



Projekti 1

Raportti

Tehnyt:
Iiro Partanen (525720_287358_A)
Nelli Rossi (2102231)

Raportti
31. tammikuuta 2024
22 s.

Turun yliopisto
Tietotekniikan laitos
Tietojenkäsittelytiede
Projekti 1

Sisällys

1	Johdanto	1
2	Suunnittelu	1
2.1	Rakenne	2
2.2	Työvaiheet	3
3	Valmiin projektin kuvaus	4
3.1	Ohjelmisto	4
3.1.1	Frontend	4
3.1.2	Backend	5
3.1.3	Tietokanta	7
3.2	Kehitys	9
3.2.1	Scrum	9
3.3	Suunnitelman toteutuminen	10
4	Projektiin tulokset	11
4.1	Sovellus	11
4.2	Kehitys	12
5	Yhteenvetö	13
6	Liitteet	13

1 Johdanto

Tavoitteenamme oli lähteä luomaan nettisivua, jossa voisimme käyttää kolmannen osapuolen API:ta, JavaScriptia Reactin kanssa, Javaa Spring Bootin kanssa ja MySQL:ää. Päädyimme valitsemaan projektin aiheeksi reseptit, jolloin ideaksi muovautui tarkemmin nettisivu, jolta olisi mahdollista hakea reseptejä, lisätä niitä suosikkeihin ja kalenteriin. Tähän saimme käytettyä kaikkia teknologioita, joita halusimmekin, ja tällaista sivua olisi helppo lähteä laajentamaan tai supistamaan toiminnoltaan tarpeen tullen.

Meille keskeisiä asioita projektissa olivat uusien teknologioiden oppiminen, ohjelmistokehityksen opetteleminen, Scrumin ja TDD:n käyttö. Scrum ja TDD olivat entuudestaan tuttuja käsitteitä, mutta molemmilta puuttui näistä aikaisempi käytännön kokemus. Näiden lisäksi lähdimme myös tavoitteemaan yleisesti ohjelmoijana kehittymistä ja erityisesti parantamaan yhdessä ohjelmoinnin sujuvuutta. Uskomme tämän olevan 5 opintopisteen laajuisen työ.

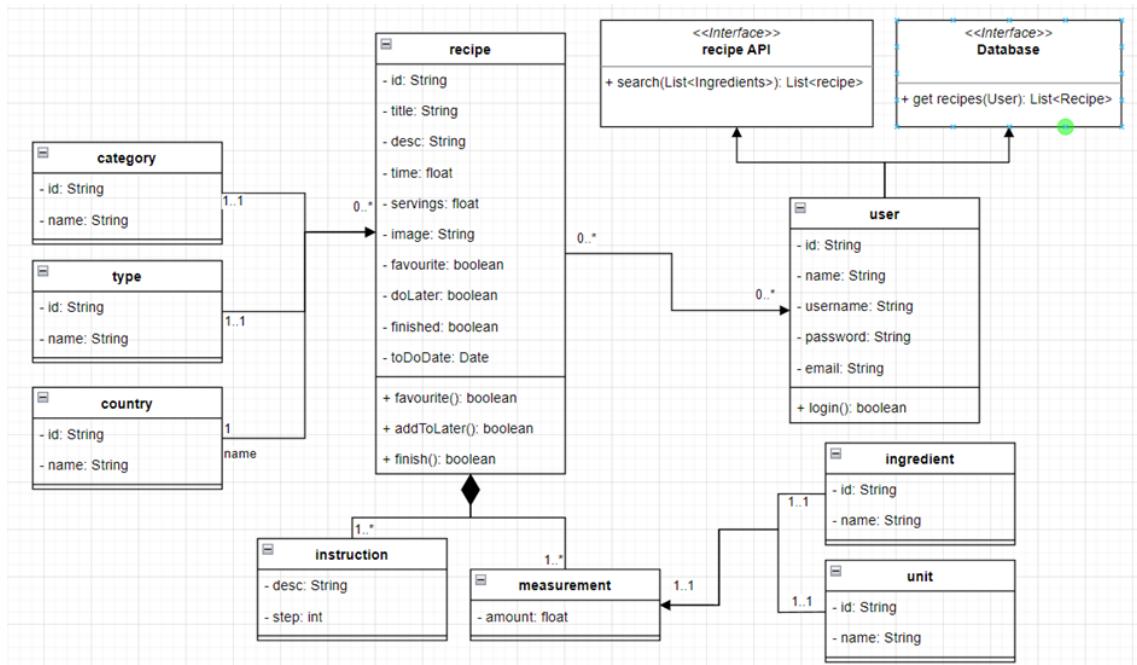
2 Suunnittelu

Aloitimme suunnittelemisen tekemällä projektin frontendin Figmalla, jonka jälkeen suunnittelimme backendin käyttäen sitä mallina. Viimeisenä suunnittelimme tietokannan rakenteen kutsujen perusteella. Kun projektin rakenne oli valmis, aloimme suunnittelemaan itse työnkulkua.

2.1 Rakenne

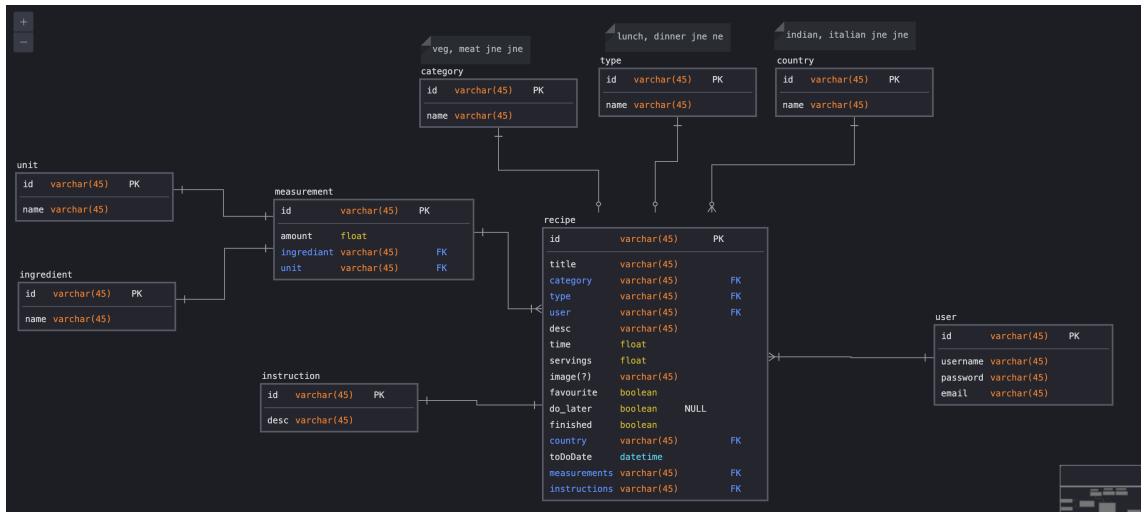
Frontendin suunnittelussa lähdimme ensin visualisoimaan projektin eri sivuja ja niiden toiminnalsuksia. Tavoitteena oli luoda tietokoneelle ja mobiilille kotisivu, hakusivu, kalenterinäkymä, käyttäjänäkymä ja asetukset. Tämän jälkeen lähdimme opettelemaan Figman käyttöä ja luomaan jokaiselle sivulle mockupit, jotka ovat kuvissa 6.4, 6.5, 6.6, 6.7, 6, 6 ja 6.9.

Frontendin suunnitelman valmistuttua siirryimme backendin suunnitteluun ja teimme UML-kaavion projektin rakenteesta, joka on kuvassa 2.1. Tämän lisäksi teimme myös sekvenssikaavioita projektin tärkeimmistä kutsuista, jotka ovat kuvassa 6.3.



Kuva 2.1: Backend tietokanta uml

Lopuksi suunnittelimme vielä backendin perusteella tietokannan rakenteen, joka on kuvassa 2.2.



Kuva 2.2: Backend tietokanta uml

2.2 Työvaiheet

Aloitimme kehityksen frontendin mockupin kehityksestä, jonka jälkeen siirryimme backendin kehitykseen ja lopuksi integroimme nämä yhteen. Siten pystyimme vielä muokkaamaan backendiä, jos frontendin kehityksessä syntyi ongelmia.

Itse kehitykseen käytimme noin viikon mittaisia sprinttejä ja TDD:tä. Suunnitelimme sprintit aina sprintti kerrallaan backlogin avulla ja sprintin suunnittelutapaamisissa sovimme tulevan sprintin implementoitavat ominaisuudet, jaettiin tehtävät ja kirjoitettiin testit valituille ominaisuuksille. Pidimme myös jokaisen päivän alussa nopean tapaamisen, jossa kerroimme edellisenä päivän omasta edistyksestä, ongelmista ja mitä teemme sinä päivänä. Jokaisen sprintin jälkeen pidimme myös lopetustapaamiset, joissa kävimme läpi edellisen sprintin onnistumisia ja kehityskohteita.

Tavoitteenaamme oli saada työ valmiiksi 1.1.2024 mennessä.

3 Valmiin projektin kuvaus

Projektiin aikataulu venyi vajaalla kuukaudella työn vaikeuden takia ja arvointi oli alun perinkin hieman optimistinen. Yhteensä projektiin käytimme noin 400 tuntia, eli 200 tuntia henkilöä kohden, mikä ylitti odotuksemme työmääärästä.

