Verkkosivuston kehittäminen Next.js:llä

Jeremy Åström

Haaga-Helia ammattikorkeakoulu
Tietojenkäsittely tutkinto
Toiminallinen raportti
2025

Tiivistelmä

Tekijä(t)
Jeremy Åström

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Verkkosivuston kehittäminen Next.js:llä

Sivu- ja liitesivumäärä
30 + 0

Drinkkireseptisivustot eivät ole kehittyneet samaa vauhtia ruokareseptisivustojen kanssa, vaikka kysyntää helppokäyttöiselle ja modernille reseptipalvelulle löytyy. Tämän opinnäytetyön aiheena on kehittää drinkkireseptejä, sekä juoma-arvosteluita käsittelevä verkkosivu käyttäen moderneja teknologioita ja työkaluja. Työn tavoitteena oli kehittää käyttäjäystävällinen ja responsiivinen verkkosivusto, jossa drinkkireseptien sekä juoma-arvosteluiden selaaminen ja hallinta on keskeisiä toimintoja. Työn rajauksena oli sivuston julkaiseminen, sekä laajempi testaaminen.

Verkkosivusto toteutettiin Next.js-nimisellä React-kehyksellä, jonka erikoisuuksiin kuuluu mm. palvelinpuolen renderöinti ja tiedostopohjainen reititys. Sovelluksen tietokantana käytetään MongoDB:tä, ja sen yhteydessä tietokantaintegraatiotyökalua Prismaa, mikä helpottaa tietokantakyselyiden käsittelyä. Drinkkireseptejä ja juoma-arvosteluita voi luoda ainoastaan ylläpitäjät, joiden tunnistamiseen käytetään Clerk-autentikointipalvelua. Sovelluksen käyttöliittymä luotiin React-komponenttien avulla ja tyylittelyyn käytettiin NextUI-komponenttikirjastoa, sekä Tailwind CSS-kirjastoa, mikä mahdollistaa tyylien muokkaamisen suoraan HTML-elementeissä.

Projektissa käytettiin Scrum-projektinhallintamenetelmää, jossa kehittämisvaiheet on jaettu neljään kahden viikon sprinttiin. Kehittämisen osa-alueet on listattu tuotteen kehitysjonoon (Project backlog).

Tuloksena syntyi toimiva verkkosivusto, jossa käyttäjät voivat selata reseptejä, sekä juoma-arvosteluita ja ylläpitäjät voivat hallita niitä. Projekti saavutti suurimman osan vaatimuksistaan, mutta laajempi testaaminen sekä hakukentän luominen jäivät toteuttamatta. Kokonaisuutena projekti syvensi oppimista full-stack kehittäjänä.

Asiasanat
Next.js, React, MongoDB, Scrum, Full-stack

Sisällys

1 Johdanto

Suomalaisessa juomakulttuurissa drinkit ja mocktailit eivät ole vielä yhtäläisessä suosiossa valmiiden lonkeroiden kanssa, mutta nykyaikaisella ja käyttäjäystävällisellä verkkosivulla saadaan uusia ihmisiä kokeilemaan drinkkien tekoa. Markkinoilta löytyy paljon erilaisia drinkkiresepti sivustoja, mutta ne eivät ole kokenut samanlaista panostusta kuin esimerkiksi ruokareseptejä tarjoavat sivustot. Tässä markkinassa pärjää moderni, käyttäjäystävällinen ja responsiivinen verkkosivusto, joka kehittyy käyttäjien tarpeiden mukaan. Aihe on ajankohtainen, sillä yhä enemmän palvelut siirtyvät digitaalisiksi, koska kuluttajat vaativat verkkopohjaisia palveluita. Palvelun digitalisoiminen, verrattuna vaikka reseptikirjaan, tuo nopeuden tuotteen julkaisuun, muokkaamiseen ja muille jakamiseen. Lisäksi se parantaa saavutettavuutta, sillä kuka tahansa internetyhteyden päässä oleva pääsee lukemaan reseptejä älylaitteellaan.

Tämän opinnäytetyön aiheena on drinkkireseptejä, sekä juoma-arvosteluita käsittelevän verkkosivun kehittäminen käyttäen moderneja web-teknologioita ja työkaluja. Päädyin aiheeseen, koska olen pitkään halunnut luoda kyseisen palvelun ja se tarjoaa samalla mahdollisuuden kehittää taitojani uusien työkalujen kanssa sekä syventää osaamistani ohjelmistokehittäjänä.

Projektin tavoitteena on luoda moderni, teknisesti toimiva ja käyttäjien tarpeita vastaava verkkosivusto. Sivusto on suunnattu kaikille erilaisista drinkkiresepteistä tai juoma-arvosteluista kiinnostuneille täysikäisille. Harrastelijat voivat etsiä kaapista löytyville aineksilleen uusia reseptejä ja ammattilaiset, kuten baarimikot voivat löytää uutta inspiraatiota työhönsä. Juoma-arvosteluiden tarkoitus on tuoda sivuston käyttäjille uutta tietoa, heille mahdollisesti tuntemattomista tuotteista.

Projektin toiminalliset tavoitteet keskittyvät ylläpitäjien sekä käyttäjien vaatimuksiin. Ylläpitäjälle tarjotaan mahdollisuus hallita reseptejä selkeästi ja tehokkaasti ylläpitäjien käyttöliittymässä. Käyttäjille luodaan responsiivinen käyttöliittymä, jolla on helppo selata ja jakaa reseptejä. Projektin tavoitteisiin kuuluu myös onnistunut projektinhallinta, jonka kautta projekti etenee määrätysti ja jatkokehittäminen on mahdollista tulevaisuudessa. Tähän kuuluu käyttäjien vaatimusten, ja niihin liittyvien työvaiheiden onnistunut priorisointi.

Verkkosivun kehitys perustuu kahden käyttäjäryhmän tarpeisiin: ylläpitäjien ja sivuston käyttäjien. Ylläpitäjien toiminnallisista vaatimuksista ensimmäinen on reseptien hallinta, joka mahdollistaa luomaan, muokkaamaan ja poistamaan drinkkireseptejä. Toisena vaatimuksena on juoma-arvosteluiden hallinta, johon pätee samat vaatimukset, kuin drinkkireseptien hallintaan. Kolmantena tavoitteena on suojata ylläpitäjien käyttöliittymä Clerk-autentikointipalvelun avulla, jolloin vain valtuutetut käyttäjät voivat kirjautua ylläpitosivuille. Verkkosivuston käyttäjien toiminallisista vaatimuksista ensimmäisenä on drinkkireseptien selaaminen, jolloin käyttäjä voi valita listasta haluamansa drinkin ja

tarkastella sen ainesosia sekä ohjeita. Toisena on juoma-arvosteluiden selaaminen ja lukeminen, jolloin käyttäjät voivat lukea arvosteluja erilaisista juomatuotteista, kuten oluista ja viineistä. Kolmantena tavoitteena on luoda hakukenttä drinkkireseptien selaamiseen, mikä nopeuttaa ja helpottaa oikean reseptin löytämistä.

Työ tarkastelee erilaisia web-teknologioita, kuten Next.js joka on Stack Overflow:n kyselyn suosituin React-kehys ja MongoDB joka on suosituin ei-relaatiotietokanta (Stack Overflow 2024). Lisäksi tarkastellaan web-työkalujen, kuten Clerk:n ja Prisman, käyttöä Next.js:n kanssa. Teknologisina tavoitteina on oppia käyttämään web-teknologioita ja työkaluja Next.js:n kanssa, kuten Clerk, Prisma ja MongoDB, ja hyödyntää näitä ohjelmistokehittämisessä. Lisäksi tavoitteena on syventää suunnittelu- ja toteutusmenetelmien osaamista full-stack kehityksen parissa, erityisesti käyttöliittymän, tietokannan ja palvelinpuolen ratkaisujen osalta. Tähän liittyy myös vahvistaa osaamista priorisoida ja sopeuttaa työvaiheita projektin aikana.

Projektissa sovelletaan scrum-projektinhallinta menetelmää, jossa kehitystyö jaetaan neljään kahden viikon sprinttiin. Ensimmäinen sprintti sisältää vaatimusmäärittelyä sekä suunnittelua. Seuraavat kaksi sprinttiä keskittyvät sivuston kehittämiseen. Kehittämisen osa-alueet on listattu tuotteen kehitysjonoon, jossa jokaisessa vaiheessa keskitytään sovelluksen tiettyyn toimintoon esim. tietokannan tai käyttöliittymän luontiin. Viimeinen sprintti painottuu virhekorjauksiin, sekä käyttäjätestaamiseen ja palautteen perusteella tehtäviin muutoksiin. Jokaisen sprintin jälkeen käydään läpi aikataulun toteutumista ja tarkastellaan vaatimusten ja työvaiheiden priorisointia.

Tällä työllä ei ole toimeksiantajaa. Sovellusta ei ole tarkoitus alussa kaupallistaa, mutta mahdollisuudet sille on olemassa.

Työn ulkopuolelle jää domainin hankinta ja sovelluksen julkaiseminen, koska se ei oleellisesti kuulu tekniseen ohjelmiston kehitykseen. Projektin aikarajoitteen vuoksi laajamittainen testaus jää niin ikään pois.

2 Teknologiat ja menetelmät

2.1 Backend

Tässä osuudessa käsitellään Next.js-kehystä, jolla voidaan luoda moderneja full-stack-verkkosovelluksia. Sen olennainen ominaisuus on palvelinpuolen renderöinti ja tiedostopohjainen reititys, joista muun muassa lisää tietoa. Lisäksi keskitytään MongoDB-tietokantaan, joka on joustava ja tehokas dokumenttipohjainen tietokanta.

2.1.1 Next.js

Next.js on React-kehys, jonka avulla voi sujuvasti kehittää full-stack-verkkosovelluksia Suurin ero Reactin ja Next.js:n välillä on siinä, että React on vain JavaScript-kirjasto, kun taas Next.js on kehys, joka mahdollistaa kokonaisvaltaisten full-stack-sovellusten luomisen. (Riva 2022, luku 1) Next.js on monen suuryrityksen käytössä mm. Spotify ja Nike (Next.js s.a. c).

