives: Cooperative[];
  cooperativeId: string;

  constructor(private cooperativeService: CooperativeService, private authService: AuthService,
              private toolbarService: ToolbarService, private dialog: MatDialog) { }

  ngOnInit(): void {
    this.currentCoach = this.authService.currentUserValue;
    this.loadCooperatives();
  }

  loadCooperatives() {
    this.cooperativeService.getCooperatives().subscribe( next: response => {
      this.cooperatives = response;
    });
  }
```

Kuva 28. Osuuskuntien listauskomponentin TypeScript-koodia

SoC:n mukaan *service*-luokat voidaan jakaa kahteen kerrokseen: *service* ja *http-service*. *Http-service*-luokan tehtävänä on tehdä http-kutsuja luokassa määritettyihin osoitteisiin. *CooperativeHttpService*-luokka (Kuva 29) tekee kutsuja järjestelmän backendiin osuuskuntiin liittyvään päätepisteeseen. Angularissa on kutsujen tekemistä varten oma luokkansa, *HttpClient*. *Service*-luokan tehtävänä on muokata dataa, jonka *http-service* on vastaanottanut, ja jakaa se eteenpäin *service*-luokkaa kutsuvalle taholle.

```
export class CooperativeHttpService {
  url: string;

  constructor(private httpClient: HttpClient) {
    this.url = environment.apiEndPointUrl + '/api/cooperatives/';
  }

  getCooperatives(): Observable<Cooperative[]> {
    return this.httpClient.get(this.url).pipe(map( project: response => {
      return response as Cooperative[];
    }));
  }

  getUsersById(id: string): Observable<User[]> {
    return this.httpClient.get( url: this.url + id + '/students').pipe(map( project: response => {
      return response as User[];
    }));
  }

  create(name: string, password: string): Observable<any> {
    return this.httpClient.post( url: this.url + 'create', body: {name, password}).pipe(map( project: response => {
      return response;
    }));
  }
```

Kuva 29. Osuuskuntien *http-service*-luokan metodeja

