

Timo Suni

HEADLESS CMS-INTEGRAATIO NEXT.JS-PROJEKTISSA

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2024



**Kaakkois-Suomen
ammattikorkeakoulu**

Tutkintonimike	Liiketalouden ammattikorkeakoulututkinto
Tekijä	Timo Suni
Työn nimi	Headless CMS-integraatio Next.js-projektissa
Toimeksiantaja	Kiinteistö Oy Lahden Karisma (Kauppakeskus Karisma)
Vuosi	2024
Sivut	30 sivua
Työn ohjaaja	Janne Turunen

TIIVISTELMÄ

Tämän opinnäytetyön aiheena oli sovelluskehitysprojekti, jonka tavoitteena oli kehittää henkilökunnan perehdytysportaali kauppakeskusympäristöön. Perehdytysportaali tuli olla käytettävissä ajasta ja paikasta riippumatta sekä helposti ylläpidettävissä kauppakeskusjohdon toimesta. Työn toimeksiantajana on Lahdessa sijaitseva Kauppakeskus Karisma, jolla oli tarve päivittää henkilökunnan perehdytyskäytäntöjä sekä materiaaleja.

Opinnäytetyö rajautui käsittelemään headless sisällönhallintajärjestelmän integrointia osaksi Next.js-projektia. Projektissa kehitettiin perehdytysportaali toimeksiantajana toimineelle kauppakeskukselle. Opinnäytetyö on kehittämis-tehtävä ja siinä seurattiin portaalin kehittämistä alusta loppuun korostaen tarpeellisia kehitystyön osa-alueita.

Projektissa käytettiin Next.js-sovelluskehystä yhdessä Vercel-julkaisualustan kanssa. Työn keskeisenä teknologisena valintana oli headless CMS-palveluntarjoajan valinta, jossa päädyttiin käyttämään Sanitya. Sovelluskehitysprosessi kattoi sisällönhallintajärjestelmän tietomallien rakentamisen, sisällön dynaamisen reitityksen sekä rikkaan tekstisisällön muotoilun käyttäen Portable Text -teknologiaa. Kehitystyö tehtiin Visual Studio Code -ympäristössä, hyödyntäen Sanityn API-yhteyksiä ja GROQ-kyselykieltä sisällön hallintaan.

Lopputuloksena syntyi käyttövalmis perehdytysportaali, joka vastasi opinnäytetyön ja toimeksiantajan asettamiin tavoitteisiin. Sovellus tarjoaa pohjan jatkokehitykselle, johon voidaan sisällyttää käyttäjäpalautteen perusteella uusia ominaisuuksia. Opinnäytetyö antoi arvokasta tietoa modernin web-sovelluskehityksen haasteista ja ratkaisuksista, sekä vahvisti tekijän ammatillista osaamista tulevia urahaasteita varten. Työn tulokset korostavat uusien teknologioiden ja innovatiivisten ratkaisujen merkitystä käytännön haasteiden ratkaisemisessa, mikä edesauttaa henkilökunnan tehokasta perehdyttämistä ja parantaa organisaation toiminnan tehokkuutta.

Asiasanat: ohjelmistokehitys, sisällönhallinta, Next.js, Sanity, headless CMS

Degree title	Bachelor of Business Administration
Author	Timo Suni
Thesis title	Headless CMS Integration in a Next.js Project
Commissioned by	Kiinteistö Oy Lahden Karisma (Kauppakeskus Karisma)
Time	2024
Pages	30 pages
Supervisor	Janne Turunen

ABSTRACT

This thesis focused on the development of an application for onboarding staff in a shopping center. The primary objective was to integrate a headless content management system into a Next.js project, creating a user-ready application. This developmental task followed the journey of creating a staff onboarding portal from inception to completion, as necessary for the scope of this thesis. The project diverged slightly from the typical thesis format, incorporating theoretical underpinnings in various stages of the development process. The thesis began with an initiation into the development work and key concepts, followed by a focus on application development and various technological solutions and concluded with the deployment and final outcome of the work.

The methodological approach of the thesis was a practical development task, using a hands-on approach to application creation. The development process involved the selection and utilization of specific technologies, notably Next.js for the framework and Sanity as the headless CMS. The choice of these technologies was based on their compatibility, ease of integration, and their robust features for web application development.

The findings and outcomes of this thesis provide valuable understanding into the practical aspects and considerations involved in creating a contemporary web application for staff onboarding. The project successfully achieved its goal of creating an accessible, online staff onboarding portal that could be independently managed and updated by the shopping center's administration. The integration of a headless CMS and the choice of technologies proved effective in meeting the project's objectives.

Evaluating the success of the project, it can be concluded that the thesis met its objectives effectively. The application provides a valuable tool for the shopping center and showcases the practical application of theoretical concepts in software development. The project also offers insights for future development work, emphasizing the importance of careful planning and user feedback in the development of web applications.

Keywords: software development, content management, Next.js, Sanity, headless CMS

SISÄLLYS

1	JOHDANTO	5
2	KEHITTÄMISTYÖN ALOITUS	6
2.1	Next.js & Vercel	6
2.2	Staatinen sivugenerointi (SSG).....	7
2.3	Sisällönhallintajärjestelmä (CMS)	8
2.4	Headless CMS palveluntarjoajan valinta	10
3	SOVELLUSKEHITYSPROSESSI JA TEKNOLOGISET RATKAISUT	11
3.1	Kehitysympäristö ja Sanityn asentaminen	11
3.2	Sanity Studio ja skeemat	12
3.3	API yhteys ja GROQ.....	14
3.4	Tiedon esittäminen käyttäen Next.js dynaamista reititystä	17
3.5	Portable text -muotoilu	20
4	JULKAISU JA LOPPUTULOS	23
4.1	Sovelluksen julkaisu ja päivitys-hookit	23
4.2	Lopputulos	26
5	PÄÄTÄNTÖ	28
	LÄHTEET.....	30

1 JOHDANTO

Tämän opinnäytetyön aiheena on sovelluskehitysprojekti. Työn tavoitteena on luoda käyttövalmis sovellus kauppakeskuksessa työskentelevän henkilökunnan perehdyttämiseen. Opinnäytetyö rajautuu headless-sisällönhallintajärjestelmän integrointiin osaksi Next.js-projektia.