3.1 Ohjelmisto

3.1.1 Frontend

Frontend on jaoteltu komponentteihin, sivuihin, palveluihin, tyyleihin ja testeihin. Näiden lisäksi kansiossa assets ovat kaikki kuvat, joita sivuilla käytetään.

Komponentit, joita käytetään sovelluksessa useaan kertaan löytyvät komponenttikansista. Niistä on pyritty tekemään sellaisia, että niitä olisi mahdollisimman helppo käyttää uudelleen eri sivuilla. Esimerkiksi Ingredients palauttaa listan reseptin ainesosia, ja sitä voidaan hyödyntää kaikissa eri reseptisivuissa, mitä projektissa on. Monimutkaisimmille komponenteille löytyy myös mockit testejä varten.

Projektiin eri sivut ovat jaoteltuna kansioihin pages kansiossa. Jokaisen kansion sisällä on sen nimeä vastaava tiedosto, joka palauttaa jonkin sivuista. Home palauttaa kotisivun, login palauttaa kirjautumisen, personal palauttaa käyttäjälle henkilökohtaisen sivun, recipe palauttaa yksittäisen reseptin sivun, search palauttaa hakusivun, settings palauttaa asetukset ja today palauttaa kalenterista haetun yksittäisen reseptin. Päätiedostossa, jonka nimi on sama kuin kansion, ovat kaikki sivun funkciot ja muihin tiedostoihin on voitu refaktoroida sivun osia, esimerkiksi login kansiossa on CreateNewForm, Login, LoginForm, index ja loginHooks. Sivuelle, joilla täytyy tehdä useampia kutsuja backendiin on tehty Hooks tiedosto, johon on kerätty nämä kutsut. Hooksista kutsutaan palveluita, jotka tekevät backendiin kutsun,

tai vastaavasti palvelua voidaan kutsua suoraan sivun tiedostosta, jos niitä on vain esimerkiksi yksi tai se on yksinkertainen. Hookeissa on myös datan käsittely kutsuja varten.

Services eli palvelut ovat omassa kansiossaan, johon ne ovat jaoteltuna niiden sivujen mukaan, mitä niissä kutsutaan. Näillä palveluiden kutsuilla tehdään kaikki frontendin kutsut backendiin. Esimerkiksi loginServicessä tehdään kutsu backendiin käyttäjän tilin saamiseksi ja uuden tilin luomista varten. Tässä tehdään vain kutsu, eikä käsittää dataa lainkaan.

Tyyleissä käytimme SaSS:ia ja kaikki tyylit löytyvät tiedostosta styles. Tyylit ovat jaoteltuna muutakin rakennetta mukailleen komponenttien kansioon ja sivulle omiin tiedostoihin. Näistä erikseen on erotettu kaikki värit ja fontit, mitä projektissa käytämme. Main.scss:ään on importattu muut tiedostot ja sinne on myös määritelty tyylejä joillekin projektissa yleisille HTML-tageille. Lisäksi sieltä löytyy layoutin tyyllity, johon kuuluvat header, login, content ja footer.

Frontendin testit ovat tests kansiossa jaoteltuna projektin muunkin hengen mukaisesti. Esimerkiksi kirjautumisen testit löytyvät kansiosta login, jossa on erikseen testit CreateNewForm.test, Login.test, LoginForm.test ja LoginHooks.test. Eri tiedostojen testit noudattavat samaa kaavaa, eli ensin niissä testataan renderöityminen, jonka jälkeen testataan funktionalisuudet. TestData kansioon on määritelty dataa testejä varten ja testHelpers sisältää funktioita, jotka tarvittaessa auttavat testeissä.

3.1.2 Backend

Backendistä löytyy kolme tärkeää kansiota, joista ensimmäinen on src/main/java, joka sisältää kaiken koodin. Src/main/resources, joka sisältää konfiguraation ja src/test, joka sisältää testit. Tämän lisäksi on Dockerfile, joka määrittelee sovelluksen kontittamisen ja pystymme deplooyaamaan sovelluksen tämän avulla. Muuten rakenne on

normaalilin mavenin mukainen.

Resources sisältää konfiguraation testeihin ja myös itse sovelluksen käyttöön. Application-properties sisältää tiedot yhteydestä AWS:ään sekä salaiset arvot ja avaimet. Application-test.properties taas sisältää samat arvot, mutta julkisina testejä varten. Data.sql sisältää testitietokannan alustamiseen tarvittavat SQL-kyselyt.

Itse src/main/java.. on jaettu vielä exceptions ja recipe kansioihin. Exceptions kansio sisältää meidän kaikki kustomoidut virheilmoitukset. Recipe kansio sisältää kaiken muun oleellisen koodin.

Koodi on jaoteltu eri kansioihin ja päätiedostoon RecipeApplication.java, jonka ajamalla sovellus käynnistyy. Muuten koodi on jaoteltu suurimmaksi osaksi omien pöytien mukaan. Jokaiselle pöydälle on oma kansio esimerkiksi account kansiossa on Account-pöydän tiedostot. Tämän lisäksi on apiClasses, joka sisältää luokkia joiden avulla pystymme vastaanottamaan API kutsuista dataa. Enums sisältää kaikki ENUM-luokat ja pages sisältää kaikki kutsut, joissa kutsutaan koko sivua. Response sisältää luokat, jotka ovat siinä muodossa, että ne voidaan palauttaa frontendiin sekä funktiot, joilla nämä muunnokset voidaan tehdä automaattisesti. Security sisältää kaikki Spring Securityn tiedostot, eli sieltä löytyy konfiguraatiot ja auktorisointifunktiot.

Kaikki pöytien luokat perustuvat samaan rakenteeseen. Ne sisältävät controllerin, jossa on määritelty endpointit, servicen, jossa suoritetaan kaikki logiikka ja repositoryn, jossa on kutsut tietokantaan. Aina kaikkia näitä ei kuitenkaan ole määritelty, jos niille ei ole tarvetta. Tämän lisäksi kansiot saattavat sisältää luokkia, joiden avulla dataa käsitellään ja näiden funktiot löytyvät JavaDocista. Tämän lisäksi recipe kansiossa on RecipeUtils luokka, jossa on funktiot API:n kutsumiseen.

Pages kansio toimii samalla rakenteella kuin pöytien kansiot, mutta tässä endpointit ovat kokonaisia sivuja eli esimerkiksi kun ladataan personal sivu ensimmäisellä kerralla, niin kutsutaan vain personal page kutsua ja saadaan kaikki oleellinen

data. ApiClasses kansio sisältää luokkia API kutsuihin, jotta pystymme käsittelemään API kutsujen dataa. Response kansio sisältää luokat, joita voi palauttaa frontendiin. Nämä poistavat sensitiivisen datan, esimerkiksi salasanan käyttäjältä tai turhan datan reseptistä. Converters sisältää funktioita, jotka refaktoroivat luokat automaattisesti.

Response sisältää kaiken tarvittavan Spring Securitylle eli tietoturvalle. Se sisältää tiedostoina: ApplicationConfig, joka määrittelee käytetyt funktiot Spring Securitylle, AuthRequestin, joka on kirjautumiskutsu frontendistä, Authorizationin, joka sisältää kaikki funktiot joiden avulla tarkistamme käyttäjän auktoriteetin haluttuun dataan, JwtAuthFilter, joka sisältää filtterin JWT tokenin tarkistamiseen, JwtService, joka sisältää funktioita JWT:n käsittelemiseen ja SecurityConfigin, jossa on CORS ja myös lopullinen filter chain, jota sovellus käyttää.

Testit sijaitsevat test kansiossa, jossa ne on jaettu tietokantapöytien mukaan. Jokaisesta kansiosta löytyy tarvittaessa controllerTest, jossa on testit endpointille ja integrationTest, jossa on integrointitestit kyseiselle kansiolle, repositoryTest, jossa on testit tietokantakutsuille ja serviceTest, jossa on testit logiikalle.

3.1.3 Tietokanta

Tietokanta on MySQL tietokanta, joka on hostattu AWS:ään. Tietokannan rakenne on kuvassa 3.1.

Recipe-pöytä on tietokannan keskeisin pöytä ja se kuvaaa yhtä reseptiä. Resepti saa teksteinä itselleen kuvaksen, linkin kuvaan, linkin alkuperäiseen ja otsikon. Numeroina se saa keston ja annosmäärän. Reseptin TODO-päivä on sql date -formaattia ja kun päivää ei ole valittu, se on null. Tämän lisäksi se saa vielä statuksen, että onko resepti lisätty do later, favourite tai finished listalle. Tätä kuvaa bitti 0, jos ei ja 1, jos on. Lopuksi resepti vielä saa käyttäjän id:n, joka on lisänyt reseptin. Resepti saa myös oman id:n parametrinä, koska pyrimme pitämään resep-

teille samat id:t kuin mitä Spoonacular käyttää. Reseptin loput arvot ovat relaatioita muihin pöytään.

Account-pöydän ainoa linkki on recipe-pöytään, joka on OneToMany relaatio, jolloin kaikki käyttäjän reseptit voidaan hakea hakemalla reseptit, jossa on käyttäjän id. Käyttäjää saa myös tekstinä arvot sähköpostille, nimelle, käyttäjänimelle ja salasanana salattuna. Käyttäjän id on automaattisesti generoitu.