Kuvassa 30 näkyy *CooperativeService*-luokan konstruktori, johon on injektoitu *CooperativeHttpService-luokka* ja metodeja, jotka kutsuvat *http-service*-luokkaa. Jos dataa tarvitsee manipuloida sen hakemisen jälkeen, se tapahtuisi *service*-luokassa.

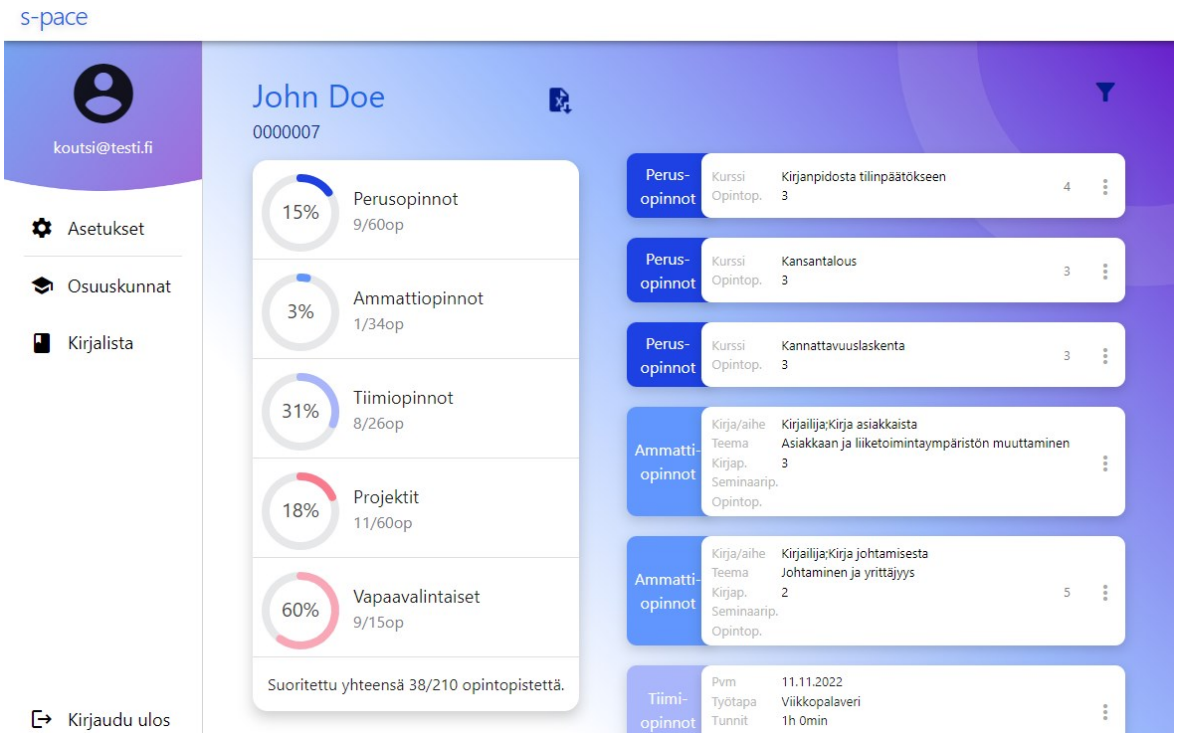
```
export class CooperativeService {

  constructor(private coopHttpService: CooperativeHttpService) { }

  getUsersById(id: string): Observable<User[]> {
    return this.coopHttpService.getUsersById(id);
  }

  getCooperatives(): Observable<Cooperative[]> {
    return this.coopHttpService.getCooperatives();
  }

  create(name: string, password: string): Observable<any> {
    return this.coopHttpService.create(name, password);
  }
```

Kuva 30. Osuuskuntien *service*-luokan metodeja

4.5 Projektin lopputulos

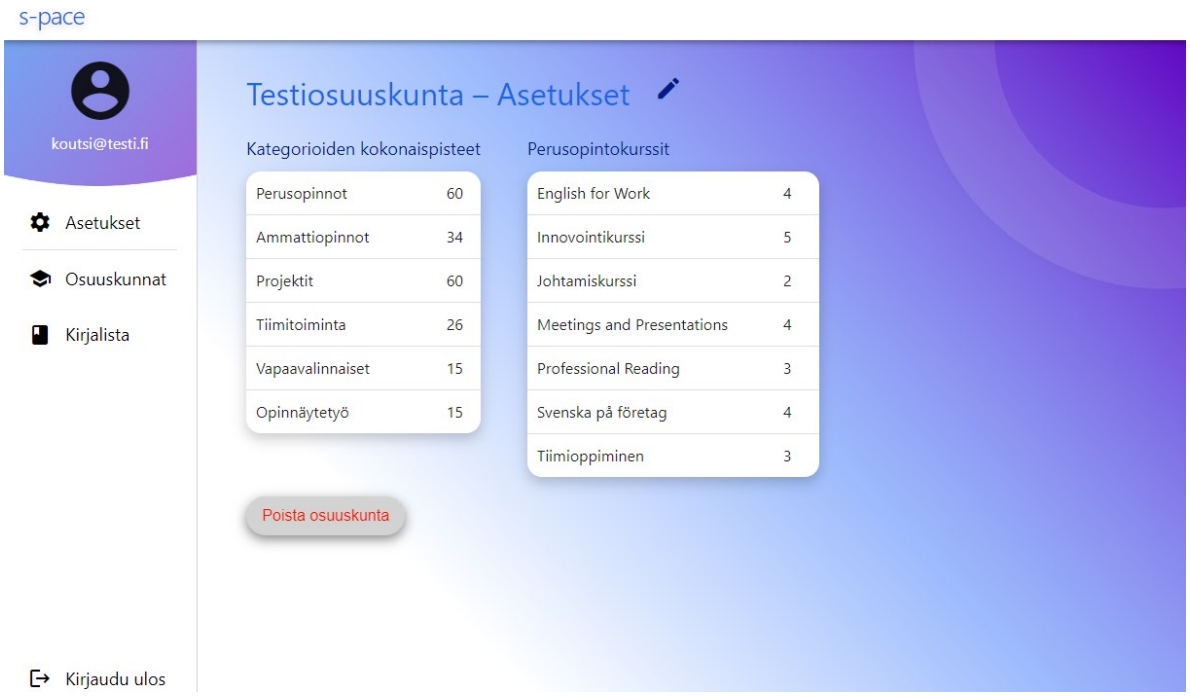
Projektissa toteutettiin kaikki vaatimuslistaan valitut ominaisuudet, jotka on listattu luvussa 4.1. Tärkeimpänä vaatimuksena oli valmentajan puolen toimiva käyttöliittymä. Toteutettu käyttöliittymä vastaa pitkälti suunnitteluvaiheessa luotua prototyyppiä, mutta lopulliseen käyttöliittymään tehtiin vielä joitain muutoksia, koska ne koettiin alkuperäistä suunnitelmaa paremmiksi. Suunnitteluvaiheeseen panostaminen nopeutti toteutusvaiheen etenemistä, kun kehitysryhmälle oli selvää, mitä lähdettiin tekemään.

Kuvassa 31 on valmentajan puolen käyttöliittymään toteutettu opiskelijan tarkastelunäkymä, jossa näkyy käyttöliittymän peruselementtien lisäksi opiskelijan edistyminen ja kirjaukset kategorioittain sekä Exceliin vienti- ja kirjausten suodattamispainikkeet. Kyseiseen näkymään navigoidaan valmentajan valitseman osuuskunnan näkymän kautta. Valmentaja voi muokata, poistaa tai arvostella opiskelijan kirjauksia. Arvostelu ja muut muutokset tulevat reaaliaikaisesti näkyviin kirjaukseen sekä valmentajan että opiskelijan puolilla.