Opinnäytetyö on kehittämistehtävä, jossa seurataan perehdytysportaalin kehittämistä alusta loppuun niiltä osin, kuin työn rajauksen kannalta on tarpeellista. Opinnäytetyö etenee kehittämistarinan mukaisesti hieman normaalista opinnäytetyön tyylistä poiketen. Työn pohjalla olevaa teoriaa käsitellään kehitystyön eri vaiheissa projektin edetessä. Opinnäytetyön alussa paneudutaan kehittämistyön aloittamiseen sekä työn kannalta tärkeisiin käsitteisiin. Työn keskivaihe koostuu sovelluskehityksestä sekä erilaisista teknologisista ratkaisuista. Lopussa käydään läpi työn julkaisu sekä lopputulos.

Opinnäytetyön toimeksiantaja on Lahdessa sijaitseva Kauppakeskus Karisma. Kauppakeskus Karismalla aiemmin työskennelleenä tiesin alustavasta tarpeesta perehdytysprosessin kehittämiseen ja tästä syystä esitin ideaa uudesta perehdytysjärjestelmästä kauppakeskuksen johdolle, johon siellä tartuttiin innolla. Toimeksiantajan asettamana tavoitteena opinnäytetyölleni on saada käyttöön henkilökunnan perehdytysportaali, joka on käytettävissä verkossa ajasta ja paikasta riippumatta. Lisäksi kauppakeskusjohdon tulee pystyä päivittämään perehdytykseen liittyviä sisältöjä itsenäisesti. Portaali tulee korvaamaan nykyisen paikkasidonnaisen ja osittain vanhentuneen perehdytysmateriaalin.

Toimeksiantaja antoi vapaat kädet teknologioiden ja toteutustavan valinnassa. Tästä syystä työssä päädyttiin hyödyntämään Next.js-kehystä, josta oli jo aiempaa kokemusta. Next.js:n rinnalle valittiin headless CMS:nä Sanity, joka osoittautui projektin tarpeisiin sopivimmaksi palveluntarjoajaksi.

2 KEHITTÄMISTYÖN ALOITUS

2.1 Next.js & Vercel

Next.js on Vercelin kehittämä ilmainen ja avoimen lähdekoodin React-sovel-
luskehys (framework), joka soveltuu modernien verkkosovellusten kehittämi-
seen. React on käyttöliittymien luomiseen kehitetty JavaScript-kirjasto. Se ei
kuitenkaan yksinään taivu kaikkien, etenkin monimutkaisempien tai suurta
skaalautuvuutta vaativien verkkosovellusten toteuttamiseen. Tämä on toki
mahdollista kolmannen osapuolen työkalujen avulla, mutta laajemmista pro-
jekteista voi tällöin tulla varsin työläitä sekä monimutkaisia kehittää ja ylläpi-
tää. (Nextjs.org s.a.)

Next.js tuo tähän haasteeseen ratkaisun tarjoamalla edistyneitä toimintoja
Reactin perusvalikoiman täydennykseksi. Se tarjoaa Reactin rinnalle ratkai-
suja, jotka mahdollistavat palvelin pohjaisen renderöinnin (Server-Side-Rende-
ring, SSR), staattisen sivugeneroinnin (Static Site Generation, SSG) sekä au-
tomaattisen koodin pilkkomisen (Automatic Code Splitting). Lisäksi Next.js tu-
kee API-reittejä mahdollistaen backend-logiikan käyttämisen suoraan sa-
massa sovelluksessa. (Next.js documentation s.a.) Näiden ominaisuuksien
ansiosta Next.js on erittäin joustava ratkaisu erilaisiin sovelluskehitystarpeisiin,
aina yksinkertaisista staattisista sivustoista dynaamisiin, palvelin pohjaisiin
verkkosovelluksiin.

Next.js:n lisäksi Vercel on kehittänyt sovellusten julkaisuun ja kehitykseen
oman alustansa. Alusta on suunniteltu erityisesti staattisten ja Jamstack-so-
vellusten julkaisemiseen ja sen avulla kehittäjät voivat ottaa sovelluksensa
tuotantoon vain muutamalla klikkauksella. (Vercel 2023b.) Vercelin virallisen
dokumentaation mukaan alusta tukee Next.js:n lisäksi myös useita muita
staattisia sivugeneraattoreita (SSG) ja sovelluskehyskehyksiä (framework) kuten
esimerkiksi Angular, Vue.js, Remix, Gatsby.js ja Jekyll (Vercel 2023a).

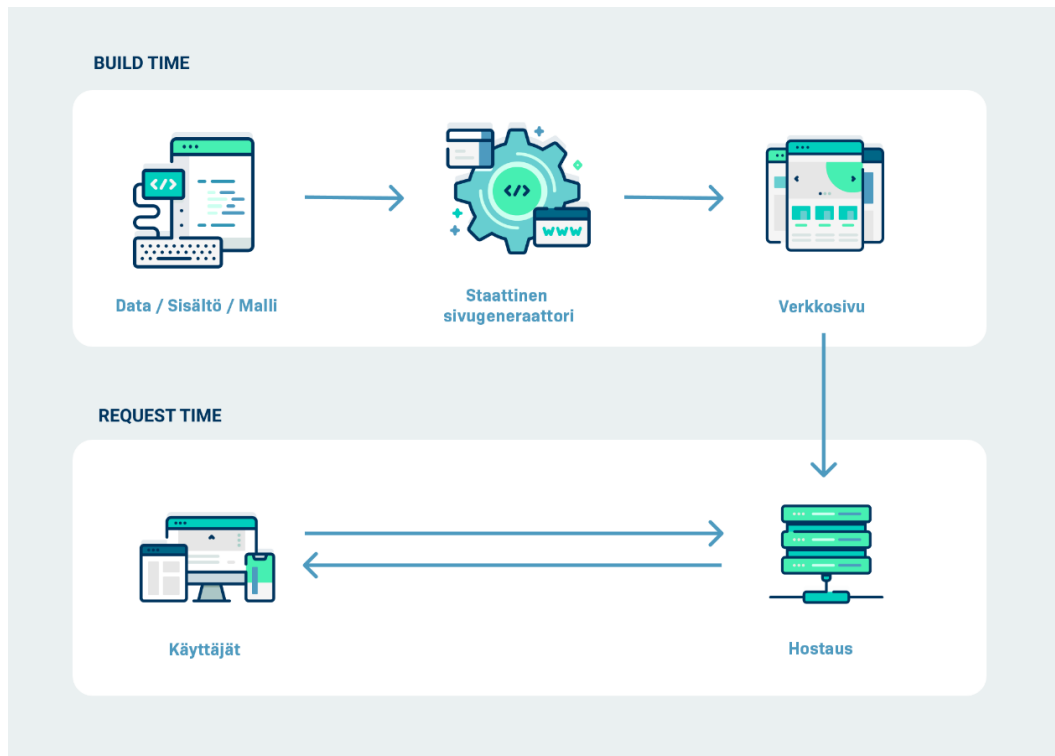
Tässä opinnäytetyön projektissa yhtenä Vercelin tärkeimmistä ominaisuuksista
voidaan pitää "Deploy Hooks" -toimintoa. Yksinkertaistettuna tämä tar-
koittaa, että staattinen sivusto rakennetaan (deploy/build) uudelleen ulkoisesta
pyynnöstä, joka voi tulla esimerkiksi GitHubista tai ulkoisesta sisällönhallinta-