Next.js mahdollistaa käyttöliittymien luomisen React-komponenttien avulla, tiedostopohjaisen reitityksen sekä palvelinpuolen, että asiakaspuolen renderöinnin. Lisäksi sen kanssa voi käyttää moderneja työkaluja kuten, Tailwind css ja TypeScript. (Next.js s.a. b)

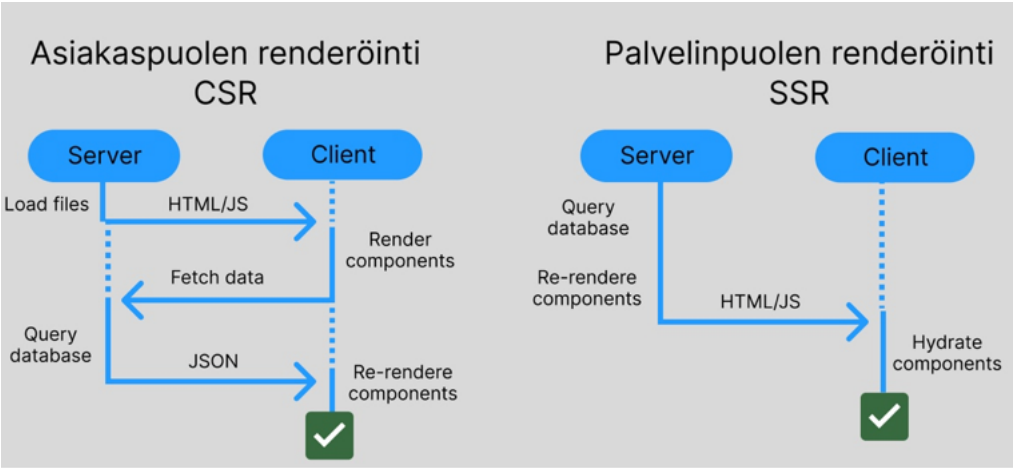
Tiedostopohjainen reititys käyttää App Router -paradigmaa, jossa sovelluksen reitit (Routes) määritellään kansiorakenteen (folder structure) avulla. Sovelluksen juurihakemisto (Root) on /src/app tai suoraa /app. Jotta saadaan uusi reitti /app/drinkit, luodaan kansio nimeltä "drinkit juureen. Tähän kansioon lisätään vielä page.js- tai, jos käytössä TypeScript page.tsx-tiedosto, jonne lisätään haluttu sisältö. Kansioon voi vaihtoehtoises lisätä route.js-tiedoston API-reitin luomiseen. Kansioon voi myös lisätä layout.js-tiedoston, jolla voidaan määrittää reitille ja sen alireiteille yhteisiä tyylejä. (Bugl 2024, luku 16)

Next.js tarjoaa kehittäjälle valinnan komponenttien renderöinnin määrittelyssä. Sen avulla voidaan päättää, mitkä komponentit renderöidään asiakaspuolella, mitkä palvelinpuolella, ja mitkä rakennusvaiheessa staattisiksi. Tämä mahdollistaa dynaamisten ja optimoitujen sovellusten kehittämisen joustavasti ja tehokkaasti. (Riva 2022, luku 2)

Palvelipuolen renderöinti (Server-Side rendering, SSR), asiakaspuolen renderöinti (Client-side rendering, CSR) sekä staattisen sivun generointi (Static site generation, SSG) ovat eri tapoja rakentaa ja esittää verkkosivuja. Kaikissa tavoissa on puolensa, jotka vaikuttavat sivuston käyttökokemukseen, suorituskykyyn, hakukone optimointiin (Search engine optimization, SEO) sekä palvelimen kuormitukseen. (Riva 2022, luku 2) Näitä asioita, sekä eri menetelmien käyttökohteita on käyty läpi taulukossa 1.

Palvelinpuolen renderöinti tarkoittaa yleisesti, että verkkosivuston HTML-renderöinti tapahtuu palvelimella, jonka jälkeen se lähetetään käyttäjän selaimeen. Jokaisen uuden pyynnön tultua, palvelin käsittelee ja jäsentää verkkosivun HTML-koodin ennen kuin se välitetään selaimeen. Tämä tarkoittaa, että käyttäjä saa valmiin HTML-dokumentin, jonka käyttäminen on nopeaa, koska kaikki sisältö on jo ladattu. Tämän takia sovellus täytyy julkaistu SSR:lle tarkoitetulle palvelimelle. SSR parantaa usein sivuston SEO-optimointia, sillä palvelimelta tuleva HTML on heti valmis hakurobottien käyttöön (kuva 1). Datan ja evästeiden hallinta palvelinpuolella lisää myös turvallisuutta, näin ollen yksityistä dataa ei paljasteta suoraan. Haittapuolena SSR:ssä on, että yksinkertaiset HTML-sivut voivat jäädä vähemmän interaktiivisiksi. Tämän takia SSR toteutetaan usein yhdessä "koodin jakamisen" ja "hydration"-prosessin kanssa, jotka tekevät palvelimella renderöidyistä sivuista dynaamisia. (Riva 2022, luku 2)

Asiakaspuolen renderöinti taas tarkoittaa yleisesti, että verkkosivuston HTML-renderöinti tapahtuu selaimessa. Pyynnön tultua palvelin lähettää tyhjän HTML-pohjan sekä paketin JavaScript-tiedostoja, jotka sisältävät kaiken sisällön ja logiikan verkkosivun rakentamiseen pyytäjän selaimessa. Tämä tekee sovelluksesta nopean ja interaktiivisen, joka mahdollistaen dynaamiset päivitykset sekä saumattomat siirtymät ilman sivun uudelleenlatauksia. Koska selain joutuu itse tuodun logiikan avulla luomaan HTML-dokumentin, sen ensimmäinen lataus on monivaiheisempi ja hitaampi verrattuna palvelinpuolen renderöintiin, kuten kuvasta 1 huomaa. Tämä heikentää hakukoneiden kykyä indeksoida sisältöä.



Kuva 1. Asiakaspuolen- sekä palvelinpuolen renderöintitavat (mukaillen Bugl 2024, luku 17)

Staattisen sivun generointi (SSG) tarkoittaa, että sivut luodaan staattisiksi jo build-vaiheess, eikä palvelimella tai selaimessa. Tämä sopii hyvin sivuille, joiden sisältö ei muutu usein. Sivut latautuvat nopeasti, koska ne tulevat suoraan valmiina verkkopalvelimelta. Koska sivut ovat luotu jo build-vaiheessa, sen sisältö pysyy muuttumattomana seuraavaan julkaisuun asti. (Riva 2022, luku 2)

Taulukko 1. Vertailutaulukko renderöintimenetelmille (GeeksforGeeks 2023)

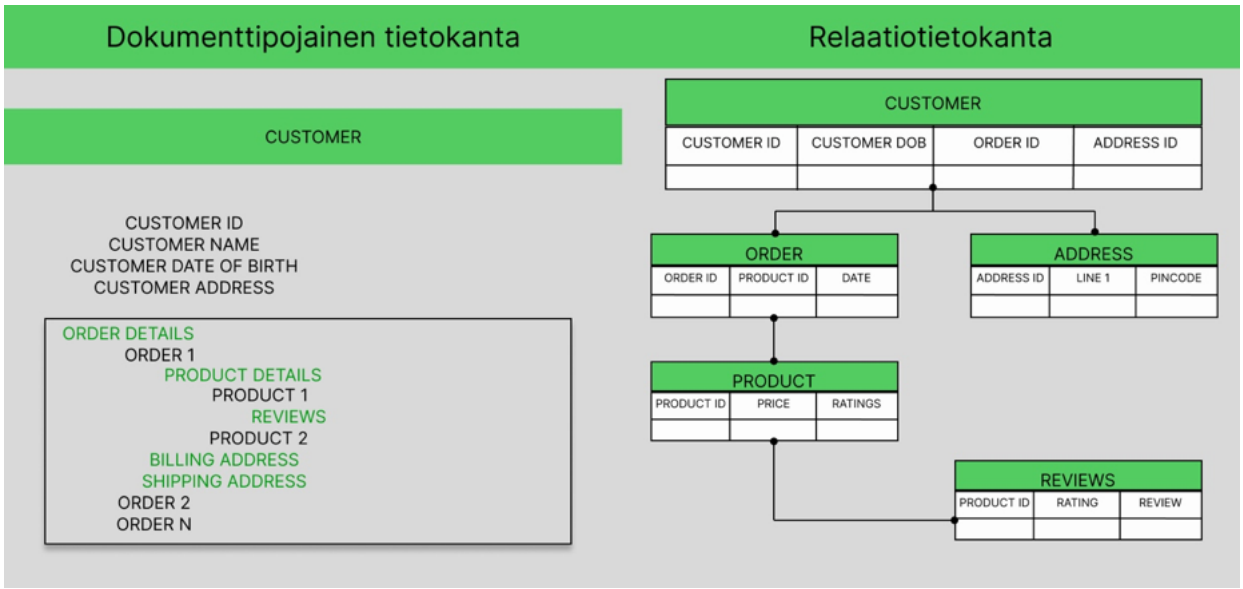
Lähestymistapa	Edut	Haitat	Toimintaperiaate	Käyttökohteet
Palvelinpuolen renderöinti (SSR)	Nopea ensilataus, parempi SEO-optimointi	Lisääntynyt palvelimen kuormitus, rajallinen interaktiivisuus	Palvelin renderöi HTML:n ja lähettää sen asiakkaalle käytettäväksi	Sisältöpainotteiset verkkosivustot, kuten blogeissa tai uutisverkkosivustoilla
Asiakaspuolen renderöinti (CSR)	Interaktiivisemmat ja dynaamisemmat verkkosovellukset, sulavampi käyttäjäkokemus	Hidas ensilataus, heikko SEO-optimointi	Palvelin lähettää alkuperäisen HTML:n, asiakas päivittää JavaScriptillä	verkkosovelluksissa, jotka vaativat korkean tason interaktiivisuutta, kuten sosiaalisen median alustoilla tai verkkokaupoissa
Staattinen sivun generointi (SSG)	Nopea ensilataus ja dynaamiset päivitykset, parempi SEO-optimointi	Rajallinen interaktiivisuus ja dynaamiset päivitykset, lisääntynyt palvelimen kuormitus	Palvelin generoi staattisen HTML:n, asiakas päivittää JavaScriptillä	Verkkosivut, jotka vaativat vähän interaktiivisuutta, kuten landing page tai portfoliosivu.