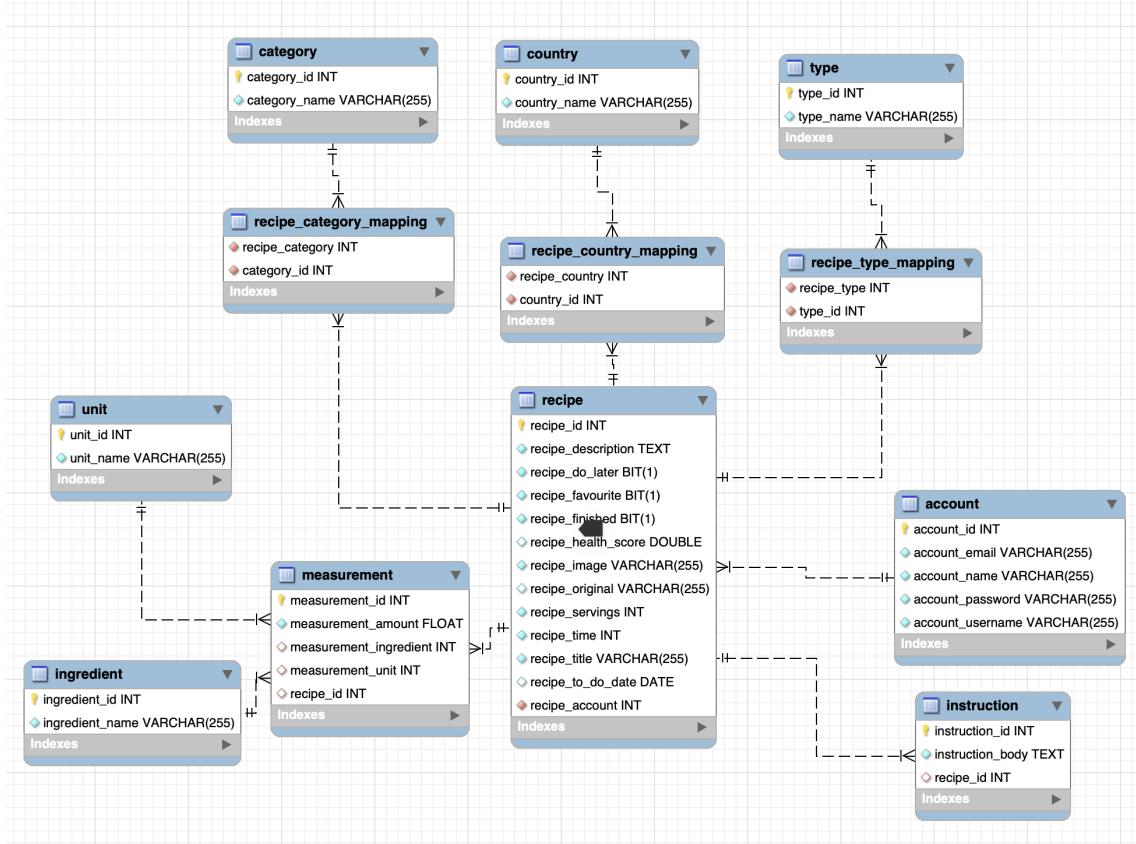
Reseptin mitat saamme relatiolla measurement-pöytään. Tämä on ManyToOne relaatio, jolloin measurement saa arvoksi reseptin id:n, jolle se kuuluu. Yksi measurement-pöydän rivi kuvaa siis yhtä mittausta reseptille ja reseptiä lisätessä luomme aina reseptille kuuluneet mittaukset uusina riveinä. Mittauksiin sisältyvä float-arvona määrä sekä relaatioina yksikkö ja ainesosa. Yksikkö ja ainesosa lisätään vain niiden id:n avulla ja niihin on ManyToOne relaatio. Id generoituu automaattisesti. Reseptin poistaessa measurement poistuu automaattisesti.

Unit- ja ingredient-pöydät sisältävät jo kaikki mahdolliset arvot ja näihin ei lisätä uusia arvoja, vaan aina luodaan relaatio näihin. Relaatiot molemmista ovat OneToMany.

Instruction-pöytä sisältää reseptin ohjeet. Pöytään on ManyToOne relaatio, eli on monta ohjetta yhdelle reseptille. Ohje saa arvoikseen vain ohjeen sisällön ja reseptin id:n, ja nämä luodaan, kun lisätään uusi resepti. Id generoituu automaattisesti ja ohjeet poistuvat automaattisesti reseptin poistaessa.

Lopuksi on vielä category, country, type ja niiden mapping-pöydät. Nämä kaikki kolme pöytää sisältävät jo kaikki mahdolliset arvot ja mapping-pöydät ovat linkki reseptin ja pöytien välillä. Esimerkiksi category-pöytä siltää jo arvoja, ja kun luomme uuden reseptin, jolla on tietyt kategoriat, niin luomme vain uuden rivin mapping pöytään, jossa on reseptin category_id ja valitun kategorian id. Näin pystymme antamaan reseptille monia arvoja, mutta poistaessa emme poista myös alkuperäisiä arvoja, vaan poistamme vain linkin. Kaikki pöydät sisältävät vain id:n ja arvon.

Mapping-pöydän relaatio on kumpiinkin suuntaan ManyToOne.



Kuva 3.1: Backend structure uml

3.2 Kehitys

3.2.1 Scrum

Projektiin yhteydessä käytimme molemmat ensimmäistä kertaa Scrumia ja ottimme käyttöön projektin yhteydessä Jira-alustan, johon saimme kerättyä sprinttien backlogit, retrospektiivit, päivittäiset tapaamiset ja aloitustapamiset. Yhteensä sprinttejä kertyi kahdeksan ja pituudeltaan ne olivat keskimäärin viikon mittaisia. Sprintit olisi voinut jakaa myös niin, että ne olisivat olleet pidempiä, jolloin niitä olisi ollut vähemmän. Koimme kuitenkin, että harjoituksen ja projektin luonteen vuoksi noin

viikon mittaiset sprintit olivat paras lähestymistapa meille.

Ensimmäisessä sprintissä tavoitteena oli saada perus layout kuntoon, eli header ja footer implementoitua kaikille sivuille, ja saada home sekä search page tehtyä mockeista. Toisessa sprintissä tavoitteena oli saada login ja recipe page sekä home ja search page mobiilille tehtyä. Kolmannessa sprintissä tavoitteena oli saada frontend valmiaksi mockupista.

Neljännen sprintin tavoitteena oli saada backend aloitettua, tietokanta pystyyn AWS:ään ja tehtyä käyttäjälle kutsut. Viidennen sprintin tavoitteena oli saada reseptien kutsut tehtyä ja sprintissä kuusi tavoitteena oli backendin valmiaksi saaminen. Sprintissä seitsemän tavoitteena oli integroida frontend ja backend ja sprintissä kahdeksan oli enää projektin deployaus ja bugien korjausta.

3.3 Suunnitelman toteutuminen

Sovelluksen rakenne suunnitelmaan nähden on hyvä. Frontendin mockit vastaavat hyvin pitkälti sovelluksen ulkonäköä. Suurimpana muutoksena oli hakusivun preview, jossa ei ole lopullisessa versiossa lainkaan tietoja reseptistä, mutta tämä on vain, koska sitä ei ollut saatavilla. Poislukien tämän saimme kaikki suunnitellut ominaisuudet ja ulkonäölliset ominaisuudet toteutettua.

Backendin ja tietokannan rakenteet ovat myös hyvin samanlaisia kuin mitä suunnittelime. Tietokannassa satunnaiset arvot muuttuivat, emmekä osanneet suunnitella mapping-pöytien tarvetta. Backendin rakenne vastaa hyvin paljon UML-kaaviota ja sen toiminta on samanlainen kuin suunnitelmassa luodut sekvenssikaaviot.

Hyödynsimme Scrumia jokaisessa kehityksen vaiheessa ja kaikki muutokset kulivat Scrumin kautta. Pidimme myös suunnitellut aloitus-, lopetus- ja päivätapaoamiset. TDD sujui myös suunnitellusti ja loimme testit kaikille paitsi viimeisen sprintin bugien korjauksille, joille emme onnistuneet kirjoittamaan sopivia testejä ennen

korjauksia. Testien kattavuus oli myös lähes 100 %, jota tavoittelimme.

4 Projektin tulokset

4.1 Sovellus

Frontendin lopullinen tulos on meidän mielestämme hyvä ja olemme tyytyväisiä siihen. Frontendissä varsinkin loistaa ohjelman hyvä rakenne ja koodimme on selkeää sekä hyvää suurimmilta osilta. Rakenteessa olemme varsinkin tyytyväisiä siihen, kuinka hyvin jaoimme sivut komponentteihin ja pystyimme hyödyntämään näitä tehokkaasti. Tämän lisäksi loimme aina ulkoisen funktion, jos jotakin funktiota kutsuttiin useasti. Tämä ja yleisesti selkeää koodi ja laaja dokumentointi auttavat koodin selkeyttää ja näiden avulla oli helppo ymmärtää mitä toinen on ohjelmoinut.