järjestelmästä. Projektissa toimintoa hyödynnetään headless CMS-sisällönhallintajärjestelmän kanssa, jolloin muutokset CMS:n sisältöön laukaisevat automaattisesti sivuston uudelleenrakennuksen. Tämän ansiosta uusi päivitetty sisältö on aina näkyvissä sivustolla vieraileville.

2.2 Staattinen sivugenerointi (SSG)

Staattinen sivu viittaa HTML-muodossa olevaan verkkosivuun, joka näyttää käyttäjilleen saman sisällön riippumatta siitä, kuka sivua katsoo. Internetin alkuaikojen verkkosivut olivat kaikki tämänkaltaisia yksinkertaisia sivustoja. Sivustot eivät olleet tuolloin riippuvaisia tietokannoista tai palvelinpuolen (backend) skripteistä, mutta nykyään monet sivustot ovat dynaamisia ja kytköksissä erilaisiin tietokantoihin ja palveluihin. Tämä voi aiheuttaa sivustojen hidastumista ja altistumista tietoturvauhille. Nämä ovatkin suurimpia syitä staattisten sivujen paluuseen ja suosion kasvuun. Staattiset sivustot soveltuvat erinomaisesti esimerkiksi blogeille, dokumentaatiolle ja erilaisille infosivustoille, joissa sisältöä ei päivitetä jatkuvasti, eikä sen tarvitse mukautua vierailijan mukaan. (Camden & Rinaldi 2017, luku 1: Why Static Sites?) Tässä opinäytetyön projektissa perehdytysportaalin aineistoa päivitetään todennäköisimmin vain muutamia kertoja vuodessa, jolloin staattinen sivusto onärkevin vaihtoehto.

Miten staattinen sivu sitten luodaan? Tähän tarkoitukseen on olemassa monenlaisia eri työkaluja, esimerkiksi niin sanottuja staattisia sivugeneraattoreita (static site generator, SSG) kuten Next.js, Hugo, Gatsby ja Jekyll. Prosessia voidaan kuvata seuraavasti. Ensimmäisenä osana on itse sisältö, joka sijaitsee CMS-palveluntarjoajan palvelimilla. Toisena osana on esimerkiksi Next.js:llä luotu käyttöliittymän malli. Kolmas osa prosessia on julkaisualusta kuten esimerkiksi Vercel. Kun CMS-sisältöä muokataan, rakentaa Vercel sivuston aina uudelleen. Tässä rakennusvaiheessa noudetaan API-kutsulla uusi data CMS-palveluntarjoajalta ja yhdistetään se luotuun käyttöliittymään. Rakentamisvaiheen tuloksena syntyy staattinen verkkosivu, jossa sisältö ja malli on sidottu yhteen. Lopuksi kuten kuvassa 1 havainnollistetaan, siirretään tämä staattinen verkkosivu palvelimelle, josta se tarjoillaan valmiina kokonaisuutena sivuston vierailijoille.



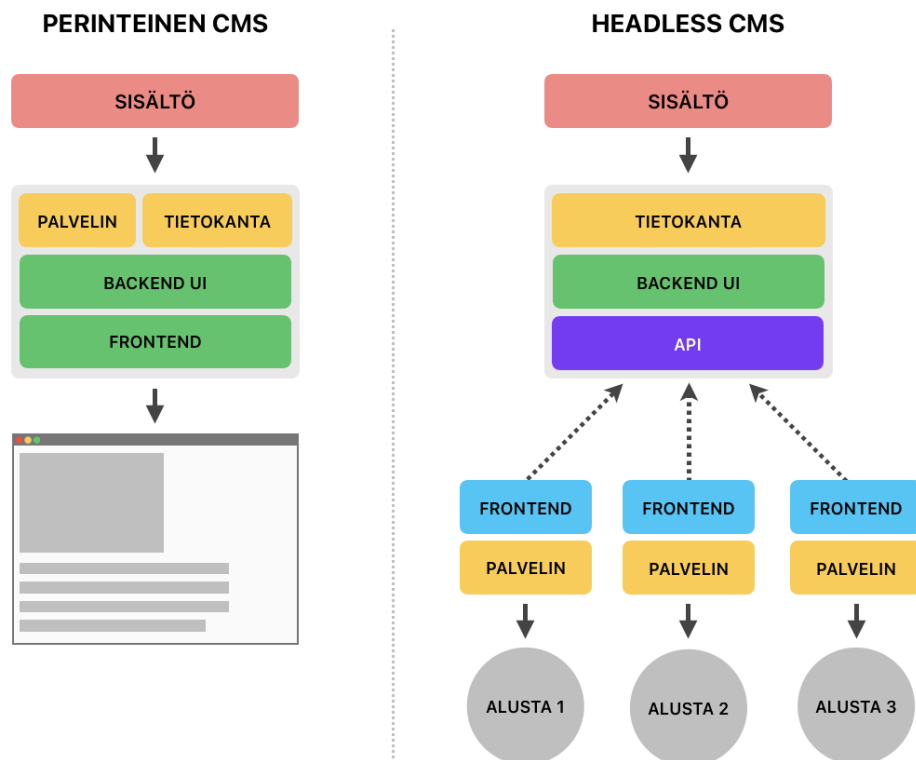
Kuva 1. Miten staattinen sivugenerointi toimii (mukaillen Hawksworth 2020)