2.1.2 Tietokannat

MongoDB on ohjelmoijien suosituin ei-relaatiotietokanta (Stack Overflow 2024). Perinteisen relaatiomallin sijaan MongoDB käyttää dokumentteja, jotka mahdollistavat monimutkaisten tietorakenteiden käsittelyn. Se on myös suosituin dokumenttipohjainen tietokantajärjestelmä, keräten 400.3 pistettä DB-Enginesin Joulukuun 2024 rankingissa verrattuna perässä tuleviin: Databricks (87.6) ja Amazon DynamoDB (72.7). (DB-Engines s.a. a) Vaikka dokumenttipohjaiset tietokannat ovat toiseksi suurin tietokantajärjestelmä kategoria kymmenellä prosentilla. Vertailussa se on osuudeltaan huomattavasti pienempi, kuin relaatiotietokantajärjestelmät, joiden suosio on yli 72 prosenttia. Sen taakse kolmanneksi jää kuitenkin huomattavalla erolla avainarvotietokantajärjestelmä (Key-value). (DB-Engines s.a. b)

MongoDB:n erikoisuuksiin kuuluvat toissijainen indeksointi, aggregaatioratkaisut ja TTL-kokoelmat. Sen vapaa skeemarakenne mahdollistaa vaihtelevien ominaisuuksien käsittelyn, mikä tekee

ohjelmoinnista sujuvampaa. Esimerkiksi kehittäjät voivat hallita kenttiä ilman, että rakenteen täytyy olla kiinteä. MongoDB tukee monia erilaisia toissijaisia indeksejä, esimerkiksi hierarkkisille rakenteille, kuten sisäkkäisille (Nested) dokumenteille ja taulukoille. MongoDB skaalautuu horisontaalisti eli se voidaan jakaa monelle palvelimelle datamäärän kasvaessa. Automaattinen kuormantasaus nopeuttaa luku- ja kirjoitusoperaatioita palvelimien välillä. MongoDB hyödyntää RAM-muistia tehokkaasti ja optimoi kyselyt automaattisesti, mikä ylläpitää korkeaa suorituskykyä. MongoDB on siis suunniteltu sovelluksiin, joissa tarvitaan joustavaa, skaalautuvaa ja suorituskykyistä tietokantaa, joka tukee nopeatempoista kehitystä ja suuriakin datamääriä. (Branshaw, Brazil & Chodorow 2019, luku 1)



Kuva 2. Dokumenttipohjainen tietokanta ja relaatiotietokanta (mukaillen MongoDB s.a)

ORM (Object relational mapping) on ohjelmointitekniikka, jonka pyrkimyksenä on yhdistää tietokantojen ja olio-ohjelmoinnin (OOP Object-oriented programming) erilaiset tietojen esitystavat. Se sijoittuu tietokannan ja sovelluksen väliin, muuttaen tiedon oikeanlaiseksi kummallekkin järjestelmälle. Tämän helpottaa kehittämistyötä ja parantaa järjestelmien yhteensopivuutta. (Kouraklis 2019, luku 1)

Prisma on ORM-työkalu, joka tekee tiedonhallinta sovelluksen ja tietokannan välillä on helppoa. Se korostaa tyyppiturvallisuutta ja kehittäjäkokemusta tarjoamalla työkaluja, kuten automaattisesti generoidut tietomallin TypeScript-tyypit ja kyselyrakentajan, josta esimerkki luvun 3.2 koodipätkässä "getDrink". Se yksinkertaistaa hankalia tietokantaoperaatioita, kuten realaatioiden hallintaa ja se tukee useita tietokantoja kuten, PostgreSQL ja SQLite. (Chernenko 2024, luku 9)

2.2 Frontend

Tässä osuudessa käydään läpi Reactin keskeiset ominaisuudet, kuten komponenttipohjaisuus, virtuaalinen DOM ja esitellään muutama ns. React koukku (React hook). Lisäksi käydään läpi Reactin laajaa ekosysteemiä, mikä on tehnyt siitä niin suositun. Kerrotaan TypeScript:stä, joka on käytössä koko projektissa sekä Tailwind:stä jonka avulla luodaan tyylit projektin käyttöliittymiin.

2.2.1 React

React on Metan kehittämä ja käyttämä JavaScript-kirjasto, jota hyödynnetään käyttöliittymien (UI) luontiin. Ractin keskeisiin ominaisuuksiin kuuluu komponenttipohjaisuus, virtuaalinen DOM (Virtual document object model) sekä laaja ekosysteemi. Sen avulla luodaan käyttöliittymä käyttäen pieniä uudelleenkäytettäviä komponentteja. Virtuaalinen DOM vertaa sivulla tapahtuvia muutoksia oikeaan DOM:n ja päivittää vain tarvittavat osat sinne. (Rippon 2023, luku 1)

Reactin ekosysteemi on sen ympärille rakennettu laaja kokoelma kirjastoja ja työkaluja. Se koostuu sadoista eri teknologioista, joita käytetään käyttöliittymän kehittämiseen, arkkitehtuurin hallintaan sekä tuottavuuden parantamiseen. Käyttöliittymän kehityksessä käytetään erilaisia UI-kirjastoja, kuten Tailwind SS ja Material UI sekä animaatio työkaluja, kuten Framer Motion. Arkkitehtuurin kehityksessä käytetään hallintatyökaluja, kuten Redux sekä autentikointikirjastoja, kuten AuthO. Tuottavuutta kehitetään käyttämällä kehitystyökaluja, kuten TypeScript ja Node.js, sekä testauskirjastoja, kuten Jest. Tämän ekosysteemin laajuus tekee Reactin kanssa kehittämisestä joustavaa, mutta on tärkeää valita projektiin juuri oikeat kirjastot ja työkalut. (Barklund 2024, luku 1)

Reactin komponentit ovat rakennuspalikoita, jotka koostuvat uudelleenkäytettävistä osioista, jotka määrittelevät käyttöliittymän rakenteen, toiminnan ja ulkoasun. Komponenteissa tiedonhallinta tapahtuu kahdella tavalla: props (ominaisuudet) ja state (tila). Propsien avulla pystytään jakamaan tietoa toisille komponenteille lukumuodossa, kuten komponentin lapsilleen (children). Staten avulla voidaan hallita ja päivittää tietoja komponentin sisällä dynaamisesti. (Sakhniuk & Boduch 2024, luku 3)

React komponentteja täydentää React koukut, jotka ovat erilaisia funktioita, joiden avulla tietoa voidaan hallita komponentissa. Seuraavaksi muutama esimerkki yleisistä koukuista. UseState palauttaa kaksi elementtiä: nykyisen tilan ja funktion, joka päivittää sitä. Tässäkin työssä on käytetty kyseistä funktiota mm. drinkin ainesosien hallintaan. UseEffect mahdollistaa sivullisten toimintojen suorittamisen, kuten tiedon hakemisen. Sitä voidaan käyttää komponentin elinkaaren eri vaiheissa, kuten ensimmäisessä renderöinnissä tai jonkun tilan muuttuessa. (Sakhniuk & Boduch 2024, luku 3)

2.2.2 TypeScript

TypeScript on JavaScriptin laajennus, joka tuo staattisen tyypityksen ja muita ominaisuuksia, jotka parantavat koodin ymmärrettävyyttä ja ylläpitoa. Se mahdollistaa muuttujien ja funktioiden tyyppien määrittelyn, rajapinnan objektien rakenteiden tarkistamiseen sekä tukee pääsyrajoittimia luokkien jäsenten hallintaan. Lisäksi se mahdollistaa nimettyjen vakioiden määrittelyn enumien avulla ja tarjoaa edistyneitä tyyppiominaisuuksia, kuten geneerisyydet, union- ja tuple-tyypit. TypeScript ei ole ohjelmointikieli, joka suoritettaisiin suoraan tulkissa, kuten JavaScript, vaan se käännetään ensin JavaScriptiksi. (Vanderkam 2024, luku 1) Staattinen tyypitys, yksi TypeScriptin keskeisistä eduista, ehkäisee virheitä jo käännösvaiheessa. Kehitysvaiheessa se auttaa automaattisella täydennyksellä, navigoinnilla ja refaktoroinnilla. Kaikki sen hyödyt tulevat parhaiten esiin suurissa koodikannoissa, jossa se tekee koodista selkeämpää ja toimii osittain myös dokumentaationa. (Chernenko 2024, luku 2)

2.2.3 Tailwind

Tailwind on utility-lähtöinen CSS-kehys, joka tarjoaa uudelleenkäytettäviä luokkia sovellusten ulkoasun kehittämiseen. Sen avulla kehittäjä voi lisätä suoraan HTML-elementtiin CSS-ominaisuuksia. Tailwind sisältää ennalta määrättyjä tyylittelysääntöjä, kuten värit ja mitat, mikä vähentää valinnanvarasta aiheutuvaa kuormitusta ja edistää johdonmukaisuutta. (Gerchev 2022, luku 1)

```
<p className="text-5xl md:text-9xl p-10  text-center ">{text}</p>;
```

Tässä Header-komponentin html-elementissä on käytetty neljää tailwind luokkaa: tekstiä on määritelty isommaksi ja suuremmilla näytöillä vielä isommaksi. P-10 tuo elementin ympärille 10 yksikön täytteen (padding) ja viimeisenä teksti kohdistetaan keskelle.

2.3 Menetelmät ja työkalut

Tässä osuudessa käydään läpi Scrum-menetelmää, joka jakaa projektin lyhyisiin jaksoihin eli sprintteihin. Tämän lisäksi tarkastellaan scrumban-menetelmää, joka yhdistää Scrumin ja visuaalisen työskentelytavan Kanbanin. Kuvausmenetelmistä käydään läpi Use Case -kaaviot, jotka havainnollistavat järjestelmän toimintaa eri käyttäjien näkökulmasta.

2.3.1 Ketterät menetelmät

Scrum on ketterä kehitysmenetelmä, jota käytetään tuotteiden ja palveluiden kehittämiseen. Alussa luodaan tuotteen kehitysjono (product backlog), joka on priorisoitu lista ominaisuuksista ja kyvykkyyksistä, mitä tuote tarvitsee menestyäkseen. Scrumissa työskennellään lyhyissä, rajatuissa jaksoissa (sprint), jotka kestävät yleisesti noin viikosta kuukauteen. Jokaisen jakson alussa tiimi valitsee tuotteen kehitysjonosta tehtäviä ja pilkkoo ne konkreettisiksi työtehtäviksi. Nämä työtehtävät yhdessä suunnitelman kanssa muodostavat sprintin kehitysjonon (sprint backlogin), jonka pohjalta tiimi toteuttaa jakson aikana tuotteelle uusia ominaisuuksia. (Rubin 2013, luku 19) Jokaisen jakson lopussa pidetään jaksokatselmointi (sprint-review), jossa kehitystiimi ja sidosryhmät arvioivat kehityksen tuloksia. Katselmointi tarjoaa mahdollisuuden tarkastella tuotteen nykytilaa, saada palautetta ja tehdä tarvittavia muutoksia tuotteen kehitysjonoon. (Rubin 2013, luku 21) Yhdessä projektissa on tietty määrä jaksoja, joita tiimi toistaa uudestaan, jatkuvasti mukautuen palautteeseen ja muuttuneisiin vaatimuksiin. Tämä tekee scrumista tehokkaan ja joustavan tavat hallita hankalimpiakin kehitysprojekteja. (Rubin 2013, luku 1)