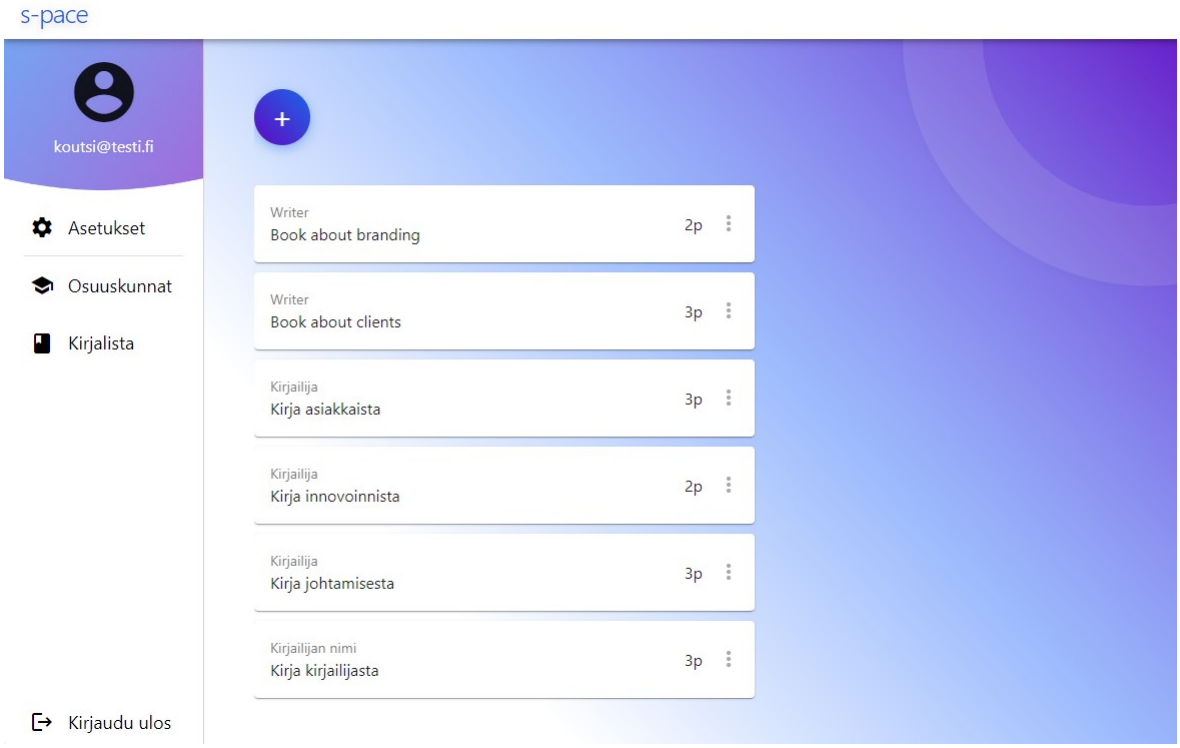
Kuva 31. Toteutettu valmentajan käyttöliittymä

Kuvassa 32 on projektissa toteutettu Osuuskunnan asetukset -näkymä, johon navigoidaan osuuskuntanäkymästä. Asetuksissa on listat osuuskunnan kategorioiden kokonaispisteistä sekä osuuskuntakohtaisista perusopintokursseista. Valmentaja voi muokata molempia listoja, jos esimerkiksi perusopintoihin tulee uusia kursseja. Valmentajan määrittämät asetukset näkyvät reaaliaikaisesti myös opiskelijan puolella.



Kuva 32. Osuuskunnan asetukset

Projektin aikana toteutettiin myös lisäominaisuuksia, kuten kirjalista (Kuva 33), johon valmentaja voi +-painikkeesta lisätä uusia kirjoja. Kirjalista päivittyy reaaliaikaisesti opiskelijoille, jotka voivat valita lukemansa kirjan listasta uutta kirjausta tehdessään. Mikäli kirjaa ei ole vielä lisätty kirjalistaan, opiskelija voi luoda kirjauksen manuaalisesti. Kirjalistan kirjoihin on tallennettu kaikki tarvittava data kirjauksen tekemistä varten.



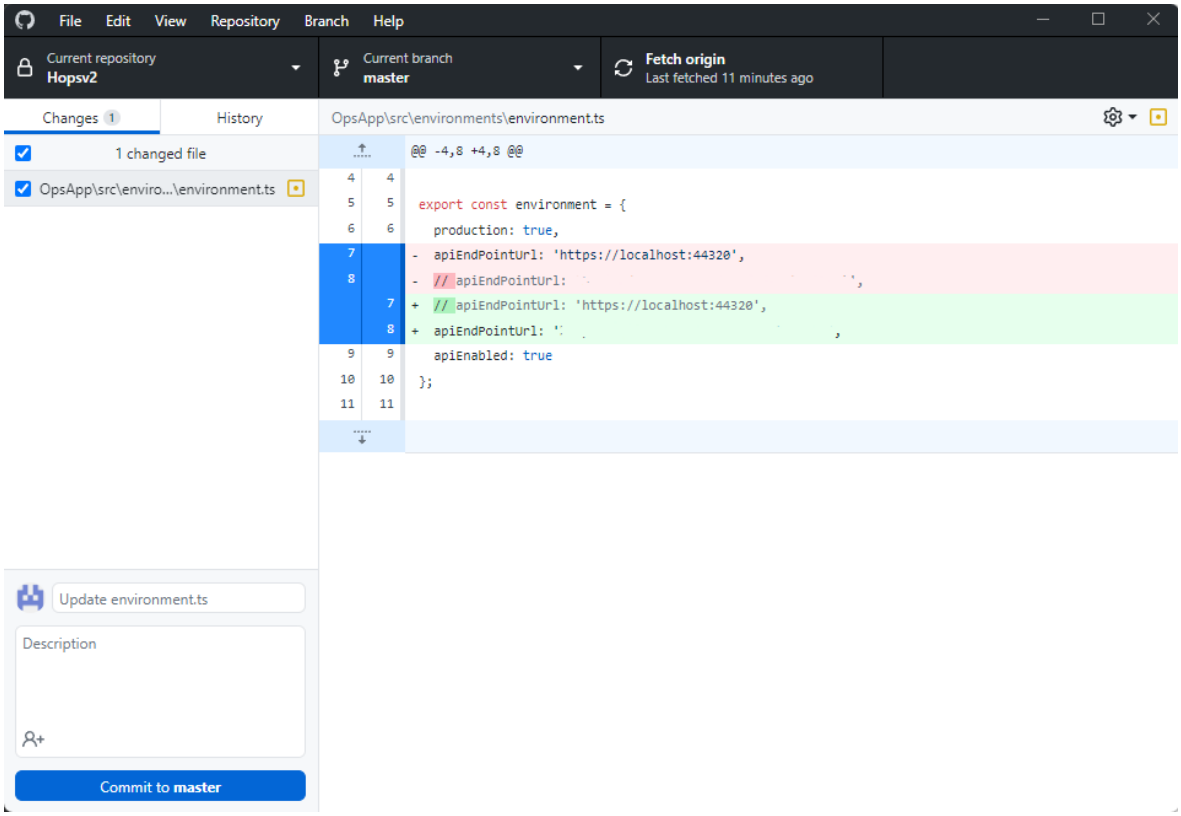
Kuva 33. Toteutettu kirjalistaominaisuus

Näiden ominaisuuksien lisäksi sovellukseen tehtiin tarvittavat virheenkorjaukset, eli korjattiin kaikki aiemmin havaitut visuaaliset ja toiminnalliset virheet. Lisäksi opiskelijan puolen ulkoasua kohennettiin ja yhtenäistettiin esimerkiksi vaihtamalla mobiiliversion taustakuva ja muotoilemalla painikkeita valmentajan puolta vastaaviksi.