Staattinen sivugenerointi ja headless CMS-integraatio ei ainoastaan yksinkertaista verkkosivustojen rakennusprosessia, vaan tuo myös lisäarvoa suorituskyvyn, turvallisuuden ja sisällönhallinnan näkökulmasta. Käyttämällä staattista sivugenerointia kehittäjät voivat hyödyntää sen nopeutta ja turvallisuutta, kun taas headless CMS-integraatio mahdollistaa rikkaan ja joustavasti päivittyvän sisällön esittämisen loppukäyttäjille. Tämä lähestymistapa antaa mahdollisuuden kehittää verkkosivustoja, joissa tehokkuus ja käyttäjäkokemus kulkevat käsi kädessä.

2.3 Sisällönhallintajärjestelmä (CMS)

Sisällönhallintajärjestelmä eli Content Management System (CMS) on olennainen digitaalisen sisällön luomisen, hallinnan, päivittämisen sekä julkaisun ja varastoinnin mahdollistava työkalu. Graafisen käyttöliittymän kautta se tarjoaa käyttäjäystävällisen rajapinnan, jonka kautta voidaan hallita erilaisia sisältötyyppejä - kuten tekstiä, kuvia ja videoita - ilman, että käyttäjän tarvitsee ymmärtää tai kirjoittaa itse koodia. (Amsler & Churchville 2021.) CMS:n keskeinen tehtävä onkin yksinkertaistaa monimutkaisia teknisiä prosesseja sekä tehdä verkkosivustojen ylläpidosta sujuvampaa.

Perinteisissä sisällönhallintajärjestelmissä sisältö pyritään sitomaan tiettyyn ulkoasuun tai esitystapaan (presentation layer), ja tällöin mukaan sisällytetään usein jonkinlaisia tyylimäärittelyjä, joka tekee sisällöstä sopivan vain tiettyihin käyttöympäristöihin. Headless CMS sen sijaan pitää huolen vain sisällöstä ja sen jakamisesta. Tämä mahdollistaa saman sisällön käyttämisen useissa eri kanavissa ja ulkoasun kustomoinnin tapauskohtaisesti kehittäjän toimesta. Headless CMS toimii tyypillisesti API:n (Application Programming Interface) kautta. Tämä tarkoittaa, että sisältö on saatavilla standardimuotoisena datana, joka voidaan noutaa ja esittää millä tahansa alustalla, joka pystyy käsittelemään kyseistä dataa. (Darji 2023.) Eräs merkittävä hyöty onkin headless CMS-palveluiden sitoutumattomuus mihinkään tiettyyn teknologiaan, mikä mahdollistaa kehittäjien itse valita heille mieluisimmat kehitysympäristöt (Holmes 2022). Seuraavassa kuvassa 2 on pyritty havainnollistamaan perinteisen ja headless-sisällönhallintajärjestelmän rakenteellisia eroavaisuuksia.



Kuva 2. Perinteisen ja headless-sisällönhallintajärjestelmän rakenteellisia eroja

Sekä perinteiset niin kutsutut monoliittiset sekä headless-sisällönhallintajärjestelmät voidaan lisäksi jakaa kahteen tyyppiin niiden ylläpitomuodon mukaan: joko itse ylläpidettäviin tai SaaS-palveluina (Software as a Service) tarjottaviin

(Wiśniewski 2021). Tässä opinnäytetyössä keskitytään juuri SaaS-palveluna toteutetun headless CMS:n integrointiin osaksi Next.js-projektia. Markkinoilta löytyykin nykypäivänä runsas määrä eri palveluntarjoajia, joita erottavat erilaiset ominaisuudet sekä palveluiden hinnoittelu. Monet palvelut ovat kuitenkin tarjolla myös ilmaiseksi pienemmän mittakaavan projekteihin.

2.4 Headless CMS palveluntarjoajan valinta

Valittaessa headless CMS-palveluntarjoajaa tähän projektiin painotettiin kolmea keskeistä kriteeriä. Ensinnäkin palvelun tuli olla maksuton, mutta silti tarjota riittävä määrä toiminnallisuuksia sen ilmaisversiossa. Toiseksi käyttäjäkokemuksen tuli olla intuitiivinen ja yksinkertainen loppukäyttäjiä ajatellen. Kolmanneksi palvelun tuli olla yhteensopiva Next.js:n kanssa ja sen tuli tarjota kattava dokumentaatio tukemaan integraatiota.

Valinta osoittautui haastavaksi, sillä markkinoilla on lukuisia vaihtoehtoja. Esimerkiksi Jamstack.org listaa yli sata erilaista sisällönhallintajärjestelmää (Jamstack.org s.a.). Tämän vuoksi oli epärealistista vertailla kaikkia saatavilla olevia vaihtoehtoja yksityiskohtaisesti. Ratkaisuna oli kohdentaa tarkastelu muutamaan alan suurempaan toimijaan ja valita näistä se, joka parhaiten vastasi projektiin asetettuja kriteereitä.

Alkuvaiheessa keskityttiin analysoimaan kunkin CMS:n ominaisuuksia suhteessa projektille asetettuihin tarpeisiin. Käyttäjäärvosteluiden, teknisen vertailun ja omien kokeilujen perusteella voitiin arvioida kunkin järjestelmän käyttönoton helppoutta, käyttäjäkokemusta sekä yhteisön ja tuen laatua. Erityisesti kehittäjäyhteisön aktiivisuus ja valmiit integraatiot Next.js:n kanssa olivat ratkaisevassa asemassa.