Frontend kuitenkin oli hyvin yksinkertainen, eikä sisältänyt vaikeita rakenteita tai animaatioita, joten koodin pitäminen selkeänä ei ollut vaikeaa. Tästä huolimatta SaSS paikoittain muodostui hiukan epäselväksi, mutta enimmäkseen sekin oli selkeää ja rakenne oli hyvä. Sovelluksen error viestit jäivät myös yksinkertaisiksi ja tämä on jotain, mitä meidän olisi pitänyt suunnitella ja toteuttaa paremmin. Frontendissä olemme myös ylpeitä testaamisesta, joka kattaa koko sovelluksen.

Backendissä olemme myös tyytyväisiä sen rakenteeseen, joka jaettiin hyvin omiin osiin ja jako muodostui selväksi. Käytettiin hyvin taas uudelleen jo määriteltyjä funktioita, eikä luotu turhaan uusia osia sovellukseen.

Vaikka backendin koodi on suurimmilta osilta selkeää ja hyvin jaoteltua, se satunnaisesti on hiukan sekavaa. Tietojen eheyden tarkistaminen suoritettiin vain if-else rakenteella, joka nopeasti muodostui epäselväksi, ja tähän olisi varmasti jokin parempi keino. Tätä vaikeutti myös luokkien huono nimeäminen, joka johti vaikeuk-

siin varsinkin API:n kanssa ja jouduttiin datalle tekemään turhaa refaktorointia, jotta se saatiin muunnettua API:n palauttamasta datasta omaan luokkaamme.

Tietokannan rakenne oli myös hyvä ja tässä oli varsinkin positiivista se, että rakenne pysyi samana lähes koko kehityksen ajan, mikä viittaa hyvään suunnitelluun. Onnistuimme myös hyvin hyödyntämään SQL:ää ja ylipäättäään tietokannan kanssa työskentely onnistui hyvin.

Vaikka rakenne pysyi hyvin samana koko kehityksen ajan, niin valitettavasti emme luottaneet suunnitelmaan ja yritimme yhdistää reseptin ohjeet suoraan resepti luokkaan, joka jouduttiin muuttaamaan lopussa omaksi pöydäksi ja tämä aiheutti hiukan ongelmia. Tietokannassa oli myös sama ongelma kuin backendissä nimeämis-ken kassassa. Meidän olisi pitänyt perehtyä enemmän API:hin ennen kuin päätimme kenttien nimet, koska nimien eroavaisuuksista muodostui täälläkin ongelmia.

4.2 Kehitys

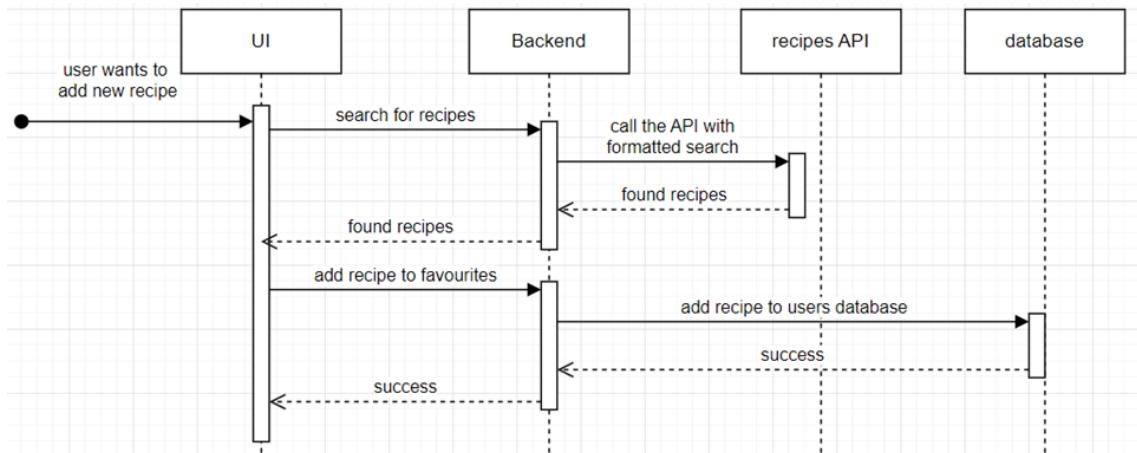
Sprintit onnistuivat suurimmilta osin suunnitelmien mukaisesti. Suurimmaksi kompatuskiveksi muodostui liian vaativat sprintit, minkä seurausena sprintit venyivät muutaman kerran. Tästä yritimme kuitenkin oppia rajoittamaan sprinttien työmääriä ja siirtämään tehtäviä myös seuraavan sprintin puolelle. Lopussa tämä tuntui onnistuvan jo hyvin ja sprinttien työmääriä pysyi realistisena.

Suurimpia onnistumisia sprinteissä olivat selvästi kommunikaatio, yhteistyö ja toisen auttaminen. Retrospektiivit ja aloitustapaamiset olivat monella tapaa hyödyllisiä, sillä ongelmakohtiin ja onnistumisiin tuli kiinnitettyä nopeasti huomiota, jolloin pystyttiin ohjaamaan työskentelyä oikeaan suuntaan. Pienemmätkin asiat oli näiden avulla helppo poimia ja korjata, esimerkiksi jatkuva dokumentoinnin tekeminen eikä sen jättäminen aina viimeiseksi.

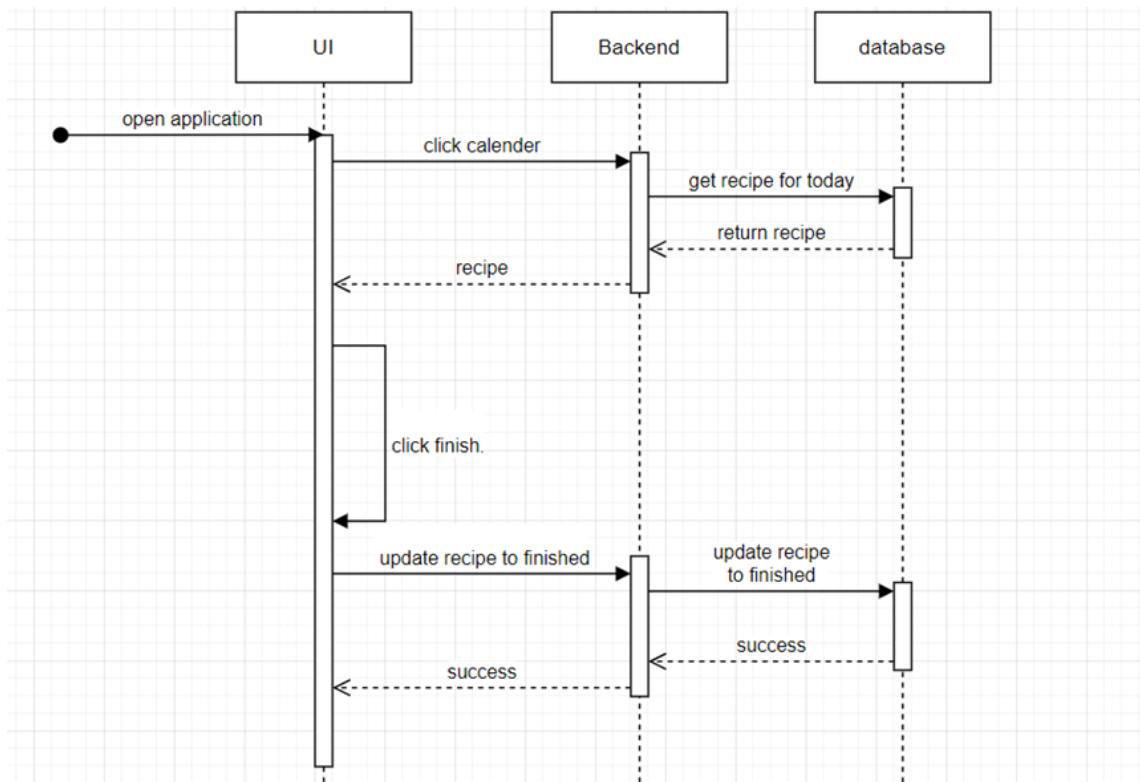
5 Yhteenveto

Kokonaisuutena olemme tyytyväisiä sovelluksen lopputulokseen. Sovellus on omniaisuksiltaan onnistunut. Siinä on hyviä ominaisuuksia, mutta onnistuimme myös huomaamaan kehityskohtia ja tavoitteita tuleville projekteille, joita tulemme luomaan tulevaisuudessa. Olemme molemmat kehittyneet hyvin paljon niin teknillisissä ohjelmointitaidoissa, kuin myös opittu paljon uutta ketterää kehittämisestä ja sen toteuttamisesta. Sovelluksen aikataulu venyi hiukan, mutta tämä ei haittaa ja olemme hyvin ylpeitä siitä kuinka nopeasti loppujen lopuksi onnistuimme toteuttamaan koko sovelluksen ja kuinka pystyimme hyödyntämään ketterää kehitystä ja TDD:tä, jotta pystyimme työskentelemään mahdollisimman tehokkaasti.