4.6 Versionhallinta

Projektin versionhallintaan käytettiin Git:iä ja versioiden tietovarastona GitHubia. Projektissa hyödynnettiin GitHub Desktopia, eli GitHubin luomaa graafisella käyttöliittymällä varustettua työpöytäsovellusta, helpottamaan ohjelmistokehitysryhmän versionhallinnan käyttöä. Kuvassa 34 näkyy GitHub Desktopin käyttöliittymä, jossa näkyy koodiin tehdyt muutokset. Käyttöliittymästä voi muun muassa vaihtaa käytettävää repositoriota ja sen haaraa. Muutokset tallennettiin (engl. *commit*) repositorion aktiiviseen haaraan (engl. *branch*), jossa ne säilyivät erillään repositorion päähaarasta (engl. *master*). Kun koko työryhmä hyväksyi tiettyyn haaraan tehdyt muutokset, ne voitiin liittää päähaaraan. Tämän jälkeen muut

työryhmän jäsenet hakivat (engl. *fetch*) päähaaran uusimman version omalle koneelleen, jolloin uusia muutoksia pystyttiin aina tekemään mahdollisimman ajankohtaiseen versioon.



Kuva 34. GitHub Desktopin näkymä

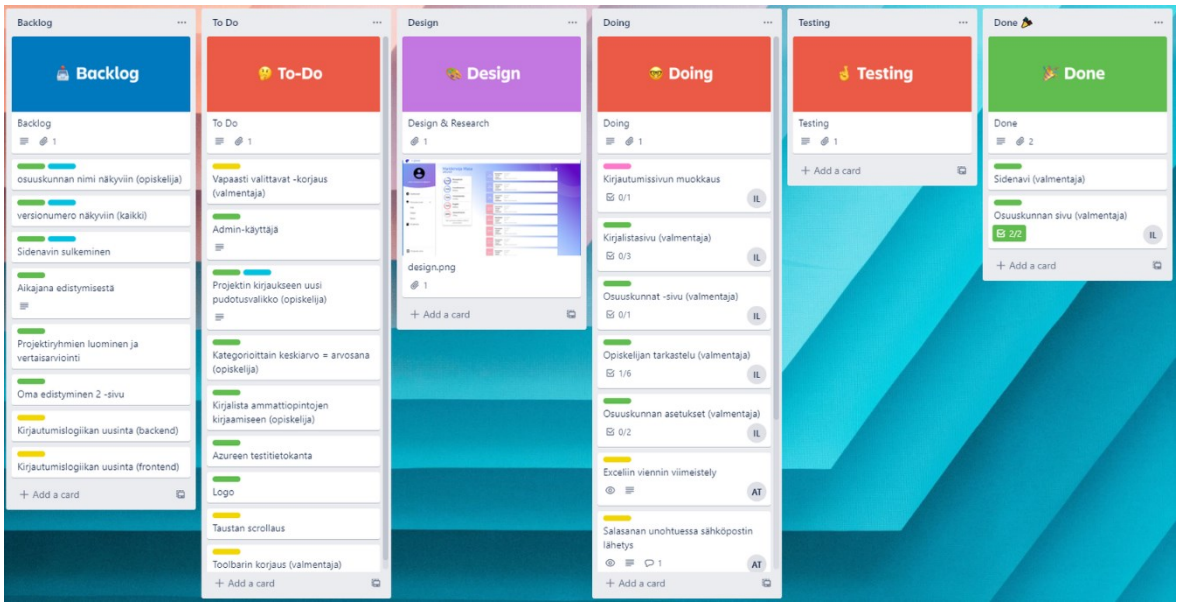
Git oli työryhmälle välttämätön työkalu projektia tehdessä, koska eri henkilöt työstivät päällekkäin eri ominaisuuksia. Jokaista ominaisuutta ja korjausta varten luotiin omat haarat, mikä mahdollisti niiden samanaikaisen työstämisen ja näin ollen nopeutti ja sujuvoitti projektin etenemistä. Ilman Git:iä kehitystyö olisi ollut hidasta ja vanhanaikaista, kun järjestelmään tehdyt muutokset olisi pitänyt käsitellä manuaalisesti.