Loppuvalintaa tehtäessä kaksi vahvaa vaihtoehtoa nousivat esiin: Contentful ja Sanity. Molemmat tarjoavat monipuolisen ilmaisversion ja käyttäjäystävälliset käyttöliittymät. Itse pidin kuitenkin Sanityn dokumentaatiota selkeämpänä ja kattavampana Contentfulin vastaavaan verrattuna. Myös Sanityn aktiivinen kehittäjäyhteisö teki siitä kehittäjän näkökulmassa houkuttelevamman vaihtoehdon. Henkilökohtaisesti myös koin Sanityn Studion käyttöliittymän miellyttävämmäksi ja uskon sen palvelevan myös loppukäyttäjää paremmin erityisesti

sen selkeyden myötä. Näistä syistä Sanity valikoitui tässä projektissa sisällönhallintajärjestelmäksi.

3 SOVELLUSKEHITYSPROSESSI JA TEKNOLOGISET RATKAISUT

3.1 Kehitysympäristö ja Sanityn asentaminen

Projektin kehitysympäristönä käytetään Microsoftin Visual Studio Codea, jossa version hallinta on toteutettu Git-integraation kautta. Ohjelmointikielenä on React-pohjainen Next.js ja sovelluksen julkaisualustana toimii Vercel. Sisällön tietokanta ja sen hallinta hoidetaan Sanityn sisällönhallintajärjestelmän kautta.

Sanityn asentaminen on sujuvaa. Se voidaan asentaa joko itsenäisenä sovelluksena tai integroida osaksi Next.js-projektia. Tässä projektissa valittiin jälkimmäinen vaihtoehto, mikä tarkoittaa, että sekä käyttöliittymä sekä sisällönhallintajärjestelmä voidaan hallita ja mukauttaa samassa kehitysympäristössä. Kuvassa 3 esitetään asennukseen tarvittavat komennot Next.js:n ja Sanityn osalta.

```
npx create-next-app@latest perehdytysportaali --typescript --tailwind --eslint --app --no-src-dir --import-alias="@/*  
---  
npx sanity@latest init --env --create-project "Perehdytysportaali" --dataset production
```

Kuva 3. Next.js (yllä) ja Sanity (alla) asennuskomennot

Sanityn asennuksen yhteydessä kirjautumisen jälkeen kysyttiin Sanityn määrittämiseen seuraavassa kuvassa 4 nähtäviä lisätietoja. Sanity tarjoaa asennuksen yhteydessä mahdollisuutta käyttää valmista blogityyppistä mallia (template), mutta tämän projektin osalta on kuitenkin luontevaa aloittaa tyhjältä pöydältä.

```
? Would you like to add configuration files for a Sanity project in this Next.js folder? Yes  
? Do you want to use TypeScript? Yes  
? Would you like an embedded Sanity Studio? Yes  
? Would you like to use the Next.js app directory for routes? Yes  
? What route do you want to use for the Studio? /studio  
? Select project template to use Clean project with no predefined schemas
```

Kuva 4. Sanityn asennuksen määrittäykset

Jotta uusi Sanity-projekti voitaisiin luoda, tarvitaan Sanity-käyttäjätili. Sanityn asennuksen yhteydessä voit joko kirjautua sisään olemassa olevalle käyttäjättilille tai luoda uuden käyttäjättilin jolle projekti alustetaan. Sanity luo asennuksen yhteydessä automaattisesti tarvittavat ympäristömuuttujat omaan tiedostoonsa, joten kehittäjän ei tarvitse niistä erikseen huolehtia. Asennusvaiheessa automaattisesti luodut ympäristömuuttujat ovat `project_id` ja `dataset`. `Project_id` määrittää, mikä Sanity-projekti on käytössä, ja `dataset` vuorostaan kertoo, mikä tietojoukko on aktiivisena.

3.2 Sanity Studio ja skeemat

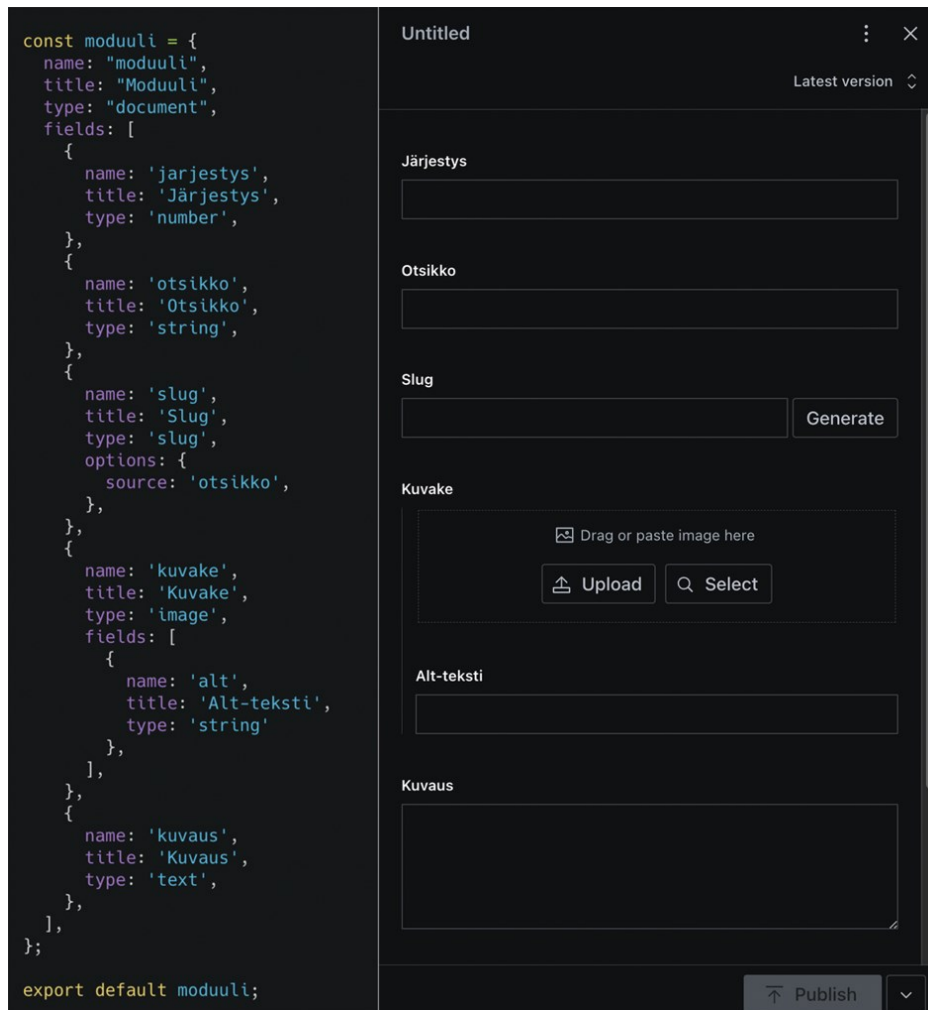
Sanitylla on oma graafinen työkalu sisällön hallintaan, joka on nimeltään Sanity Studio. Kyseessä on niin sanottu SPA-sovellus (single-page application), jota käytetään verkkoselaimella. Studio on yhteydessä API:n kautta Content Lake nimeä kantavaan Sanityn ylläpitämään pilvipohjaiseen tietovarastoon, mistä se noutaa ja mihin se tallentaa sisältöä. (Sanity Documentation 2023b.) Koska Sanity asennettiin osaksi Next.js-projektia ja määritettiin hyödyntämään Next.js:n reititystä, päästään kehitysvaiheessa Sanity Studion hallintapaneeliin yksinkertaisesti käyttämällä polkua <https://localhost:3000/studio>. Alkuun Sanity Studiossa ei voi vielä tehdä juuri mitään, sillä se vaatii vähintään yhden tietomallin toimiakseen.