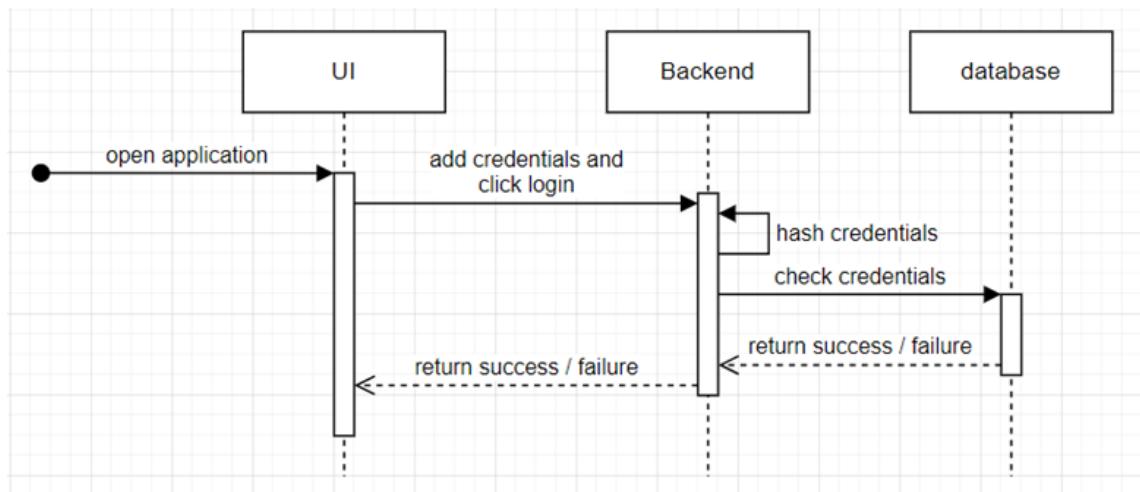
6 Liitteet



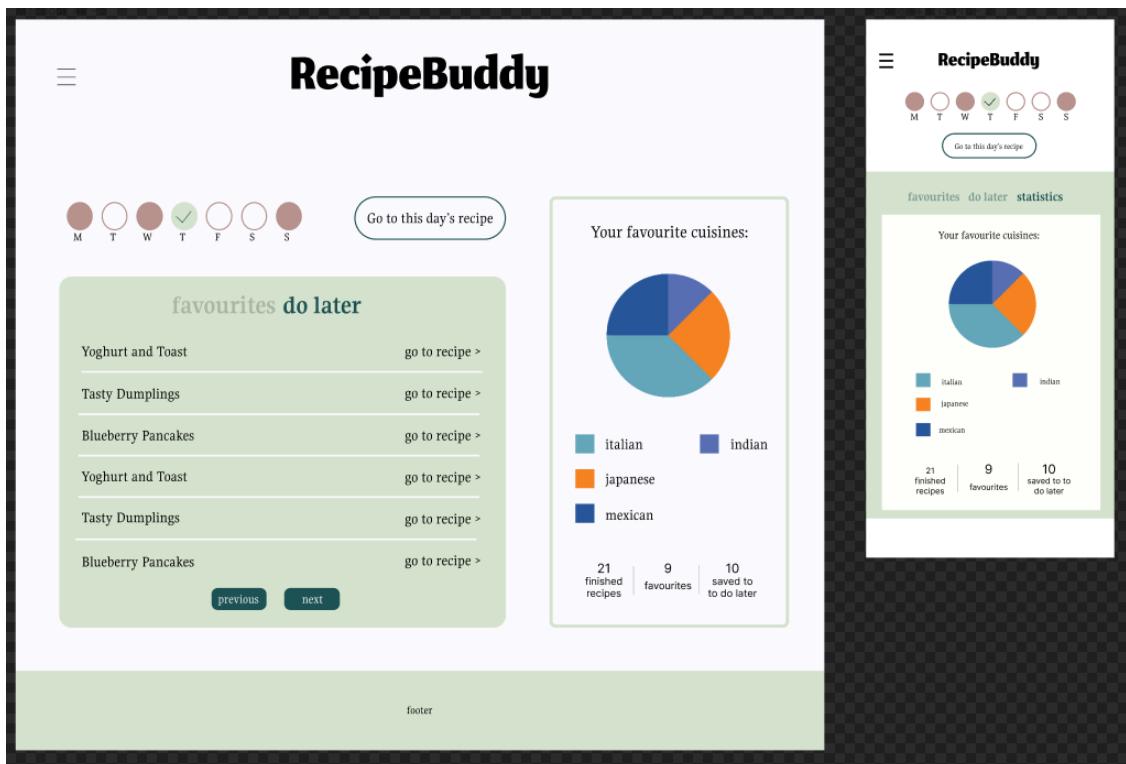
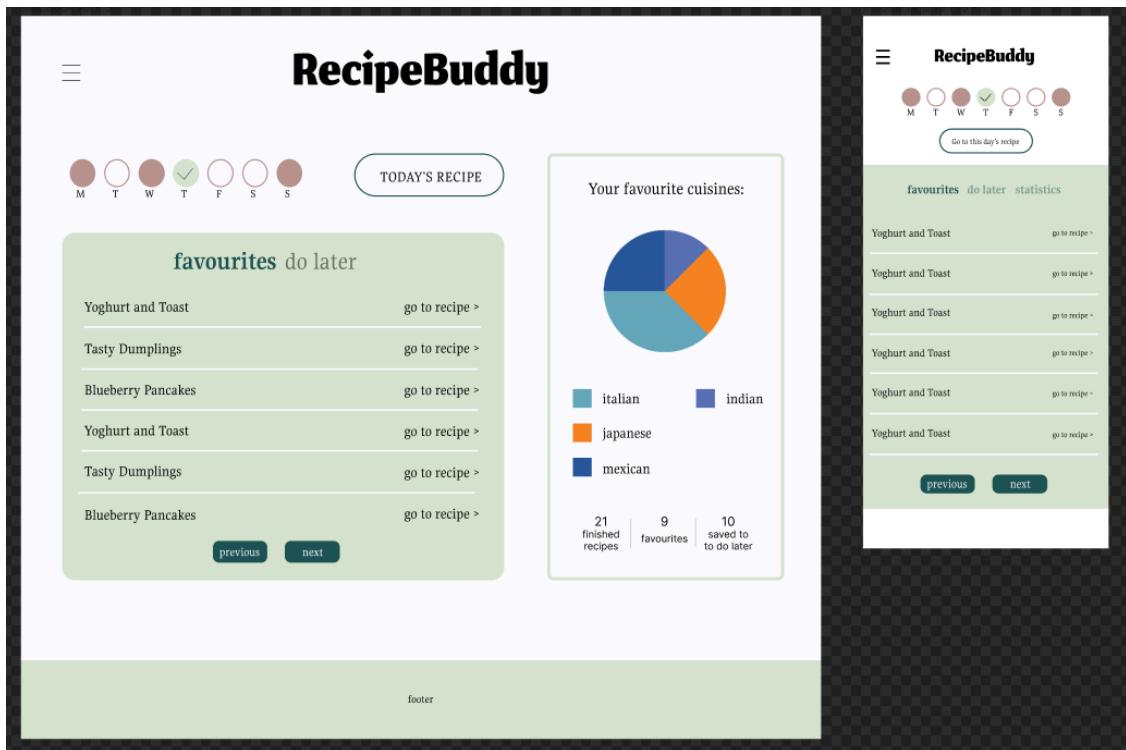
Kuva 6.1: Backend sekvenssikaavio