4.7 Projektinhallinta

Projektinhallinnassa käytettiin tehtävienhallintaan ja viestintään keskittyvää Trello-työkalua, koska se oli työryhmän aiemman kokemuksen perusteella hyvin projektiin soveltuva työkalu ja sen käyttö oli ennalta tuttua. Trellon avulla tehtävienhallinta oli vaivatonta, sillä tehtäväkortit oli helppo siirtää sarakkeesta toiseen ja niihin pystyi lisäämään kuvia, tarkistuslistoja ja värikoodauksia.

Projektin Trello-taulu (Kuva 35) koostui sarakkeista, jotka kertovat sovelluksen vaatimuksien pohjalta muodostettujen tehtävien tilat. Backlog-sarakkeeseen listattiin aluksi kaikki projektiin liittyvät tehtävät, joista To-Do-sarakkeeseen valittiin kaikki ne tehtävät, jotka

projektin aikana aiottiin toteuttaa. Design-sarake oli niille tehtäville, jotka olivat suunnitteluvaiheessa, Doing-sarake kehitysvaiheessa oleville ja Testing-sarake testauksessa oleville tehtäville. Done-sarakkeeseen siirrettiin valmiiksi saadut tehtävät.



Kuva 35. Trello-taulu

Trello-taulun avulla työryhmän oli helppo seurata, missä vaiheessa kunkin tehtävän työstäminen oli. Tehtäviin määritettiin lisätietoja, jolloin työryhmän kesken ei syntynyt epäselvyyksiä tehtävien tarkemmasta sisällöstä. Trello-taulua hyödynnettiin myös silloin, kun valmentajat halusivat kuulla projektin etenemisestä ajankohtaisesti. Backlogiin kirjattiin ylös valmentajien uudet toiveet, jotka projektin aikana syntyivät, joten samaa backlogia voidaan hyödyntää sovelluksen tulevissa jatkokehitysprojekteissa.

4.8 Testaus

Projektin aikana kehitysryhmä testasi yksittäisiä ohjelmistokoodin osia koodin kirjoitusvaiheessa ja järjestelmän toiminnallisuutta aina yksittäisen ominaisuuden valmistuttua. Jokaista ominaisuutta varten kehitettiin käyttötapauksia, ja järjestelmän testaus suoritettiin mustalaatikkotestauksena (engl. *black-box testing*) käyttötapausten avulla. Ohjelmistoon ei kirjoitettu yksikkötestejä tai järjestelmän *end-to-end*-testejä ajan säästämiseksi. Siksi testaus ei ollut niin kattavaa kuin työryhmä olisi halunnut, mutta tarkempia testejä aiotaan kirjoittaa myöhemmin sovelluksen jatkokehityksessä.

Järjestelmän backendin toiminnallisuuksia testattiin Postmanilla. Postman on API-rajapintojen testaukseen kehitetty työkalu, jolla voi manuaalisesti lähettää http-kutsuja API-rajapintojen testausta varten. Jokaisen backendin uuden ominaisuuden oikeanlainen toiminta

varmistettiin Postmanilla testaamalla, ennen kuin ominaisuus toteutettiin frontendissä. Ominaisuuksien toiminnan varmistus vähensi työmäärää frontendiä kehittäessä, kun tiedettiin varmuudella, mitä dataa frontendistä pitää lähettää backendille, ja että ominaisuudet varmasti toimivat backendissä.

Projektin lopuksi testiryhmä testasi projektin aikana toteutettuja ominaisuuksia sekä valmentajan ja opiskelijan puolten yhteensopivuutta Azure-pilvipalvelussa pyörivässä testiversiossa. Testiryhmä koostui markkinointiosuuskunta Volan markkinointiopiskelijoista ja valmentajasta.

Testipäivänä valmentajalle luotiin käyttäjätunnus järjestelmään, minkä jälkeen hän pääsi lisäämään siihen uuden osuuskunnan. Osuuskunnan luomisen jälkeen opiskelijat pääsivät rekisteröitymään syöttämällä omat tietonsa ja juuri luodun osuuskunnan salasanan. Rekisteröidyttyään opiskelijoita kehotettiin tekemään testikirjauksia, jotta päivitetyn valmentajan näkymän toimintaa päästiin testaamaan.

Testaus sujui odotetulla tavalla: opiskelijat pääsivät liittymään valmentajan luomaan osuuskuntaan, opiskelijoiden kirjausten tekeminen onnistui, kirjaukset päivittyivät reaaliajassa valmentajan näkymään ja koko osuuskunta pystyi valmentajan johdolla käymään läpi toistensa kirjauksia. Ainoastaan rekisteröitymisvaiheessa järjestelmä ei toiminut niin kuin odotettiin: Azuressa pyörivä testiversio ei pystynyt vastaamaan kaikkiin noin 30 samanaikaisen käyttäjän rekisteröitymiskutsuihin. Tämän vuoksi sovellus jäi osalla käyttäjistä jumittamaan, ja joidenkin rekisteröityminen ei onnistunut heti ensimmäisellä yrityksellä. Muita ongelmia testauksessa ei tullut ilmi.