Sanity kutsuu omaa tietomallejaan skeemoiksi (schema). Skeema kuvaa, millaisia dokumentteja Sanity Studiolla voidaan luoda sekä minkä tyyppisiä kenttiä dokumenteissa on. Skeemat määritellään koodieditorilla, ja ne määritellään käyttäen JavaScript- tai TypeScript-objekteja, jotka kuvaavat erilaisia sisältötyyppejä ja niiden kenttiä. Jokainen skeema voi sisältää useita kenttiä, kuten tekstiä, numeroita, kuvia, tiedon validointia sekä myös viitteitä muihin skeemoihin. Tämä mahdollistaa monipuolisten ja rikkaiden sisältörakenteiden luomisen, jotka voidaan sitten esittää verkkosivuilla tai sovelluksissa. Tässä projektissa käytetään neljää erilaista skeemaa, jotka ovat ohjelma, moduuli, artikkeli, kysymys. Seuraavassa kuvassa 5 voidaan nähdä esimerkki yksinkertistetun artikkeli-nimisen skeeman määrittelystä.

```
const artikkeli = {
  name: 'artikkeli',
  title: 'Artikkeli',
  type: 'document',
  fields: [
    {
      name: 'otsikko',
      title: 'Otsikko',
      type: 'string',
    },
    {
      name: 'sisalto',
      title: 'Sisältö',
      type: 'text',
    },
    {
      name: 'kuva',
      title: 'Kuva',
      type: 'image',
    },
  ],
};
```

Kuva 5. Esimerkki yksinkertaisen skeeman määrittelystä

Edellä olevan kuvan skeeman mukaan "artikkeli" on tyypiltään dokumentti ja se sisältää kolme erilaista sisältötyyppiä, jotka ovat otsikko, sisältö ja kuva. Name-parametri on se, jolla viitataan kyseiseen sisältötyyppiin sekä koodissa että API-kutsuissa. Title-parametri puolestaan on kyseisen tyypin nimi, joka näkyy käyttöliittymässä. Type-parametri määrittää kyseisen sisältötyypin datatyyppin, eli sen, millaista dataa kyseinen kenttä voi sisältää. Sanity tukee myös useita muita erilaisia sisältötyyppejä kuten: array, block, object, slug, URL, number, geopoint, date, reference jne. Sanityn sisältötyypeistä ja niiden käytöstä löytyy kattavasti lisätietoa Sanityn omassa dokumentaatiossa. Skeemoja voidaan hyödyntää myös niin sanottujen alkuarvomallien (initial value template) määrittämiseen. Tällä mallilla (template) voidaan määrittää tiettyjen sisältötyyppien alkuarvot, joka voi olla hyödyllistä esimerkiksi silloin, jos eri dokumentit koostuvat usein samoista arvoista tietyissä kentissä. Näin saadaan nopeutettua ja yksinkertaistettua sisällöntuotantoprosessia. Tämän projektin osalta näitä malleja ei kuitenkaan hyödynnetä, sillä sisältökenttien tiedot vaihtelevat usein. Kun tietomalli eli skeema on määritetty, voidaan sitä tarkastella Sanity Studioon kautta. Kuvassa 6 nähdään tässä projektissa käytetyn moduulityypin skeema yksinkertaistettuna sekä miten se näkyy käyttäjälle Sanity Studioon.



Kuva 6. Moduuli-skeema koodina (vasemmalla) ja Sanity Studiassa (oikealla)

Kuvan 6 skeemasta voidaan lisäksi mainita, mihin tietoja "järjestys" ja "slug" käytetään tässä projektissa. Järjestysnumeroa käytetään moduulien järjestyksen määrittämiseen itse moduulinäkymässä. "Slug" taas generoidaan moduulin otsikosta ja sitä käytetään URL-polkuna, jolla viitataan koodissa tähän tiettyyn moduuliin. Kun halutut tiedot on täydennetty, voidaan kyseinen moduuli julkaista Sanity Studio Publish-painikkeella, jolloin moduuli tietoineen tallentuu Sanity Content Lake -tietovarastoon.

3.3 API yhteys ja GROQ

Sanityn Content Lake on pilvipohjainen tietovarasto, johon sisältöä tallennetaan ja josta sitä noudetaan Sanityn API-rajapintaa käyttäen. Tietovarastossa sijaitseva data eli sisältö säilötään datajoukoissa (dataset) joita voidaan kuvitella eräänlaisina tietokantoina. Sanityn tapauksessa datajoukko koostuu JSON-muotoisista (JavaScript Object Notation) dokumenteista. Nämä doku-

mentit voivat olla erityyppisiä ja sisältää viittauksia toisiinsa. (Sanity Documentation 2023a.) API-rajapintaa voidaan käyttää joko suoraan HTTP:n välityksellä, tai niin sanotulla client-kirjastoilla. Koska tässä projektissa käytetään Next.js-kehystä (framework), joka pohjautuu JavaScriptiin, on myös järkevää yhteensopivuuden ja kehityksen ketteryuden kannalta hyödyntää Sanityn tarjoamaa JavaScript client-kirjastoa sisällön hakemiseen tietovarastosta.