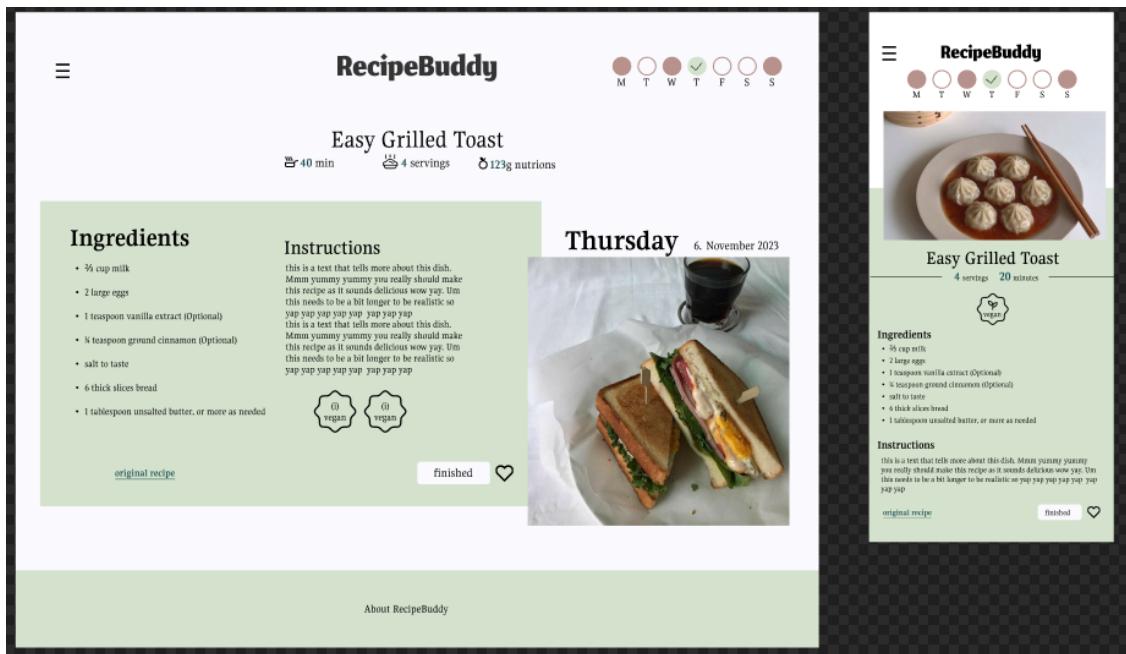
Kuva 6.2: Backend sekvenssikaavio



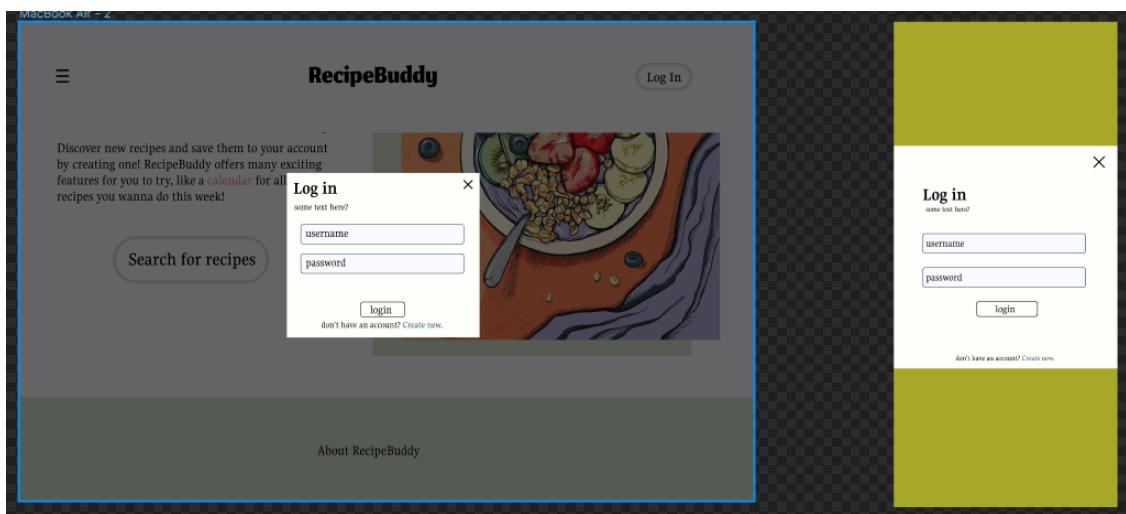
Kuva 6.3: Backend sekvenssikaavio

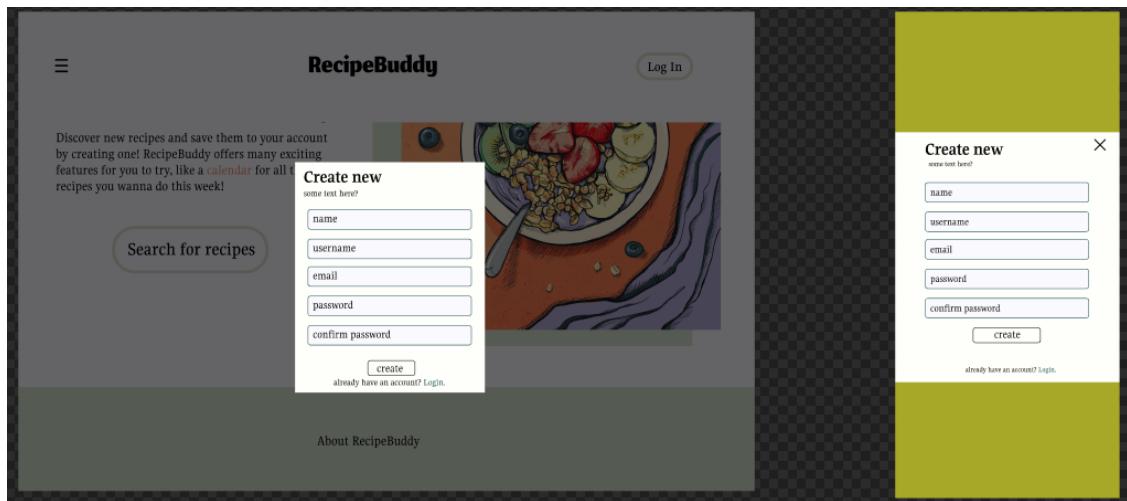


Kuva 6.4: Personal sivun mock

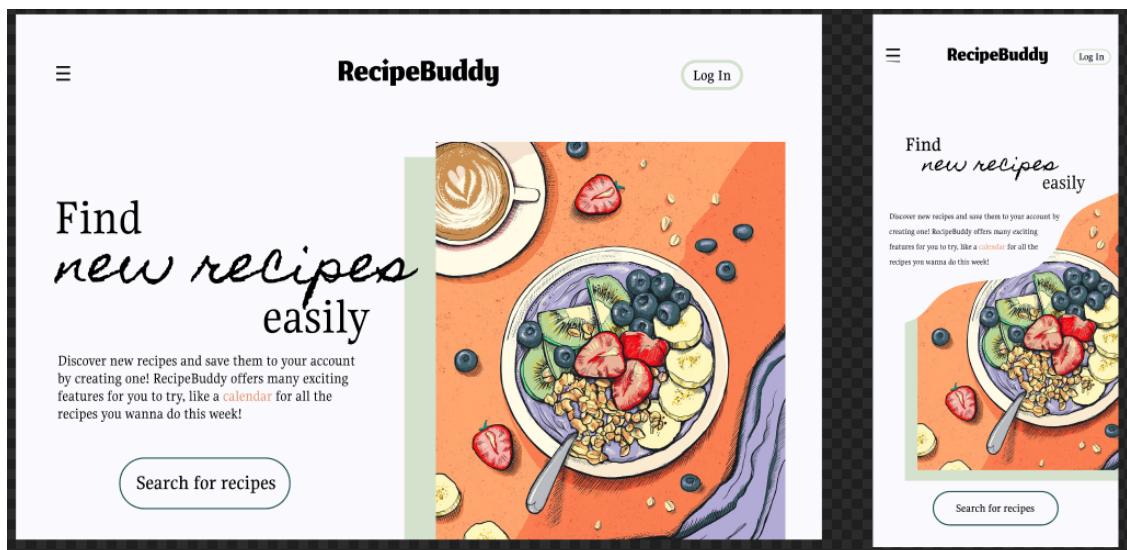


Kuva 6.5: Kalenteri sivun mock

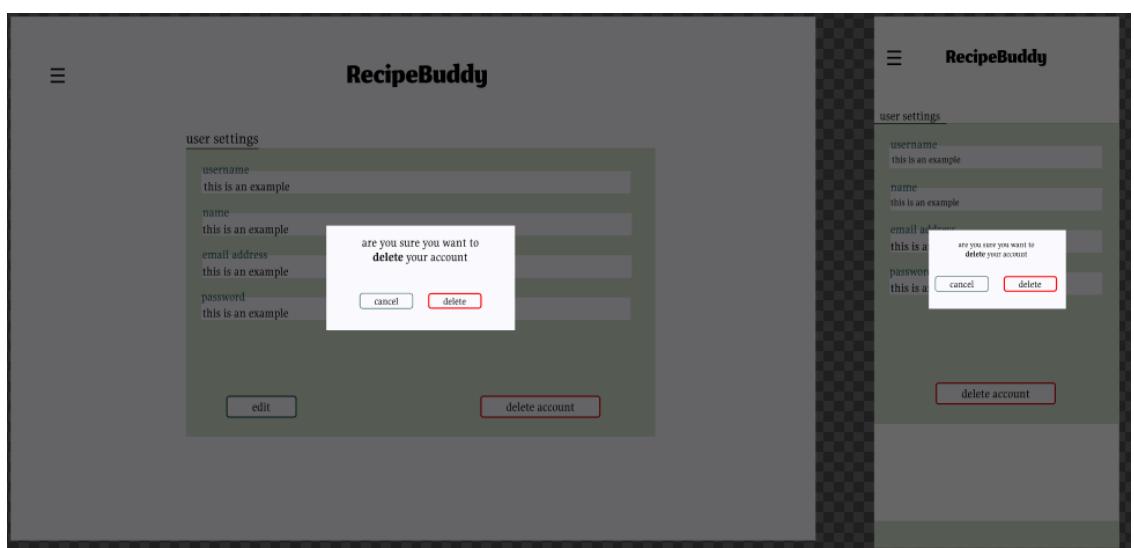
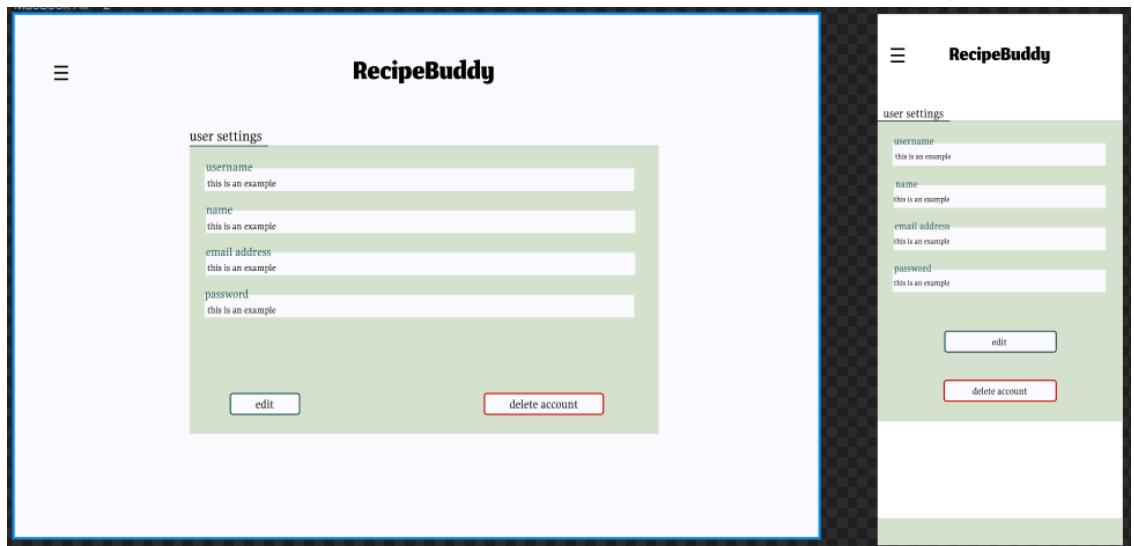