Projektin jälkeen sovellus jäi Volalle pidempään testikäyttöön, koska he olivat tyytyväisiä sovelluksen toimintaan ja kokivat sen huomattavasti Excel-taulukkoa paremmaksi tavaksi tehdä opintokirjauksia ja seurata opintojensa etenemistä. Vaikka sovellusta ei vielä ole virallisesti julkaistu, Vola käyttää sitä sen oikeassa käyttötarkoituksessa: Opiskelijat tekevät ajankohtaisesti opintojaan vastaavia kirjauksia, eikä testikirjauksia. Valmentaja seuraa opiskelijoidensa opintojen etenemistä ja lisää arvosanoja opiskelijoiden kirjauksiin. Osuuskuntalaiset käyvät tiimipalavereissa yhdessä läpi kunkin opiskelijan viimeaikaiset kirjaukset valmentajan puolen käyttöliittymää hyödyntäen.

5 Yhteenveto ja pohdinta

Opinnäytetyössä haettiin vastausta siihen, miten luodaan käyttäjäystävällinen ja moderni web-sovellus, mitkä teknologiat sen kehitykseen kannattaa valita, ja mitä käyttöliittymän suunnittelussa tulee ottaa huomioon, kun pyrkimyksenä on mahdollisimman hyvä käyttäjäkokemus. Nämä kysymykset ohjasivat opinnäytetyön toiminnallisessa osassa toteutettavan projektin työstämistä.

Opinnäytetyön toiminnallisen osan tavoitteena oli toteuttaa järjestelmäpäivitys opintojenseurantatyökaluun pyrkien noudattamaan teoriaosassa läpikäytyjä modernin web-ohjelmoinnin periaatteita. Työkalun kehityksessä tavoitteena oli priorisoida loppukäyttäjien, eli markkinointiopiskelijoiden ja valmentajien, tarpeita ja toiveita, joten suunnittelussa ja toteutuksessa kiinnitettiin erityistä huomiota käyttäjäkokemukseen ja käyttäjäkeskeiseen suunnitteluun tekemällä tiivistä yhteistyötä loppukäyttäjien kanssa.

Päivityksen tekemisessä käytetyt teknologiat valittiin sen perusteella, että samoja teknologioita käytettiin järjestelmän kehityksessä aiemminkin, eikä niiden vaihtamiseen nähty tarvetta. Järjestelmän frontend toteutettiin yksisivuisena sovelluksena (SPA) Angular-ohjelmistokehyksellä. Backend toteutettiin REST API -rajapintana ASP.NET-ohjelmistokehyksellä. Tietokanta pidettiin Microsoft SQL -relaatiotietokantana, mutta alkuperäisen tietokannan rakenteeseen tehtiin projektin aikana muutoksia. Teknologiat haluttiin pitää samoina myös siksi, että jo alun perin järjestelmän haluttiin noudattavan modernin web-ohjelmoinnin periaatteita – sekä SPA että REST edustavat modernia web-teknologiaa.

Päivityksen viimeistelemä opintojenseurantatyökalu vastaa osuuskuntien opiskelijoiden alkuperäiseen tarpeeseen nopeasta ja helppokäyttöisestä järjestelmästä, mutta päivityksen myötä myös valmentajien tarpeeseen on vastattu. Koska opinnäytetyön toimeksiantajat olivat tyytyväisiä projektin tuloksena syntyneeseen päivitykseen, ja sen seurauksena opintojenseurantatyökalu oli valmis käyttöönotettavaksi, se otettiin ensimmäistä kertaa oikeaan käyttöön yhdellä Saitemian osuuskunnista. Jatkossa työkalu tulee käyttöön myös Saitemian uusille osuuskunnille sitä mukaa, kun niitä perustetaan. Loppukäyttäjien halusta ottaa sovellus käyttöön voidaan päätellä, että käyttäjäkeskeisestä suunnittelusta oli suuri hyöty loppukäyttäjää miellyttävän käyttäjäkokemuksen toteuttamisessa, ja että työn tuloksena syntynyt sovellus tehostaa käyttäjiensä arkea. Loppukäyttäjien tyytyväisyyden perusteella opinnäytetyön toiminnallisen osan tavoitteet voidaan katsoa saavutetuiksi.

Opinnäytetyössä tuotettua designia ja sen elementtejä voidaan hyödyntää sovelluksen jatkokehityksessä. Pienellä jatkokehityksellä opintojenseurantatyökalua voisi mukauttaa eri oppilaitoksiin sopivaksi, ja laajemmalla jatkokehityksellä sovellusta voisi hyödyntää

monessa eri tarkoituksessa eri toimialoilla. Jatkokehitystä ja laajempaa asiakassegmenttiä ajatellen järjestelmä kaipaa muutoksia etenkin tietokannan rakenteeseen ja mahdollisuutta esimerkiksi opintokokonaisuuksien laajempaan muokkaukseen.

Jatkokehitystä ja tuotteen kaupallistamista varten sovelluksen ympärille perustetaan yritys, jonka kautta toiminnan jatkuvuus on mahdollista. Yrityksen tavoitteena on kehittää tuotteesta ensin ratkaisu osuuskuntamuotoisen opiskelun seurantaan useissa oppilaitoksissa Suomessa ja mahdollisesti myös ulkomailla. Myöhemmin tuotetta voitaisi kehittää myös muiden opintomuotojen opintojen seurantaan tai esimerkiksi yritysten sisäiseen työajan ja työtehtävien tai projektien etenemisen seurantaan.