Datajoukot voivat olla joko julkisia tai yksityisiä. Tässä projektissa datajoukko on julkinen, sillä mitään sen sisältöä ei ole tarve salata. Julkiseen datajoukkoon yhdistämiseen tarvitaan vain projektin ID:n sekä datajoukon nimen. Jos datajoukko haluttaisiin pitää yksityisenä, vaadittaisiin yhdistämiseen myös API-avain. Edellä mainittujen tietojen lisäksi voidaan myös määritellä käytettävä API-versio ja halutaanko hyödyntää CDN:ää (Content Delivery Network) sisällön jakeluun. API-version avulla voidaan varmistaa, että sovelluksen ja Sanityn välinen kommunikaatio toimii odotetulla tavalla, kun käytössä on aina sama versio. Tämä lisää rajapinnan käytön yhteensopivuutta ja vakautta. CDN:n käyttö puolestaan mahdollistaa sisällön nopeamman ja tehokkaamman toimituksen loppukäyttäjille. Kuvassa 7 nähdään, miten yhteys muodostetaan Sanityn JavaScript client-kirjaston avulla. Kuvassa näkyvät "createClient" funktion argumentit apiVersion, dataset, projectId ja useCdn määritetään erillisessä tiedostossa, mutta ne voitaisiin toki määrittää suoraan itse funktiossakin.

```
import { apiVersion, dataset, projectId, useCdn } from "./env";

const client = createClient({
  apiVersion,
  dataset,
  projectId,
  useCdn,
})
```

Kuva 7. Sanity clientin määrittäminen

Yhteyden muodostamisen jälkeen voidaan sisältöä hakea Content Lakesta Sanityn omalla kyselykielellä nimeltä GROQ (Graph-Relational Object Queries). GROQ on suunniteltu erityisesti räätälöitävyys ja joustavuus huomioiden, mahdollistaen monimutkaisten kyselyjen tekemisen ilman monimutkaista logiikkaa. Se eroaa perinteisistä SQL-kyselyistä ja muista JSON-pohjaisista

kyselykielistä siinä, että se on erityisen vahva relaationaalisissa ja hierarkkississa tiedonhauissa. Toisin sanoen GROQ on suunniteltu erityisesti sellaisten tietorakenteiden käsittelyyn, joissa data on linkitetty toisiinsa monimutkaisilla tavoilla tai järjestetty useissa alatasoissa. Tämä tekee siitä tehokkaan työkalun monimutkaisten datarakenneongelmien ratkaisemiseen, joita voi esiintyä esimerkiksi sisällönhallintajärjestelmissä.

GROQ:ssa kyselyitä tehdään käyttämällä objekteja, listoja ja erilaisia toimintoja. Kuvassa 8 näytetään `getArtikkelit`-funktio, jolla haetaan kaikki artikkelityyppiset dokumentit GROQ-kyselyn avulla.

```
export async function getArtikkelit(): Promise<Artikkeli[]> {
  return client.fetch(
    groq`*[_type == "artikkeli"]{
      _id,
      "moduuli": moduuli->{
        otsikko,
        "slug": slug.current
      },
      otsikko,
      jarjestys,
      sisalto,
      "kuvat": kuvat[]{
        jarjestys,
        "url": asset->url,
        "alt": kuva.alt
      }
    }`
  );
}
```

Kuva 8. Kysely GROQ-kyselykielellä

Kuvassa 8 esitetty kysely voidaan pilkkoa osiin sen sisällön selventämiseksi. Ensimmäisessä osassa kyselystä, `*[_type == "artikkeli"]`, kerrotaan, että kaikki dokumentit, joilla on tyyppinä "artikkeli", halutaan hakea. Tämä on yksinkertainen kysely, joka käyttää objektioperaattoria `*` hakeakseen kaikki dokumentit ja suodatinta hakasulkeiden sisällä [...] rajatakseen hakutuloksia tietyillä kriteereillä, tässä tapauksessa `_type`-kentällä ja sen arvolla "artikkeli".

Seuraavaksi aaltosulkeiden sisällä {...}, määritellään, mitä tietoja näistä artikkeleista halutaan palauttaa. Kyselyssä pyydetään palauttamaan artikkelien tunniste (`_id`), otsikko, järjestys ja sisältö. Esimerkkinä kyselyn syvemmistä ta-

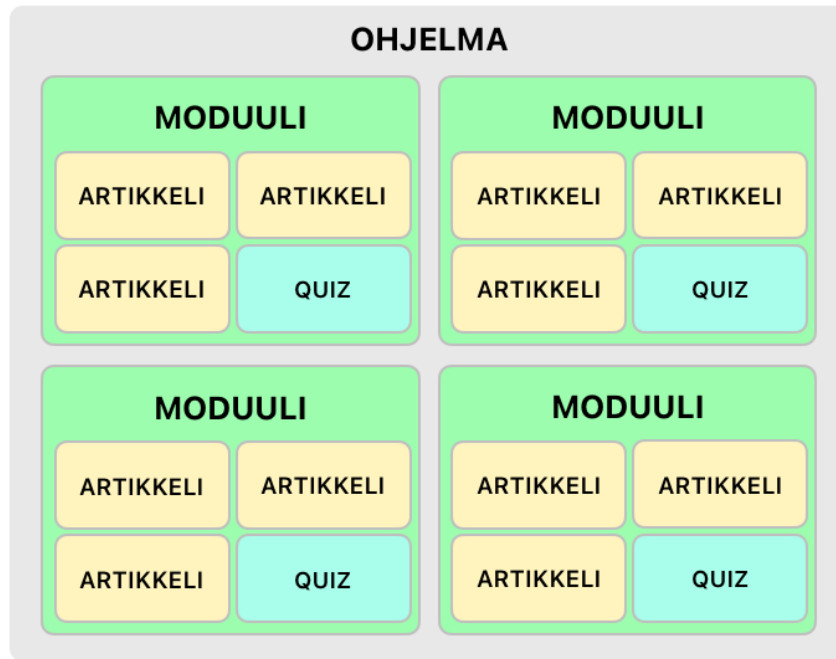
soista ja viittausten seuraamisesta pyydetään palautuksessa moduulin ja kuvan tiedot. Moduuli-kentässä käytetään nuolioperaattoria (->) seuraamaan viittauksen kohteena olevaa dokumenttia ja hakemaan sieltä tietyt kentät: otsikko ja slug. Vastaavasti "kuvat"-kentässä taulukko-operaattoria [{...}] käytetään kaikkien kuvien ja niiden tiettyjen kenttien hakemiseen sekä seurataan (->) jälleen viittausta asetiin kuvan URL:n ("url": asset->url) hakemiseksi.