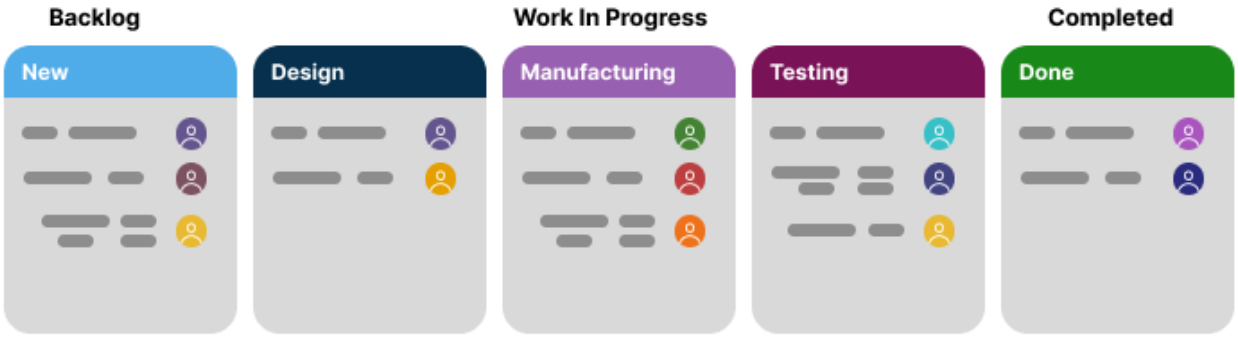
Tuotteen kehitysjono on Scrum-kehityksessä käytettävä priorisointi lista kehitettävän tuotteen ominaisuuksista. Sen avulla kehitystiimi näkee mitä ominaisuuksia tehdään ja missä järjestyksessä. Kehitysjonon rakenne on joustava, jossa lähiajan tehtävät ovat tarkemmin määriteltyjä, kun taas myöhemmät tehtävät ovat suurempia ja ennen toteutusta ne pilkottiin pienempiin osiin. Näin varmistetaan, että jaksoon valitut tehtävät ovat toteutuskelpoisa, jotka eivät vaadi sen suurempaa suunnittelua enää. (Rubin 2013, luku 6)

Jokaista tuotteen kehitysjonon tehtävää kutsutaan kehityskohteeksi (Product backlog item). Suurin osa näistä on ominaisuuksia, jotka tuovat jotain konkreettista asiakkaalle tai tuotteen käyttäjälle. Ne voivat olla myös virhe korjauksia, teknisiä parannuksia tai mitä tahansa tuotteen kehittämiseen liittyvää työtä. (Rubin 2013, luku 6)

Kanban on Lean-ajatteluun perustuva menetelmä prosessien parantamiseen, joka on adoptoitu ohjelmitokehitykseen valmistusteollisuudesta. Sen tarkoitus on parantaa kehitystiimin työ tapaa. Kanban-taulu on työkalu, jolla tehdään projektin eri työvaiheet visuaalisiksi. Se keskittyy suurempiin työalueisiin, kuten ominaisuuksiin verrattuna perinteisempään tehtävätauluun (Task board). (Stellman & Greene 2014, luku 9)

Scrumban on Scrumin ja Kanbanin yhdistelmä. Se yhdistää Scrumin projektinhallinnan ja tuotteen kehityksen vaiheet visuaaliseen Kanban-tauluun. (Stellman 2019) Kuvan 3 Scrumban-taulu on jaettu kolmeen pääosaan: kehitysjono (Backlog), työn alla (Work In Progress) ja valmiit (Completed). Backlog sisältää kaikki suunnitellut tehtävät, joista tiimin jäsenet valitsevat yleensä

yhden kerrallaan työn alle. Work In Progress jaetaan useisiin sarakkeisiin, jotka kuvaavat tehtävien etenemistä prosessin eri vaiheissa. Kun tehtävä on valmis, siirretään se Completed-osioon, jossa on kaikki tehtävät, jotka tiimi katsoo valmiiksi. (teamhood s.a.)

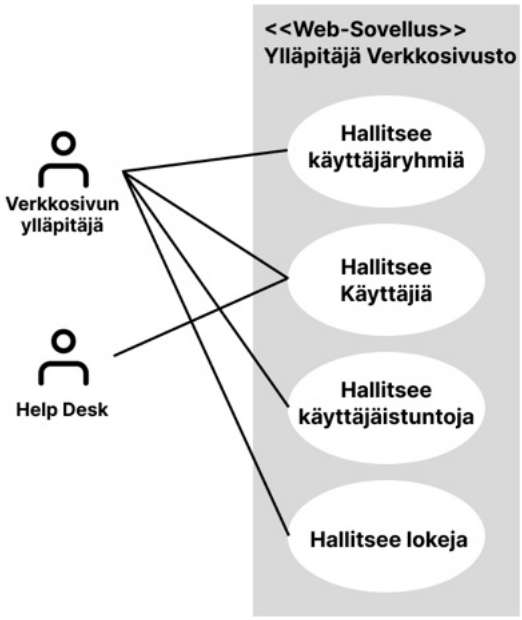


Kuva 3. Esimerkki Scrumban-taulusta (mukaillen teamhood s.a.)

2.3.2 Kuvausmenetelmät

Use Case eli käyttötapaus kuvaa, miten käyttäjä käyttää järjestelmää saavuttaakseen tietyn tavoitteen. Sen avulla voidaan määrittää järjestelmän toiminalliset vaatimukset. Käyttötapaukseen kuuluu toimija, joka suorittaa toiminnon ja skenaario, joka kuvaa kuinka käyttäjä saavuttaa tavoitteen. Toimijat ovat rooleja, jotka ovat vuorovaikutuksessa sovelluksen kanssa, esimerkiksi henkilö tai järjestelmä. (Larman 2004, luku 6)

Käyttötapauskaavio on yksi UML(Unified modeling language) mallinnuksen muoto. Sillä havainnollistetaan käyttötapausten ja toimijoiden väliset suhteet. (Larman 2004, luku 6) Kaaviossa voidaan käyttää yleisesti neljää eri suhdetta: "association relationships", joka yhdistää käyttötapaukset ja toimijat. "Generalazition relationships", joka kuvaa toimijoiden hierarkiaa. "Include relationships", jossa yksi käyttötapaus on toisen käyttötapauksen käytössä. "Extend relationships", jossa käyttötapaus laajentaa toisen käyttötapauksen käyttämistä. (IBM 2023) Kuvan 4 käyttötapauskaavio esimerkki esittää ylläpitäjäverkkosivuston toimintoja ja niiden suhteen toimijoihin.



Kuva 4. Esimerkki Web-sovellus Use Case -kaaviosta (mukaillen Fakhroutdinov s.a)

2.3.3 Git ja Github

Git on ilmainen, avoimeen lähdekoodiin perustuva versionhallintajärjestelmä, joka helpottaa ohjelmistokehitystä tarjoamalla tavan hallita ja jakaa muutoksia haluamallaan tarkkuudella. Se tekee haarojen (branch) luomisen, yhdistämisen ja poistamisen helppoa ja vaivatonta. Perinteisiin versionhallintajärjestelmiin verrattuna, se antaa käyttäjille täyden hallinnan sisältöönsä, jopa historian muokkaamiseen tarvittaessa. Git käsittelee sisältöä hakemistorakenteiden tasolla, mikä tekee hankalankin projektin hallinnasta kätevää. Tämän takia suuryritykset ympäri maailmaa suosivat sen käyttämistä. (Laster 2017, luku 1)

Github on suosittu pilvipohjainen alusta Git-repositorioiden ylläpitoon ja se on merkittävä työkalu Git-ekosysteemissä. Se mahdollistaa koodin säilyttämisen, hallinnan ja jakamisen repositorioina. Se tarjoaa myös paljon tämän lisäksi, kuten virheiden seurantajärjestelmiä (Issues, kuva 13) ja tiimityöskentelyn hallintaa (Projects), minne voi luoda oman tuotteen kehitysjonon (Product backlogin). Githubissa käyttäjät voivat kopioida olemassa olevia projekteja, tehdä muutoksia ja ehdottaa niiden yhdistämistä alkuperäiseen projektiin (Pull request). Tämä on muodostanut suuren yhteisön Githubin ympärille. (Laster 2017, luku 1)

3 Verkkosivuston luonti

Sovelluksen tarkoitus on vastata tarpeisiin saada käyttäjäystävällinen ja moderni drinkkireseptisivusto. Sen on tarkoitus olla hyvin skaalautuva, jota pystyy käyttämään millä laitteella tahansa.

Sovellus on tehty kaikille täysi-ikäisille. Se pyrkii innostamaan tavallisten ihmisten drinkkikulttuuria lähestyttävällä tyylillään ja käyttäjäystävällisyydellään. Sen tarkoitus on myös herättää kokeneempien drinkkiharrastajia kokeilemaan uusia reseptejä jo olemassa olevista ainesosista. Ja viimeisenä sen tarkoitus on antaa inspiraatiota ammattilaisille ja tarjota tulevaisuudessa alustan drinkkilistan luomiseen.

Sovelluksen toinen osa “arvostelut” on myös tarkoitettu kaikille, joita kiinnostaa kuulla valittujen harrastelijoiden mielipiteitä vaihtelevista alkoholituotteista. Näiden tarkoitus on saada kohderyhmä löytämään uusia tuotteita.

Projektissa on käytetty Scrum-projekntihallintaa, jossa työ on jaettu neljään sprinttiin. Jokaisen sprintin pituus on kaksi viikkoa, mikä pakottaa priorisoimaan ominaisuuksia, sekä kehitystapoja. Kehittämisen aikana on paljon uutta opittavaa, mikä voi hidastaa työn etenemistä.

Sovelluksen laadullisia kriteereitä on monia. Ensimmäisenä on vastata käyttäjien tarpeisiin, luomalla käyttöliittymä, jossa on helppoa ja sulavaa selata drinkkireseptejä ja arvosteluita. Toisena on luoda toimiva ja selkeä käyttöliittymä ylläpitäjille, joka on suojattu autentikoinnilla. Kolmantena on kehittää produktista saavutettava ja responsiivinen, jolloin se tarjoaa hyvän käyttäjäkokemuksen eri laitteilla. Viimeisenä on tarkoitus luoda mahdollisimman hyvää ja selkeää koodia, joka tekee jatkokehittämisestä miellyttävämpää.

Projekti on jaettu neljään kahden viikon sprinttiin, joista ensimmäinen kuluu produktin suunnitteluun ja vaatimusmäärittelyyn. Sprintti nollassa dokumentoidaan käyttäjävaatimukset ja alustetaan Next.js-projekti.