Teknisestä näkökulmasta tuotetta voisi jatkokehittää käytettäväksi entistä useammilla laitteilla, esimerkiksi älykelloilla, joilla opiskelijat voisivat nopeasti napauttaa ajastimen päälle alkaessaan opiskelemaan. Lisäksi sovelluksen toteuttaminen natiivisovelluksena tai progressiivisena web-sovelluksena (engl. *Progressive Web Application*, PWA) lisäisi sen käyttömukavuutta ja -mahdollisuuksia, kun käyttäjän ei tarvitse avata sovellusta aina selaimen kautta. Samalla järjestelmän käytettävyys laajenisi, kun sovellusta voisi käyttää myös ilman internetyhteyttä, jolloin sitä voisi hyödyntää myös esimerkiksi opiskelijavaihdon aikana, kun käyttäjällä ei välttämättä ole jatkuvaa pääsyä internettiin. Laaja käytettävyys on tärkeää niin käyttäjäystävällisyyden kuin tuotteen levinneisyyden ja yrityksen kasvun näkökulmista.

Responsiivisuuden tärkeys korostuu jatkokehityksessä etenkin, jos tuotetta kehitettäisiin käytettäväksi muillakin laitteilla kuin älypuhelimilla ja tietokoneilla. Erilaisia laitteita on nyt jo laaja kirjo, ja niiden ominaisuudet vaihtelevat suurestikin eri laitteiden välillä. Teknologiamaailman ollessa jatkuvassa murroksessa erilaisia laitteita ja käyttöympäristöjä kehitetään koko ajan lisää. Kun responsiivisuus on otettu huomioon jo digitaalisen tuotteen suunnitteluvaiheessa ja tuotteen kehityksen alkutaipaleella, sitä on helpompi toteuttaa myös tuotteen tulevaisuudessa.

Käyttäjäkeskeisen suunnittelun periaatteiden ymmärtäminen selkeytti sekä suunnittelu- että toteutusvaiheita, kun käyttäjän tarpeisiin vastaaminen oli alusta asti korkein prioriteetti. Loppukäyttäjien kuuleminen ohjasi sovelluksen kehitystä oikeaan suuntaan, eli selkeän ja miellyttävän käyttäjäkokemuksen tarjoamiseen käyttöliittymän kautta, koska kehitysryhmän ei tarvinnut arvailla käyttäjien tarpeita.

Sekä opinnäytetyön teoriaosan että toiminnallisessa osassa kerätyn kokemuksen perusteella moderni käyttäjäkokemus syntyy ajattomasta ja intuitiivisesta designista. Sen saavuttamiseksi ei ole tiettyjä ohjenuoria, jotka kattaisivat kaikenlaiset käyttäjäkokemukset eri kohderyhmissä, vaan jokainen käyttäjäkokemus on suunniteltava yksityiskohtaisesti juuri tietylle kohderyhmälle haluttuun käyttötarkoitukseen. Modernin käyttäjäkokemuksen

suunnittelussa huomioidaan mielikuvat, jotka sovelluksen halutaan herättävän kohderyhmässä, jotta käyttäjäkokemus palvelee loppukäyttäjää parhaalla mahdollisella tavalla. Tämän takia kohderyhmän tutkimus on oleellinen vaihe käyttäjäkeskeistä suunnittelua, koska käyttäjän eri ominaisuudet, kuten ikä ja kulttuuri, voivat vaikuttaa suunnitteluvalintojen herättämiin mielikuviin käyttäjässä. Myös erilaiset rajoitteet, kuten värisokeus, on tärkeää huomioida käyttöliittymän suunnittelussa, koska ne vaikuttavat tuotteen saavutettavuuteen.

Laajassa mittakaavassa mitä enemmän positiivisia digitaalisia käyttäjäkokemuksia syntyy, sitä vaivattomampaa palveluiden käyttäminen on, ja ihmisillä on mahdollisuus keskittyä vain olennaiseen. Kehitys ei ikinä lopu: vaikka käytössä olisi valmiiksi hyvä ja toimiva järjestelmä, sen käyttäminen voisi aina olla miellyttävämpää, nopeampaa tai ketterämpää, kuten opintojenseurantatyökalun tapauksessa. Palveluiden digitalisoimista viedään koko ajan pidemmälle, ja vielä löytyy paljon erilaisia palveluita, jotka eivät ole digitaalisessa muodossa. Sovelluskehittäjillä on siis jatkuva tarve kehittää uusia tai aiempaa paremmin toimivia digitaalisia järjestelmiä ja miellyttävämpiä käyttäjäkokemuksia, jotta itse asiaan – esimerkiksi työntekoon tai opiskeluun – käytettävä aika voidaan maksimoida.

Lähteet

Angular. 2022a. Introduction to the Angular Docs. Viitattu 27.9.2022. Saatavissa https://angular.io/docs

Angular. 2022b. What is Angular? Viitattu 26.10.2022. Saatavissa https://angular.io/guide/what-is-angular

Ansari, T. 2022. Stack Overflow survey: JavaScript most used programming language for 10th year in a row. Analytics India Mag. Viitattu 27.9.2022. Saatavissa https://analyticsindiamag.com/stack-overflow-survey-javascript-most-used-programming-language-for-10th-year-in-a-row/

Association for Project Management. What is project management? Viitattu 16.11.2022. Saatavissa https://www.apm.org.uk/resources/what-is-project-management/

Atlassian a. What is version control? Viitattu 16.11.2022. Saatavissa https://www.atlassian.com/git/tutorials/what-is-version-control

Atlassian b. What is Git. Viitattu 16.11.2022. Saatavissa https://www.atlassian.com/git/tutorials/what-is-git

Beaird, J. & George, J. 2014. The Principles of Beautiful Web Design. 3. painos. SitePoint Pty Ltd.