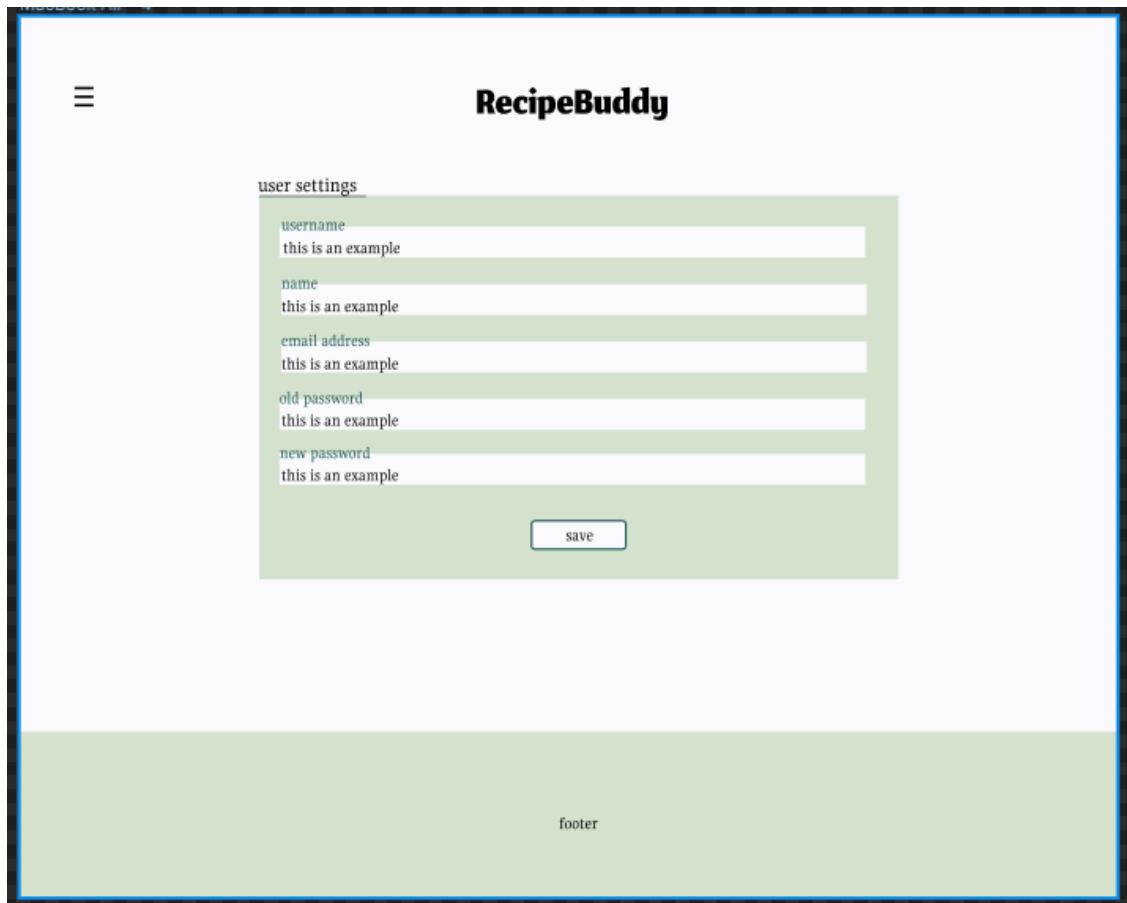


Kuva 6.6: Login-sivun mock

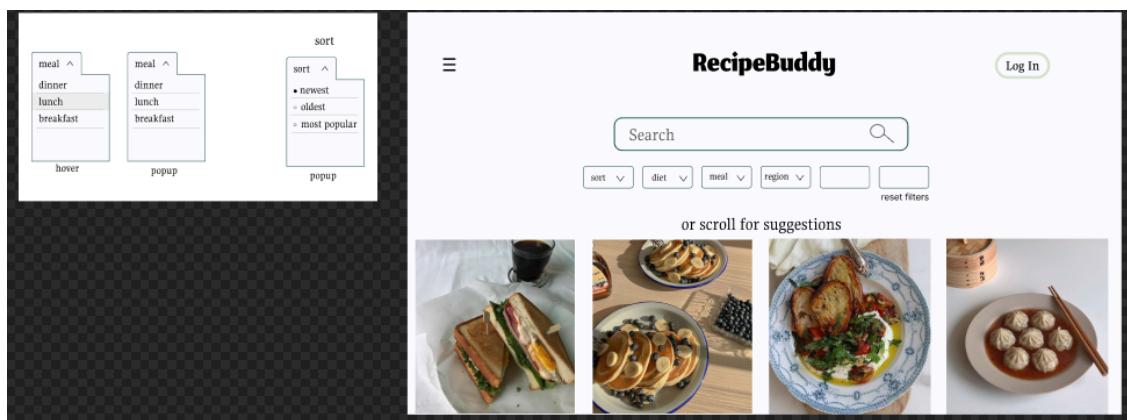


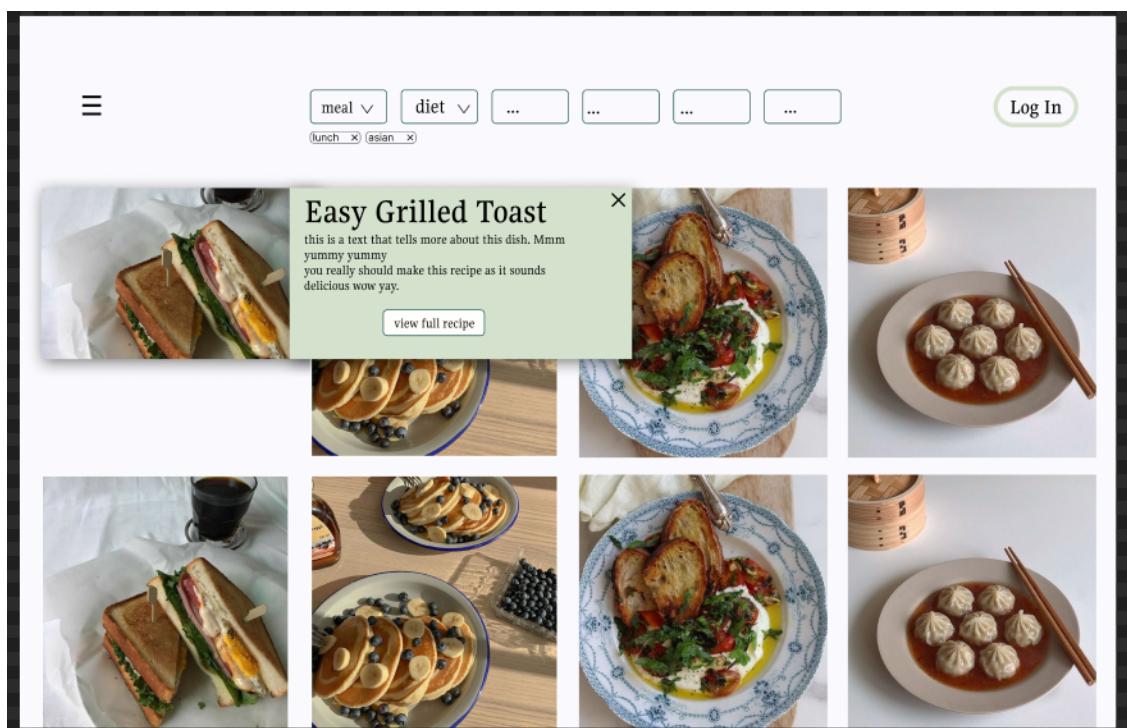
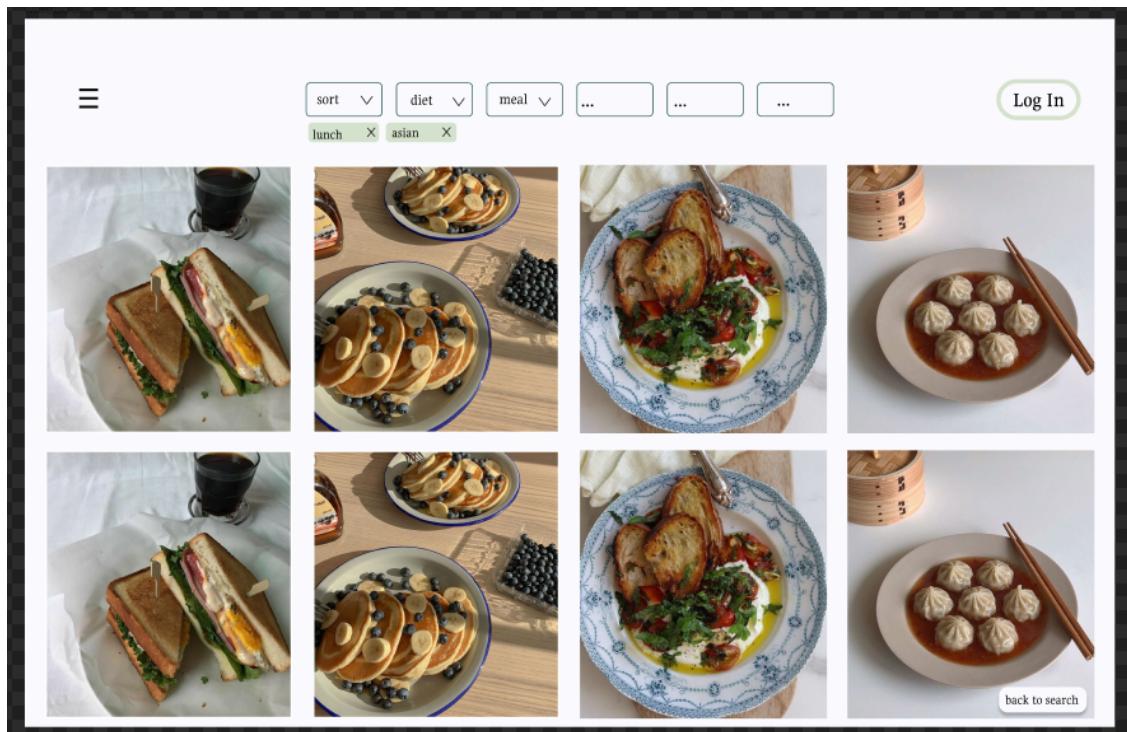
Kuva 6.7: Kotisivun mock