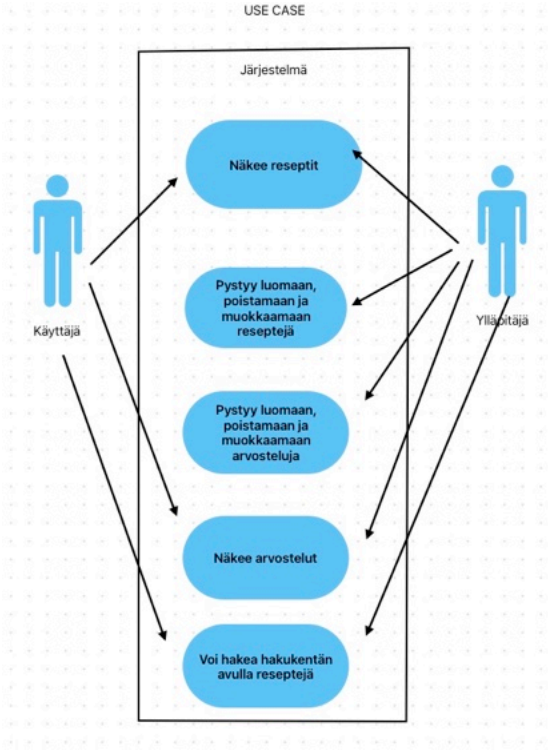
Sprintti ykkösellä toteutettiin ylläpitäjän käyttöliittymä ja siihen tarvittavat CRUD-toiminnot drinkkireseptien hallintaan. Integroidaan MongoDB tietokanta osaksi projektia käyttäen Prismaa sekä MongoDB Atlas pilvipalvelua. Lisäksi luodaan ylläpitosivuille autentikaatio käyttäen Clerkiä. MongoDB Atlas sekä Clerk ovat Saas palveluita eli Software as a Service, mikä tarkoittaa ohjelmiston tarjoamista palveluna internetin kautta. SaaS on eräänlainen pilvipalvelu, jossa palveluntarjoaja huolehtii ohjelmiston ylläpidosta, hallinnasta ja päivityksistä.

Sprintti kakkosella keskityttiin asiakkaan käyttöliittymään ja luodaan siihen tarvittavat komponentit. Komponenttien luomisen avuksi otetaan käyttöön NextUI-komponenttikirjasto, josta saadaan mm. navigointipalkki ja korttielementti drinkeille. Lopussa palataan vielä ylläpidon puolelle ja luodaan tarvittavat palvelin toiminnot, joilla voidaan hallita arvosteluja. Ja viimeiseksi luodaan sivut, jossa esittää näitä arvosteluita.

Sprintti kolmosella keskitytään asiakaslähtöiseen testaamiseen sekä ongelmien havainnointiin ja korjauksiin. Tässä vaiheessa lisätään kaikki löydetyt ongelmat Github issues-sivulle, josta näkee helposti, mitä kaikkea tarvitsee korjata. Näiden sekä käyttäjäpalautteen pohjalta tehdään korjauksia ja parannuksia. Lisäksi lisätään ylläpidon käyttäjäkokemuksen parantamiseksi Toastify, joka ilmoitta suoraan selaimessa tapahtuneista tapahtumista.

3.1 Sprint 0

Sprint 0 aikana keskityttiin vaatimuksien määrittelyyn sekä suunnitteluun. Vaatimuksien kuvaamiseen luotiin käyttötapauskaavion. Kyseinen kuvan 5 kaavio oli alustavassa suunnittelussa yksinkertainen, mutta kertoo järjestelmän mallista.



Kuva 5. Use Case -kaavio.

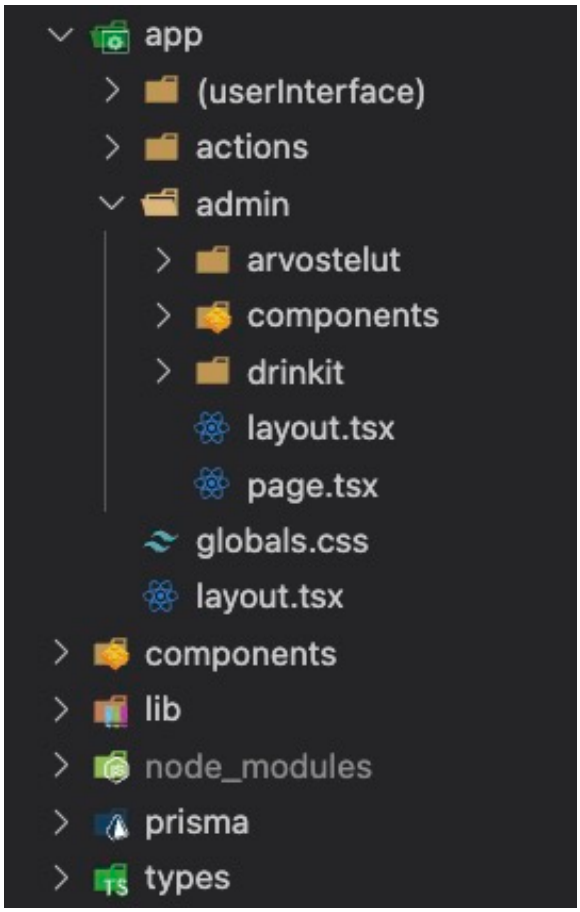
Luotiin Github projekti, johon kerättiin kaikkia vaatimukset tuotteen kehitysjonoon. Tämän kehitysjonon kehityskohteet olivat käyttäjätarinoita, joiden sisään on määritelty tehtävät asiat pienemmiksi

osiksi. Kehitysjonossa on kolme kohtaa: tehdä (Todo), Työn alla (In Progress) sekä valmis (Done). Projektin alussa työn alla oli vain projektin luominen ja alustaminen sprintti ykköstä kohti.

Product item:ssa on listattu tehtävät, jota kehittäjän täytyy suorittaa, jotta kyseinen käyttäjätarina toimii järjestelmässä. Kuvan 6 esimerkissä käydään läpi, mitä kaikkea tulisi tehdä, että ylläpitäjä pystyy näkemään ja hallitsemaan arvosteluita.

Kuva 6. Product item.

Seuraavaksi luotiin projekti komennolla npx create-next-app@latest. Tässä versiossa käytetään Next.js:n versiota 14.2.14. Tämän jälkeen poistettiin kaikki turha alussa tuleva koodi ja tehtiin valmiiksi muutama kansio.



Kuva 7. Sovelluksen projektipuu

Tämä projektipuu on kuvattu projektin loppuvaiheilta, mutta kaikki siinä on tehty, niin kuin oli suunniteltu. Kuvassa 7 app-kansion sisään tulee kaikki sovelluksen ydintoimintoihin liittyvät kansiot ja tiedostot. UserInterface-kansio sisältää kaikki käyttäjien käyttöliittymään vaadittavat asiat. Se sisältää polut (Paths) arvosteluille sekä drinkeille ja siellä on komponenttikansio missä on yksi komponentti aineksien esittämiseen drinkkireseptissä. Seuraavaksi on layout.tsx-tiedosto, jonka avulla voidaan jakaa käyttäjäsivujen yhteisiä rakenteita sen lapsikomponenteille, kuten tässä tapauksessa navigointikomponentti. Viimeisenä on page.tsx-tiedosto, koska userInterface-kansio on nimetti sulkujen sisään, ei siitä tule uutta reittiä, joten vaikkei tämä tiedosto ole suoraan /app-juuressa niin on se silti pääsivusto (Landing page).

Actions-kansiosta löytyy kaksi tiedostoa: drink.actios.ts ja reviews.actioms.ts. Ensimmäisestä löytyy kaikki sovelluksen tarvittavat toiminnot drinkkireseptien hallintaan, kuten luominen (createDrink), poistaminen (deleteDrink), ja hakeminen (getDrink). Toisesta tiedostosta löytyy kaikkia samoja toimintoja, mutta arvosteluille.

Admin-kansio on rakenteeltaan lähes identtinen kuin userInterface. Komponenttikansiossa on paljon enemmän komponentteja, kuten drinkin poistonappia ja drinkin luomislomaketta. Drinkit-

kansion sisällä oleva [slug]-kansio esittää yhden tietyn drinkin polkua ja sen sisällä oleva page.tsx-tiedosto esittää sen. Näin saamme esimerkiksi margarita-reseptin sivulle /drinkit/margarita.

Globals.css-tiedostoon määritellään globaalit tyylit, jotka pätevät koko sovellukseen. Tänne määriteltiin vain sovelluksen pääväritys sekä fontti, sillä tyylit luodaan pääosin Tailwind.css-klirjastolla.

Projektin juuren layout.tsx-tiedostoon on lisätty työkaluja, jotka ovat käytössä koko sovelluksessa, kuten NextUI-kirjasto sekä Toastify.

Projektin ulkopuolella oleva komponenttikansio sisältää uudelleenkäytettäviä komponentteja, joita hyödynnetään sekä käyttäjä- että ylläpitäjäkäyttöliittymässä. Tällaisia on mm. DrinkList.tsx-komponentti, joka esittää listana kaikki DrinkCard-komponentit, joka on toinen kansiosta löytyvä komponentti.

Lib-kansioon tulee apufunktioita, joita käytetään usein sovelluksen eri osissa. Esimerkki tästä on utils.ts-tiedosto, jossa funktiot muotoilevat tiedon haluttuun tyyliin.

Types-kansiossa on index.ts-tiedosto, jossa on TypeScriptin tyyppimäärittelyt. Se on kätevä tapa keskittää kaikki sovelluksen tyypit yhteen paikkaan.

3.2 Sprint 1

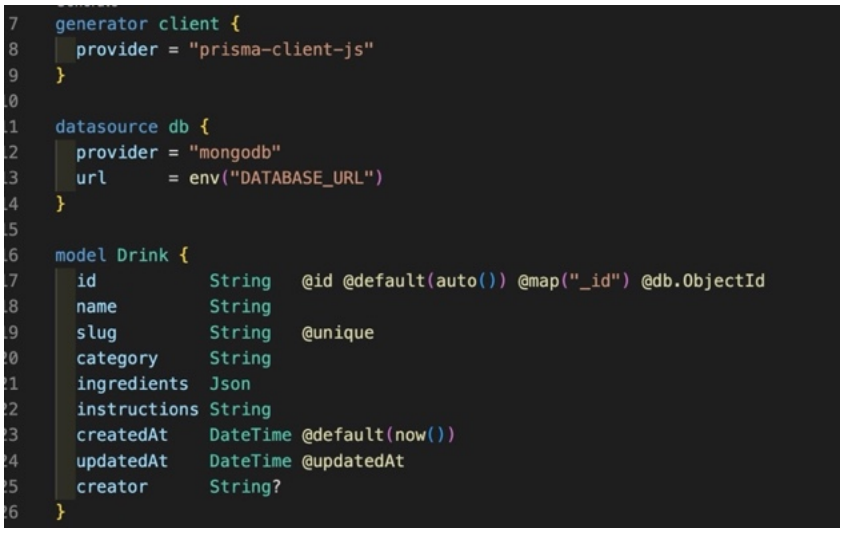
Tarkoituksena sprintti 1:llä oli luoda verkkosivun toiminnalliset ominaisuudet ja tietokanta. Luotiin ylläpitäjille käyttöliittymä sekä mahdollisuudet drinkkireseptien hallintaan. Sovelluksen ylläpitäjille tarkoitettuihin toiminnallisuuksiin pääsee vain autentikoinnin kautta.

Ensimmäisenä luotiin uusi projekti ja siihen klusteri (Cluster) MongoDB Atlas sivulla, joka on pilvipohjainen tietokantapalvelu. Klusteria luodessa saat tietokantaan liittyvän URL-osoitteen (DATABASE_URL), joka pitää lisätä oman projektin .env-tiedostoon.