Beaulieu, A. 2020. Learning SQL, 3rd Edition. O'Reilly Media, Inc.

Doglio, F. 2018. REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development. Apress.

eLab. 2022. Opintosuunnitelma HOPS. Viitattu 10.3.2022. Saatavissa https://elab.lab.fi/fi/opintojen-suorittaminen/opintojen-suunnittelu/opintosuunnitelma-hops

Freeman, A. 2020. Pro Angular 9: Build Powerful and Dynamic Web Apps. E-kirja. Appress.

Github. 2022. Angular. Viitattu 27.9.2022. Saatavissa https://github.com/angular/angular

Goel, A. 2022. 10 Best Web Development Frameworks. Hackr.io. Viitattu 21.11.2022. Saatavissa https://hackr.io/blog/web-development-frameworks

Graafinen. 2015. Yleistä typografiasta. Viitattu 25.10.2022. Saatavissa https://www.graafinen.com/suunnittelu/typografia/yleista-typografiasta/

Grant, W. 2022. 101 UX Principles. Packt Publishing.

JavaTpoint. Software Project Management. Viitattu 16.11.2022. Saatavissa https://www.javatpoint.com/software-project-management

Jensen, E. E. 2022. Reverse Engineering. Viitattu 21.11.2022. Saatavissa https://github.com/ErikEJ/EFCorePowerTools/wiki/Reverse-Engineering

Kenzie Academy. 2020. Front End vs. Back End: What's the Difference? Viitattu 9.10.2022. Saatavissa https://kenzie.snhu.edu/blog/front-end-vs-back-end-whats-the-difference/

Maioli, L. 2018. Fixing Bad UX Designs. Packt Publishing.

Martin, R. C. 2012. The Clean Architecture. Viitattu 14.3.2022. Saatavissa https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Martin, R. C. 2017. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson.

Material Design. 2014. Errors. Viitattu 2.11.2022. Saatavissa https://m1.material.io/patterns/errors.html#errors-user-input-errors

Material Design. 2018a. States. Viitattu 1.11.2022. Saatavissa https://m2.material.io/design/interaction/states.html

Material Design 2018b. Introduction. Viitattu 23.10.2022. Saatavissa https://m2.material.io/design/introduction

McKay, E. N. 2013. UI is Communication. Morgan Kaufmann.

Microsoft. 2021a. Architectural principles. Viitattu 14.3.2022. Saatavissa https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles

Microsoft. 2021b. Common web application architectures. Viitattu 14.3.2022. Saatavissa https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

Microsoft. 2022. Choose Between Traditional Web Apps and Single Page Apps (SPAs). Viitattu 28.9.2022. Saatavissa https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

.NET. 2022a. What is .NET? Viitattu 10.11.2022. Saatavissa https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

.NET. 2022b. ASP.NET Web APIs. Viitattu 10.11.2022. Saatavissa https://dotnet.microsoft.com/en-us/apps/aspnet/apis

Pulkkanen, A. Projektityön digiopas. Agendium Oy. Viitattu 17.11.2022. Saatavissa https://www.agendium.com/projektinhallinta/miksi-projektityokaluja-tarvitaan

Rae, M. 2020. What is Adobe XD and what is it used for? Adobe. Viitattu 1.11.2022. Saatavissa https://www.adobe.com/products/xd/learn/get-started/what-is-adobe-xd-used-for.html

REST API Tutorial. 2022. What is REST. Viitattu 26.10.2022. Saatavissa https://restfulapi.net/

Richardson, L., Amundsen, M. & Ruby, S. 2013. RESTful Web APIs. O'Reilly Media, Inc.

Ritter, M. & Winterbottom, C. 2017. UX for the Web. Packt Publishing.

Saimaan tiimiyrittäjäakatemia ry. 2022. Viitattu 4.4.2022. Saatavissa https://www.saitemia.fi/

Silk, J. 2021. 10 Important Software Development Technologies You Should Know About. Startechup. Viitattu 21.11.2022. Saatavissa https://www.startechup.com/blog/software-development-technologies/

Tutorialspoint. TypeScript – Classes. Viitattu 20.11.2022. Saatavissa https://www.tutorialspoint.com/typescript/typescript_classes.htm

Uluca. D. 2018. Angular for Enterprise-Ready Web Applications. Second Edition. Packt Publishing.

Usability First. 2015. Introduction to User-Centered Design. Viitattu 18.11.2022. Saatavissa https://www.usabilityfirst.com/about-usability/introduction-to-user-centered-design/#main

Ustwo. 2014. Pixel Perfect Precision Handbook. Viitattu 14.3.2022. Saatavissa https://www.ustwo.com/blog/pixel-perfect-precision-handbook/

Zapanta. K. 2022. Is SQL Worth Learning? 5 Reasons to Learn SQL. Career Karma. Viitattu 22.11.2022. Saatavissa https://careerkarma.com/blog/is-sql-worth-learning/