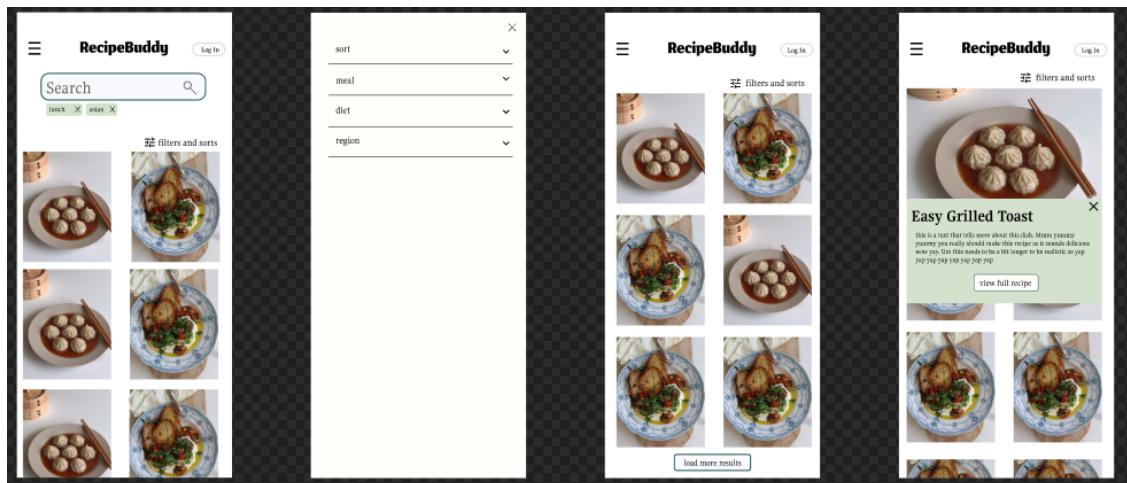
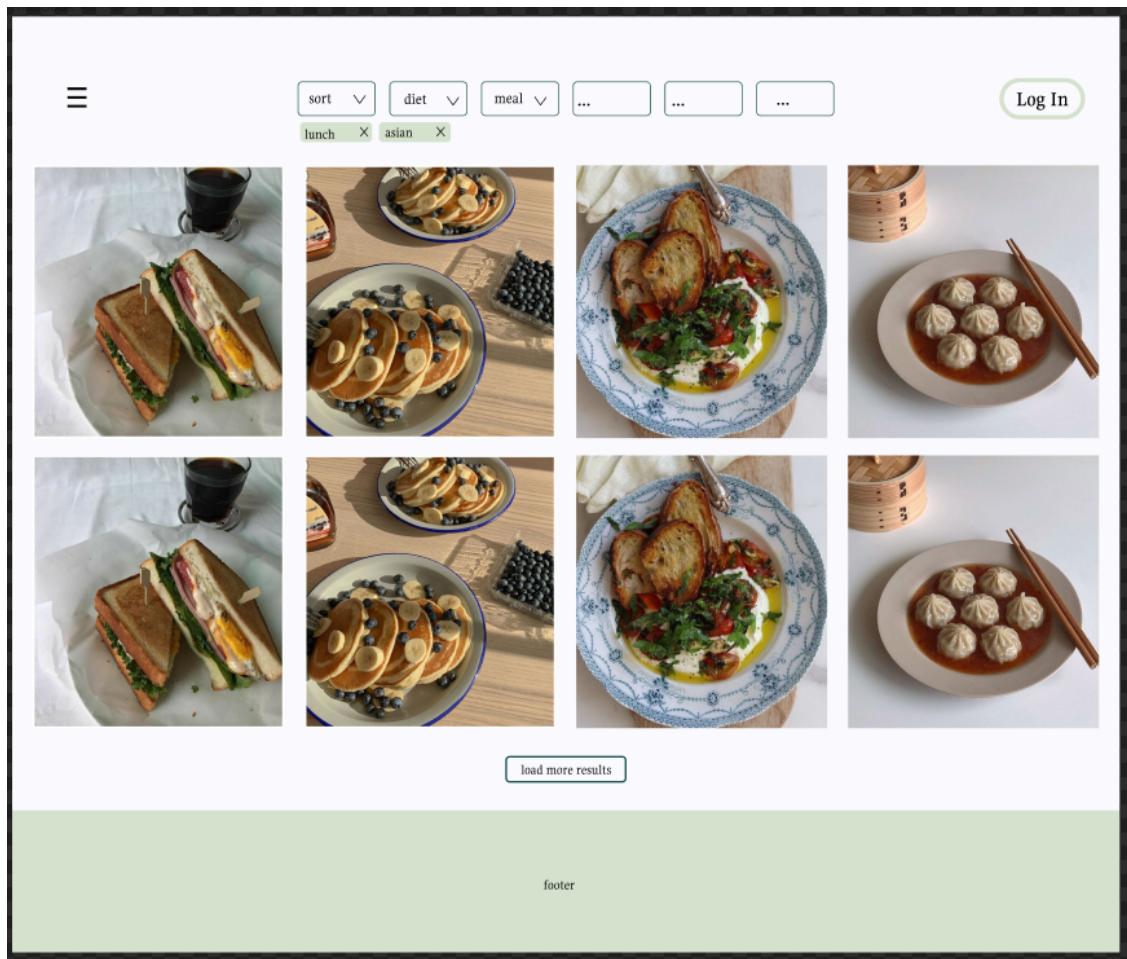




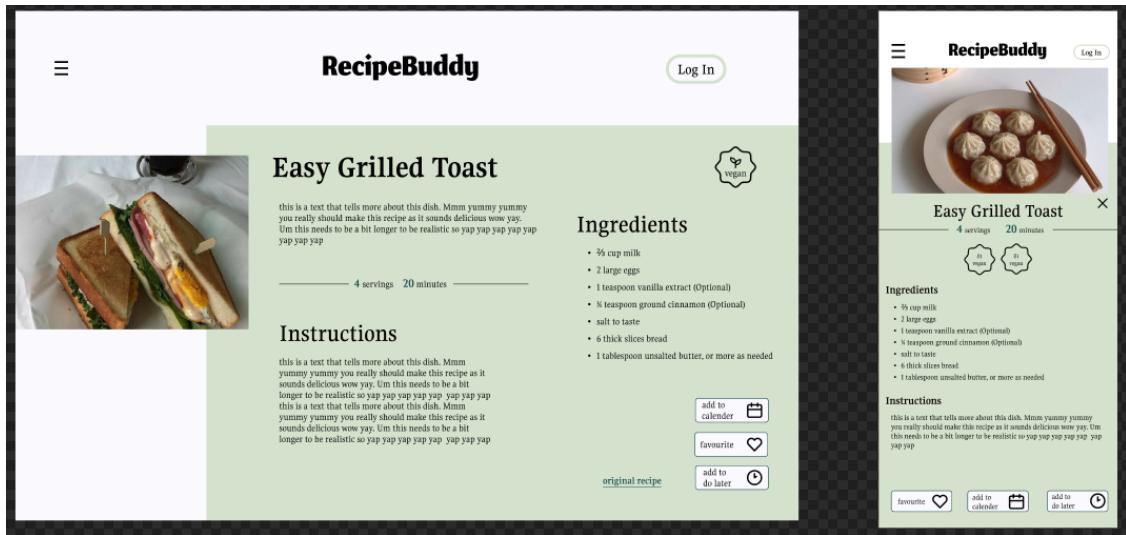
Kuva 6.8: Settings-sivun mock







Kuva 6.9: Hakusivun mock



Kuva 6.10: Reseptisivun mock