Sprintin alkuun ladattiin prisma ja prisma/client paketit. Tämän jälkeen pystyttiin luomaan prisma skeema ja otettiin yhteys MongoDB serverille.

Pienenä huomioitavana next.js ja prisman käytössä, että kun luo prisman "PRISMA INIT" komento luo valmiiksi .env-tiedoston mihin se tarvitsee tietokannan URL-osoitteen. Tämä .env tiedosto ei ole kuitenkaan automaattisesti next.js gitignore-tiedostossa, koska yleisesti next.js käyttää gitignore.local. Helpointa on siis lisätä vain gitignore-tiedoston listaukseen .env-tiedosto. ”Prisma init”-komento luo myös prisma nimisen kansion ja sinne schema tiedoston, johon lisättiin kaikki haluamat tietokanta rakenteet helposti. Kuvassa 8 ensimmäisenä on prisma clientin asetukset.

Datasource db on tietolähteen valinta, eli tässä tapauksessa käytettiin mondoDB:tä. Model-attribuuttissa kerrotaan Drink-taulun tietorakenne.



```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

model Drink {
  id           String   @id @default(auto()) @map("_id") @db.ObjectId
  name         String
  slug         String   @unique
  category     String
  ingredients  Json
  instructions String
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  creator      String?
}
```

Kuva 8. Schema.prisma-tiedosto ja drinkin malli

Drinkki-taulun id-kentän avulla jokainen drinkki pystytään tunnistamaan yksilöllisesti.

Ingredients on suurin syy, miksi projektissa käytetään dokumenttipohjaista tietokantaa, eikä perinteisiä taulukkoja. Drinkin ainekset halutaan kuvattavaksi avain:arvo (key:value) pareiksi. Koska jokaisessa drinkissä on eri aineksia ja niiden määrät vaihtelevat, taulukoiden käyttämisestä tulisi monimutkaista. MongoDB:n avulla tämä on erittäin joustavaa. Json muodossa aineosat näyttävät tältä:

{
“Vodka”: “50ml”,
“Lime mehu”: “20ml”,
“Sokeri siirappi”: “10ml”
}

Erikoisuutena mallista löytyy slug, jonka avulla saadaan drinkkien URL-osoitteista houkuttelevimpia, kuin käyttämällä id:tä. Slug on URL-osoitteen lyhyt ja selkeä tekstiosa, kuten sivun otsikko, jota hyödynnetään järjestelmissä, jossa sivuja luodaan automaattisesti (Swartz 2013, luku 14). Esimerkkinä mojiton URL-osoite on /drinkit/mojito, eikä /drinkit/3292904. Koska slugia käytetään URL:ssä, sen täytyy olla uniikki. Sen käytössä tarvitsee ottaa myös huomioon välilyönnit ja skandimerkit, jotka eivät toimi perinteisessä URL-osoitteessa. Tämän takia drinkin luonnin yhteydessä slug luodaan muokkaamalla drinkin nimestä URL-ystävällinen. Esimerkki näkyy alempana

updateDrink funktiossa. Tämän takia “vodka redbull” onkin “vodka-redbull”. Kysymysmerkki kohdassa “creator String?”, tarkoittaa, että tämän tiedon lisääminen on vapaaehtoista. Tässä kohtaa se on siellä tulevaisuutta varten.

Tietokanta luotiin komennolla ”npx prisma db push”, jonka jälkeen se ilmestyi näkyviin MongoDB Atlas-sivulle. Samaa komentoa käytetään myös, kun halutaan päivittää prisma skeemaa ja tietokannan taulukkoa. Tietokannan päivittäminen on siis tehty hyvin helpoksi. Esimerkiksi “needsShaker”-boolean lisätään skeemaan ja ajetaan sama ”npx prisma db push”-komento, jolloin se päivittyy tietokantaan. Skeeman ja tietokannan päivittäminen saattaa toisinaan tuhota kaikki aikaisemmin tallennetut tiedot taulukosta. Prisma studio on kätevä selaimeen aukeava sivu, missä näkyy tietokanta taulut ja sinne voi manuaalisesti syöttää tietoa. Se aukeaa komennolla “npx prisma studio”. Next.js:n kehitysvaiheessa hot reload voi aiheuttaa useiden PrismaClient-instanssien luomisen, mikä voi johtaa tietokantayhteyksien ylikuormitukseen ja varoitukseen konsolissa. Ratkaisu on alustaa PrismaClient vain kerran ja tallentaa se globalThis-objektiin, jolloin samaa instanssia käytetään uudelleen. (Prisma s.a.) Kyseinen objekti on lib-kansion db.ts-tiedostossa, ja jatkossa on tärkeä muistaa, että PrismaClient tuodaan sieltä, eikä Prisman omasta kirjastosta.

Seuraavaksi Luotiin ylläpitäjän käyttöliittymä. Next.js:ssä sivujen luominen on yksinkertaista. Sovelluksen juuressa luotu kansion nimi on samalla polku ja verkkosivun URL-osoite. Tähän kansioon luotu page.tsx sivusto esittää polun näkymän.

Sivulla /admin/drinkit halutaan esittää kaikki drinkit listana. Tähän tarvitaan funktio, joka hakee tietokannasta kaikki drinkit. Tätä varten luotiin actions kansioon actions.drinks.ts tiedosto, johon lisätään kaikki drinkkien käsittelyyn tarvittavat funktiot. Prisma tekee näistä funktioista erittäin yksinkertaisia. Tässä funktio, joka hakee tietyn drinkin sen uniikin URL-nimen (Slug) mukaan. Tässä voitaisiin hakea myös id:n kanssa.

```
export async function getDrink(slug: string) {
  const drink = await prisma.drink.findUnique({
    where: {
      slug: slug,
    },
  });
  if (drink == null) return notFound();
  return drink;
}
```

Kaikki tämän tyyppiset palvelinpuolen toiminnot (server actions) ovat actions-kansiossa.

Luotiin alustava lomakekomponentti drinkkien käsittelyyn. Tätä komponenttia en enempää näytä, koska se tulee muuttumaan sprintti kakkosen aikana. Tämä komponentti tarvitsi myös omat funktiot ”luo” ja ”muokkaa”. Koska drinkin ainekset ovat json objekti key : value muodossa ainesosan nimi : ainesosan määrä, tarvitaan lomakkeen sisään lomake, joka lisää näitä objekteja itse reseptiin. Kaikkiin sovelluksen CRUD-menetelmät tarvitsevat palvelinpuolen toimintoja, ja prismaa hyödyntämällä datan hallinta tietokannan kanssa on helppoa. Alla oleva esimerkki muokkaa drinkkireseptiä.

```
export async function updateDrink(
  id: string,
  formData: FormData
): Promise<UpdatedDrinkResult> {
  const name = formData.get("name") as string;
  const category = formData.get("category") as string;
  const instructions = formData.get("instructions") as string;

  const ingredientsData = formData.get("ingredients") as string;
  const ingredients = JSON.parse(ingredientsData);

  if (!name || !category || !ingredientsData || !instructions) {
    return { error: "Kaikki kentät ovat pakollisia." };
  }

  const slug: string = name
    .toLowerCase()
    .replace(/ä/g, "a")
    .replace(/ö/g, "o")
    .replace(/\s+/g, "-");

  try {
    const updatedDrink = await prisma.drink.update({
      where: { id },
      data: {
        name,
        slug,
        category,
        ingredients,
        instructions,
      },
    });
    await Promise.all([
      revalidatePath(`/drinkit`),
      revalidatePath(`/admin/drinkit`),
      revalidatePath(`/drinkit/${slug}`),
      revalidatePath(`/admin/drinkit/${slug}`),
    ]);

    return { data: updatedDrink };
  } catch (err: any) {
    console.error("Virhe drinkkiä päivittäessä:", err);
    return { error: err.message || "Drinkin päivitys epäonnistui." };
```

```
  }
}
```

Tässä koodiesimerkissä funktio ottaa vastaan muokattavan drinkin id:n sekä lomakkeeseen syötetyt tiedot. Lomakkeen tiedot tarkistetaan, ettei mikään ole jäänyt tyhjäksi ja tämän jälkeen luodaan drinkin URL-nimi. Lopuksi muutokset tallennetaan prsima.drink.update funktiolla, jossa valitaan oikea drinkki id:n avulla ja päivitetään tiedot uusiksi. Tämän jälkeen kaikki, missä kyseinen drinkki näkyy verkkosivuilla, päivitetään revalidatePath funktiolla. Onnistuneen päivityksen jälkeen palautetaan vielä selaimelle päivitetty drinkki, jotta se voidaan heti esittää käyttäjälle sekä mikäli URL-osoite on vaihtunut niin ei tule “page not found”.

Komponenteissa, kuten luomislomakkeessa, jossa tarvitaan käyttäjän syöttämää tietoa, käytetään hydratoitua sivun tuottamista. Tämä tarkoittaa, että Next.js ei renderöi pelkästään staattista HTML-sisältöä palvelimella, vaan lisää myös JavaScript-funktiot, jotka tekevät käyttöliittymästä interaktiivisen selaimessa. Tämä on Next.js:ssä tehty hyvin helpoksi, lisäämällä vain tiedoston ensimmäiselle riville "use client", joka määrittää komponentin client-komponentiksi. Tällöin tarvittavat toiminnot voidaan suorittaa selaimessa. (Next.js s.a. a)

Seuraavaksi lisättiin projektiin Clerk authentication. Se on erittäin yksinkertainen ja nopea tapa lisätä autentikaatio sovellukseen. Ensimmäisenä luotiin sovellus Clerkin verkkosivuilla. Heti ensimmäisenä Clerk antaa monia vaihtoehtoja, millä eri vaihtoehdoilla käyttäjä voi kirjautua sisään, kuten sähköposti, google-tili tai puhelinnumero. Tässä projektissa ainoa tapa kirjautua on Clerkin sivuilla sallimien käyttäjien google-tileillä. Sitten asennettiin clerk/nextjs-kirjasto npm:n (Node package manager) avulla ja lisätään .env-tiedostoon Clerkin antamat API-avaimet. Lopuksi luodaan middleware.ts-tiedosto projektin juureen ja lisätään sinne Clerkin antama koodipätkä.

Clerkin tapaisissa Saas palveluissa asiakas maksaa vain käytön mukaan. (Prithviraj 2015, luku 1) Koska projektissa ei tarvita käyttäjätunnuksia, kun ylläpitäjille, ei tämä maksa mitään.

Kun Clerk on asennettu projektiin, sen suojelemat komponentit ja sivut upotetaan Clerk-komponenttien sisään lapsiksikomponenteiksi. Tässä koodiesimerkissä näkyy kuinka kaikki ylläpitäjänsivut näkyvät, jos käyttäjä on kirjautunut sisään. Jos käyttäjä ei ole kirjautunut, ohjataan hänet kirjautumissivulle. Sisään pystyy kirjautumaan vain ne, jotka on erikseen lisätty Clerkin sivuilla käyttäjiksi. Clerkin avulla saadaan turvallinen suojaus sivuilla helposti ja nopeasti.

```
const AdminLayout = ({ children }: { children: React.ReactNode }) => {
  return (
    <ClerkProvider>
      <SignedIn>
        <AdminNavbar />
```

```
      {children}
    </SignedIn>
    <SignedOut>
      <RedirectToSignIn />
    </SignedOut>
  </ClerkProvider>
 );
};
export default AdminLayout;
```

Tämä koodipätkä on /admin-kansion layout.tsx-tiedostosta, joka pätee jokaiseen /admin-sivustoon. Tämän avulla ei ulkopuoliset pääse ylläpitäjien sivuille.

3.3 Sprint 2

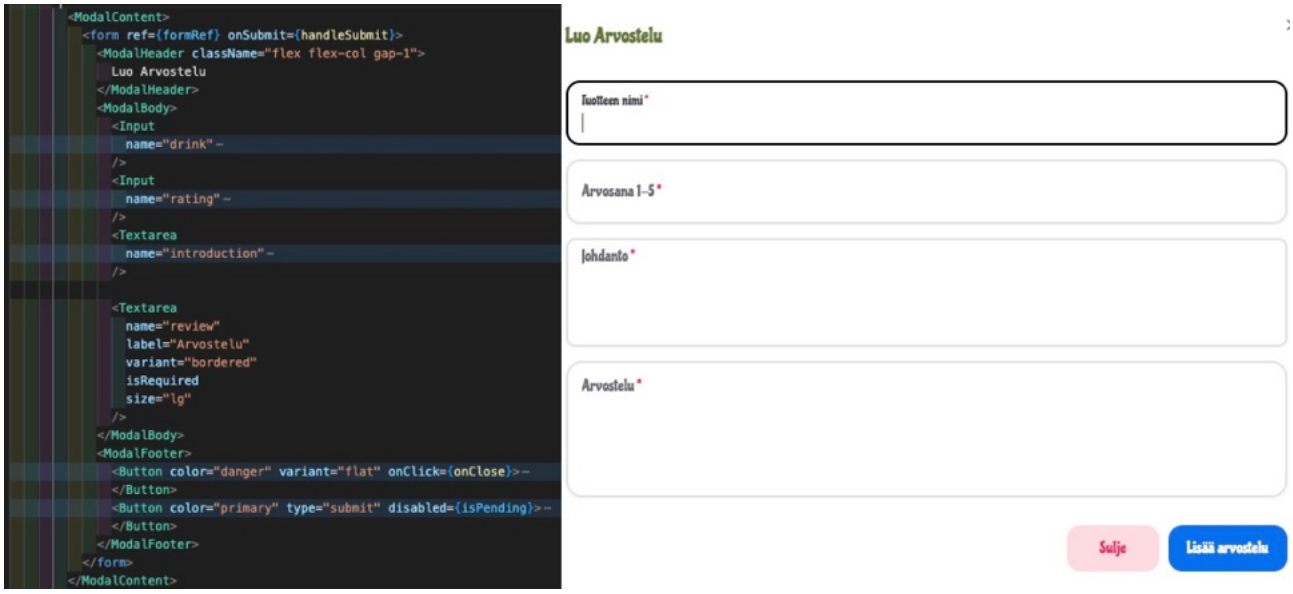
Sprintti 2:sen tavoite oli tehdä asiakasnäkymä drinkeille. Tämän lisäksi tehtiin "arvostelut" osio, johon voi luoda arvosteluja erilaisista juotavista.

Tässä vaiheessa otettiin käyttöön NextUi-komponentti kirjasto, mikä tarjoaa valmiita komponentteja ohjelmistokeitykseen. Helpoin tapa käyttää kirjastoa on ladata nextui-cli globaalisti tietokoneelle. Tämän ansiosta yksinkertaisella ”nextui add navigation”, projektiin latautuu navigaatio komponentti, sekä se listään tailwind.config.ts tiedostoon automaattisesti. Tailwind.config.ts tiedostoon, joka löytyy sovelluksen juuresta, täytyy alussa lisätä myös “plugins: [nextui()]”. Viimeisenä NextUIProvider täytyy lisätä sovelluksen juureen toimiakseen (kuva 9).

Kuva 9. Sovelluksen juuren näkymä

Sovellus käyttää navigointikomponenttia, jotka luotiin erikseen käyttäjille ja ylläpitäjille. Monimutkaisuuden estämiseksi sekä ymmärrettävyyden lisäämiseksi komponentit luotiin erikseen. NextUI:n navigointi tukee suoraan mobiilinäkymää, jossa navigointi muuttuu hampurilaisvalikoksi.

Kuvan 10 juoma-arvostelujen luontiluomakkeessa käytettiin NextUI:n valmiita komponentteja, jotka osoittautuivat niin käteväksi, että käytiin muokkaamassa drinkkireseptien lomakkeet samanlaisiksi. Tämän lomakkeen sisältö on luotu täysin Next:Ui:n komponentteja käyttäen, kuten Input, Textarea ja Button. Kuvan 10 koodipätkässä näkee esimerkkejä, mitä erilaisia ominaisuuksia voidaan antaa Textarea-komponentille. Tässä on kuitenkin vain murto-osa, mitä kaikkia ominaisuuksia siihen voi lisätä. Tämän tyyppisen valmiin komponentin lisääminen projektiin on nopeaa ja kehittäjäystävällistä.



Kuva 10. Juoma-arvosteluiden luontilomake ja sen koodipätkä

Juoma-arvosteluiden tarkoitus on esittää arvosteluita esilaisista tuotteista, kuten oluista ja viineistä. Kuvassa 11 näkyy juoma-arvosteluiden malli schema.prisma-tiedostossa.

```
model Review {
  id           String   @id @default(auto()) @map("_id") @db.ObjectId
  drink        String
  slug         String   @unique
  rating       String
  introduction String
  review       String
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  creator      String?
}
```

Kuva 11. Juoma-arvosteluiden malli.

Drink:llä tarkoitetaan arvosteltavaa tuotetta. Myöhemmin huomattu, että nimeäminen samaksi, kuin toinen malli, johti hämmennykseen. Rating on string, koska halutaan jättää mahdollisuus desimaali arvosanoille. Introduction on johdanto, mikä esitellään arvostelukortissa, ja review on itse juoma-arvostelu, joka jatkuu johdannon perään arvostelun omalla sivulla. Arvostelujen kohdalla otettiin jo käyttöön creator-muuttuja, eli otettiin Clerkin avulla ylös tietokantaan kyseisen arvostelun luoja. Clerkiltä saa käyttäjän tiedot käyttämällä auth funktiota:

```
  const { userId } = auth();

  let creatorName = "unknown";

  if (userId) {
    const user = await clerkClient.users.getUser(userId);
    if (user?.firstName) {
      creatorName = user.firstName;
    }
}
```

Tässä koodinpätkässä etsitään tämänhetkisen sisään kirjautuneen käyttäjän id, jos sitä ei ole, luojaksi jää “unknown”. Id:n löytyessä haetaan oikea käyttäjä se clerkClient.users.getUser-funktilla.

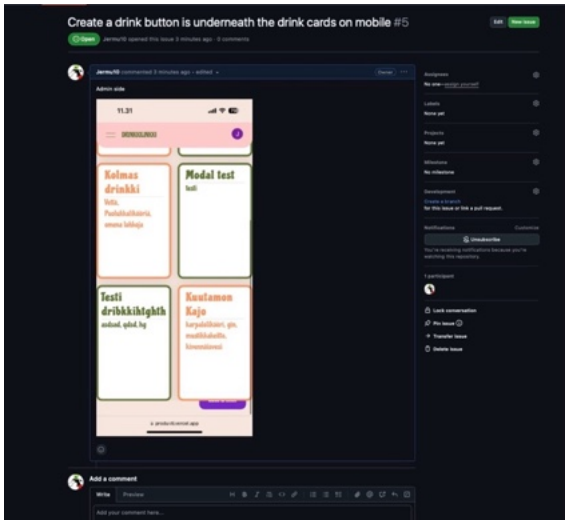
Sitten luotiin drinkeille korttikomponentti, käyttäen NextUI:n korttikomponenttia, joka esittelee drinkin nimen, aineosan ja kuvan (kuva 12). Kortin ympärykset ovat värikoordinoidut sen mukaan, onko kyseessä drinkki vai mocktail. Kuva tulee pysymään projektin aikana kaikissa samana, koska niiden ylläpito ei kuulu tämän opinnäytetyön alueeseen. Sama korttikomponentti tehtiin myös arvosteluille, joka esittelee arvostelun kohteen, johdannon, arvosanan, arvostelijan sekä arvostelun päivämäärän.

Kuva 12. Etusivu, jossa arvostelu- ja drinkkikortit.

Nämä kortit ovat pääroolissa käyttäjänäkymässä. Etusivulla esitellään viisi uusinta drinkkireseptiä ja arvostelua.

3.4 Sprint 3

Kolmannessa sprintissä testattiin sovellusta käyttäjillä. Käyttäjien ilmoitukset ongelmista kirjattiin Github repositorion onglema (Issues) kohtaan ja niitä käytiin läpi mitä tärkeysjärjestyksessä. Ensin kaikki toiminallisuuteen liittyvät häiriöt ja sitten vasta visuaalisuuteen liittyvät. Kuvassa 13 on esimerkki toiminallisen häiriön virheraportista.



Kuva 13. Github ongelma näkymä

Lisättiin Toastify, joka ilmoittaa suoraan verkkosivun ylälaidassa, kun jokin tapahtuma tapahtuu. Esimerkiksi, kun luodaan uusi drinkki, käyttäjä saa tiedon, että onnistuiko drinkin lisääminen. Kun Toastify-kirjasto on ladattu projektiin, se lisätään sovelluksen juureen (kuva 9). Tämän jälkeen sitä voidaan käyttää helposti, kuten drinkin poistamisen jälkeen:

toast.success(‘Drinkki poistettiin onnistuneesti”). Tämän avulla saadaan helposti ilmoitettua ylläpitäjälle, onnistuiko tietty tapahtuma, eikä hänen tarvitse mennä sitä selaimen konsolista etsimään.

Tämän osion lukemisen yhteydessä voi tutustua sovelluksen koodiin Github-repositorion kautta: https://github.com/Jermu10/produkti/

4 Pohdinta

Projektin päätyttyä olin tyytyväinen saavuttamiini tavoitteisiin, sekä oppimaani. Tärkeintä oli toimiva sovellus ja käsitys mitä tekisi ensikerralla toisin.

4.1 Tavoitteet

Suurin osa projektin alussa asettamista toiminallisista tavoitteista toteutuivat suunnitellusti. Projektin sprinttijakoon perustuva aikataulutus toimi hyvin, koska se mahdollisti työmäärän realistisen jakamisen ja selkeän etenemisenhallinnan.

Verkkosivuston ylläpitäjän ensimmäinen vaatimus reseptien hallinnasta toteutui suunnitellusti sprintti 1:llä. Ylläpitäjälle luotiin käyttöliittymä, jonka avulla he voivat luoda, muokata ja poistaa drinkkireseptejä. Toinen vaatimus, juoma-arvosteluiden hallinta toteutui sprintti 2:lla suunnitellusti. Viimeinen kolmas vaatimus, ylläpitosivujen suojaaminen Clerk-autentikoinnilla toteutui myös suunnitellusti sprintti 2:lla.

Verkkosivuston käyttäjille asettamista vaatimuksista suurin osa toteutui kuten oli suunniteltu. Ensimmäinen vaatimus, mahdollisuus selata drinkkireseptejä toteutui onnistuneesti sprintti 2:lla. Käyttäjät pystyvät tarkastelemaan valitsemansa drinkin ainesosia sekä ohjeita. Toinen vaatimus, juoma-arvosteluiden selaaminen toteutui myös sprintti 2:lla onnistuneesti. Kolmas vaatimus, hakukentän luominen drinkkireseptien selaamiseen jäi kokonaan toteuttamatta. Tämä epäonnistui aikataulullisista syistä, koska se oli alhaisimmalla prioriteetilla verrattuna muihin vaatimuksiin, eikä sitä päästy ollenkaan aloittamaan. Tämän lisäksi testaaminen jäi minimaaliin ja syy tähän on priorisoinnin väärin arvioiminen. Tästä “oppimani” luvussa.

Projektin teknologiset tavoitteet onnistuivat siinä määrin, että kaikkia alussa suunnitellut käytettäväksi tulevat työkalut ja teknologiat saatiin toimimaan yhdessä Next.js kanssa.

4.2 Oppimani

Full-stack kehityksen aikana oppiminen on suurta, sillä kaikki osa-alueet on hallittava, jotta voidaan saavuttaa näkyviä tuloksia. Kehitys edellyttää tarkkaa suunnittelua ja ymmärrystä siitä, mitä tekee ja miksi. Esimerkiksi pienikin muutos tietokantaan voi vaikuttaa koko sovellukseen, jolloin muutokset täytyy huomioida tietokannan lisäksi myös backend sekä frontend puolella. Kehittäessä täytyy jatkuvasti miettiä, että miten muutokset vaikuttavat minnekin ja mitä se tarkoittaa kokonaisuuden kannalta. Lisäksi asioiden nimeämiset kannattaa pohtia tarkasti. Sprintti 2:lla ilmeni hämmennystä, kun juoma-arvostelun itse juoman muuttuja (Variable) oli nimeltään ”drink”, mikä oli sama kuin drinkkireseptien luokka (Class).

Projektin aikana opin testaamisen tärkeyden jo itse kehitysvaiheessa. Testaaminen oli aikataulutettu viimeiseen sprinttiin, vaikka sen tulisi kulkea kehityksen rinnalla koko prosessin ajan, tämä on kouluprojekteissakin jäänyt vähälle huomille. Jokaisen uuden funktion luomisen jälkeen olisi hyvä kirjoittaa heti testit, jotta mahdolliset tulevat muutokset eivät rikkoisi olemassa olevaa toiminnallisuutta. Tämä nopeuttaa ja helpottaa kehittämistä merkittävästi. Työn aikana törmäsin, monesti virheisiin, joita en ollut huomannut, koska en ollut pienten muutosten jälkeen testannut kunnolla. Järjestelmällisen testauksen avulla monet ongelmat olisi voitu havaita jo aikaisemmin.

Vaikka projektissa käyttämät modernit työkalut ja teknologiat, kuten Next.js, Prisma ja Clerk, ovat tehty mahdollisimman kehittäjäystävällisiksi ja olivat minulle pintapäällisesti tuttuja, niiden kanssa kokonaisen projektin luominen erittäin opettavaista. Toinen virhe oli ottaa mukaan nextUI-komponenttikirjasto vasta kolmannessa sprintissä. Heti alussa tämän käyttöönotto olisi säästänyt aikaa, sillä omien komponenttien suunnittelu ja toteutus vei paljon aikaa. Tämän ajan olisin voinut käyttää verkkosivun toiminnallisuuksien kehittämiseen, mitä pidän projektin alussa tärkeämpänä.

4.3 Jatkokehitys

Verkkosivuston jatkokehittäminen tullaan aloittamaan uudella suunnitteluvaiheella, jossa määritellään uudet vaatimukset ja otetaan mukaan tästä työstä toteuttamatta jäänyt hakukenttä-vaatimus. Tämän hakukentän luomisen yhteydessä on luultavasti tulossa myös jonkinlainen suodatinvalikko, jossa verkkosivun käyttäjä voi esimerkiksi valita nähtäväkseen kaikki ginipohjaiset reseptit. Hakukenttä ja suodatinvalikko parantavat käyttäjäkokemusta merkittävästi, koska käyttäjät voivat helpommin löytää juuri heitä kiinnostavia reseptejä.

Yksi kehitysidea on tapahtumiin luoda lista kaikista tarjolla olevista drinkeistä, ns. menu. Tästä listasta näkisi hinnat, drinkkireseptit ja tapahtuman järjestäjä voisi päivittää sitä reaaliajassa. Toinen kehitysidea olisi saada drinkkiresepteihin ”suosio”-kohta, jossa verkkosivun käyttäjät voivat arvostella reseptin 1–5 arvosanalla.

Sivustoa ei ole alustavasti kaupallistettu, mutta mahdollisuudet sille on olemassa jatkokehityksessä. Ensimmäinen mahdollisuus on sivulla pyörivät kohdennetut mainokset. Tämän toteuttaminen vaatii GDPR:n (General data protection regulation) asettamien vaatimusten huomioimista käyttäjätietojen käsittelyssä sekä käyttäjien suostumuksen hankkimista kohdennetuille mainoksille. Toinen on mahdollisuus yhteistyösopimuksille, joissa reseptien ainesosina on kaupallisia tuotteita. Esimerkiksi "vodkan" sijaan olisikin tietyn brändin vodkaa.

Lähteet

Barklund, M. 2024. React in Depth. Manning Publications. E-kirja. Luettu: 3.12.2024.

Branshaw, S., Brazil, E. & Chodorow, K. 2019. MongoDB: The Definitive Guide. 3. painos. O'Reilly Media, Inc. Sebastopol. E-kirja. Luettu: 10.11.2024.

Bugl, D. 2024. Modern Full-Stack React Projects. Packt Publishing. Birmingham, UK. E-kirja. Luettu: 2.12.2024

Chernenko, M. 2024. Full-Stack Web Development with TypeScript. 1. painos. Packt Publishing Ltd. Birmingham, UK. E-kirja. Luettu: 10.11.2024.

DB-Engines. s.a. a DB-Engines Ranking. Luettavissa: https://db-engines.com/en/ranking/document+store. Luettu: 16.12.2024.

DB-Engines. s.a. b DB-Engines Ranking. Luettavissa: https://db-engines.com/en/ranking_categories. Luettu: 16.12.2024.

Fakhroutdinov, K. s.a. UML Use Case Diagram Examples. Luettavissa: https://www.uml-diagrams.org/use-case-diagrams-examples.html. Luettu: 27.12.2024.

GeeksforGeeks 2023. Server Side Rendering vs Client Side Rendering vs Server Side Generation. Viimeksi päivitetty: 27.3.2023. Luettavissa: https://www.geeksforgeeks.org/server-side-rendering-vs-client-side-rendering-vs-server-side-generation/. Luettu: 16.12.2024.

Gerchev, I. 2022. Tailwind CSS. SitePoint. E-kirja. Luettu: 1.12.2024.

IBM 2023 Rational Software Architect Use-case diagrams. Luettavissa: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-use-case. Luettu 3.11.2024.

Kouraklis, J. 2019. Introducing Delphi ORM: Object Relational Mapping Using TMS Aurelius. Apress. E-kirja. Luettu: 3.12.2024.

Larman, C. 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition. Pearson. E-kirja. Luettu: 9.12.2024.

Laster, B. 2017. Professional Git. Wrox, a Wiley Brand. Indianapolis, IN. E-kirja. Luettu: 18.11.2024.

MongoDB. s.a. Relational vs. Non-Relational Databases. Luettavissa: https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases. Luettu: 15.11.2024.

Next.js. s.a. a Next.js Client Components. Luettavissa: https://nextjs.org/docs/app/building-your-application/rendering/client-components. Luettu: 20.11.2024.

Next.js. s.a. b Next.js Introduction. Luettavissa: https://nextjs.org/docs. Luettu: 23.11.2024.

Next.js. s.a. c Next.js Showcase. Luettavissa: https://nextjs.org/showcase. Luettu: 23.11.2024.

Prisma. s.a. Best practice for instantiating Prisma Client with Next.js. Luettavissa: https://www.prisma.io/docs/orm/more/help-and-troubleshooting/help-articles/nextjs-prisma-client-dev-practices. Luettu: 12.11.2024.

Prithviraj, S. 2015. *Architecting Cloud SaaS Software*. Pearson India. E-kirja. Luettu: 8.11.2024.

Rippon, C. 2023. Learn React with TypeScript - Second Edition. Packt Publishing. E-kirja. Luettu: 1.12.2024.

Riva, M. 2022. Real-World Next.js. Packt Publishing. Birmingham. E-kirja. Luettu: 6.11.2024.

Rubin, K. S. 2012. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley Professional. Upper Saddle River, NJ. E-kirja. Luettu: 19.11.2024.

Sakhniuk, M. & Boduch, A. 2024. React and React Native - Fifth Edition. Packt Publishing. E-kirja. Luettu: 1.12.2024.

Stack Overflow 2024. Stack Overflow Developer Survey 2024: Most Popular Technologies – Web Frameworks. Luettavissa: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-webframe. Luettu: 19.10.2024.

Stellman, A. 2019. What Is Scrumban? O'Reilly Media, Inc. Sebastopol. E-kirja. Luettu: 23.11.2024.

Stellman, A. & Greene, J. 2014. Learning Agile. O'Reilly Media, Inc. Sebastopol. E-kirja. Luettu: 23.11.2024.

Swartz, A. 2013. Building for Users: Designing URLs. Teoksessa: *Aaron Swartz's A Programmable Web: An Unfinished Work*. *Synthesis Lectures on Data, Semantics, and Knowledge*. Springer, Cham. Luettavissa: https://doi.org/10.1007/978-3-031-79444-5_2. Luettu: 16.12.2024.

Teamhood s.a. 5 Scrumban Board Examples To Get You Started. Luettavissa: https://teamhood.com/agile-resources/scrumban-board-examples/. Luettu: 27.12.2024.

Vanderkam, D. 2024. Effective TypeScript. 2. painos. O’Reilly Media, Inc. Sebastopol. E-kirja. Luettu: 18.11.2024.