Tämä kysely on hyvä esimerkki siitä, kuinka GROQ:n avulla voidaan tehdä monimutkaisia kyselyjä yksinkertaisella ja selkeällä tavalla. Se näyttää, kuinka objekteja, taulukoita ja viittauksia voidaan yhdistellä, jotta saadaan juuri ne tiedot, joita tarvitaan, yhdellä kyselyllä. Se vähentää tarvetta useille erillisille kyselyille ja tekee tiedonhausta tehokkaampaa. Se on selkeä esimerkki siitä, miksi GROQ on suosittu valinta monimutkaisten tietomallien käsittelyssä.

GROQ:n lisäksi Sanity tukee myös GraphQL:ää (Graph Query Language), joka on erittäin suosittu avoimen lähdekoodin datankäsittelykieli. Vaikka GROQ on erityisesti suunniteltu toimimaan Sanityn kanssa, GraphQL tarjoaa laajemmat mahdollisuudet ja on tuttu monille kehittäjille. Toisin kuin GROQ, joka keskittyy vain tiedon hakuun, GraphQL:lla voi myös lisätä, muokata ja poistaa tietoja. Tässä projektissa käyttöön valittiin kuitenkin GROQ, koska se sopii yksinkertaisempiin tarpeisiin ja on tiiviimmin integroitu Sanityyn.

3.4 Tiedon esittäminen käyttäen Next.js dynaamista reititystä

Perehdytysportaali koostuu eri kohderyhmille suunnatuista perehdytysohjelmista. Näiden perehdytysohjelmien alle on koottu erilaisiin aihealueisiin liittyviä moduuleita. Moduulit sisältävät aihealueen erilaisia artikkeleita, sekä kysymyksiä portaalin quiz-osiota varten. Kuvassa 9 on pyritty hahmottamaan sitä, millainen rakenne perehdytysportaalin sisällöllä on.



Kuva 9. Perehdytysportaalin sisällön eri tyypit

Sovelluksessa ohjelmat, moduulit, artikkelit sekä kysymykset muodostavat jokainen oman reitin seuraavasti `"https://sovellus.com/ohjelma/moduuli/artikkeli"`. Esimerkin reitissä `"ohjelma"`, `"moduuli"` ja `"artikkeli"` korvataan ko. sisällölle määritetyillä slugeilla eli URL-ystävällisillä otsikoilla. Kyseessä on tällöin siis niin sanottu dynaaminen reititys, joka riippuu siitä, mitä tiettyä sisältöä ollaan reitittämässä. Dynaamisen reitityksen toteuttaminen onnistuu Next.js:llä kuitenkin helposti.

Perehdytysportaali hyödyntää niin kutsuttua täysin dynaamista reititystä (Fully Dynamic Routing), jossa kaikki reitin osat kuten ohjelma, moduuli ja artikkeli sekä niiden väliset linkit ja sisällöt ovat dynaamisia. Reitien osat määritellään Next.js:ssa dynaamisiksi ympäröimällä reitin nimi kansiorakenteessa hakasulkeilla, esimerkiksi `[ohjelma]`. Tähän hakasulkeilla merkittyyn reittiin voidaan nyt asettaa tietoa dynaamisesti. Reittien ja linkkien dynaamisuus muodostuu erilaisten tietokantahakujen ja niiden manipuloinnin tuloksena. Kuvassa 10 nähdään yksinkertaistettu esimerkki siitä, miten esimerkiksi ohjelma reitistä siirrytään linkin avulla järjestysnumerolla pienimmän moduulin sivulle.

```

export default async function OhjelmaPage({ params }: { params: { ohjelma: string } }) {
  // Tietokantahaut
  const moduulit = await getModuulit();
  const ohjelmat = await getOhjelmat();

  // Haetaan ohjelman otsikko
  const ohjelmanNimi = () => {
    const ohjelma = ohjelmat.filter((ohjelma) => ohjelma.slug === params.ohjelma)
    return ohjelma[0].otsikko;
  };

  // Selvitetään pienimmän järjestysnumeron moduuli
  const smallestOrderModuuli = moduulit.sort((a, b) => a.jarjestys - b.jarjestys)[0].slug;

  return (
    <>
    <TopBar title={ohjelmanNimi()} />
    <div className="flex flex-col text-center rounded-md p-2">
      { /* Kyseisen ohjelman kuvaus */ }
      { ohjelmat.filter((ohjelma) => ohjelma.slug === params.ohjelma).map((ohjelma) => (
        <div key={ohjelma.slug}>
          <div>
            <PortableText
              value={ohjelma.kuvaus}
              components={RichTextComponents}
            />
          </div>
          { /* Linkki ohjelman ensimmäiseen moduuliin */ }
          <Link href={`/${params.ohjelma}/${smallestOrderModuuli}`}
            className="btn btn-secondary">
            Aloita
          </Link>
        </div>
      ))}
    </div>
    </>
  )
}

```

Kuva 10. Esimerkki dynaamisen reitityksen toteutuksesta *ohjelma* reitistä *moduuli* reittiin

Reittien lisäksi myös sisällön esitys tehdään dynaamisesti. Kun käyttäjä siirtyy esimerkiksi edeltävässä kuvassa 10 esitellylle ohjelmasivulle, haetaan kyseisen ohjelman kuvaustiedot reitin URL-parametrin perusteella. Tämä tarkoittaa, että kaikkien ohjelmien joukosta suodatetaan esiin se, jonka "slug" vastaa URL-parametrina saatuja tietoja ("params.ohjelma"), ja näin käyttäjälle voidaan esittää juuri kyseistä reittiä vastaavan ohjelman sisältö.

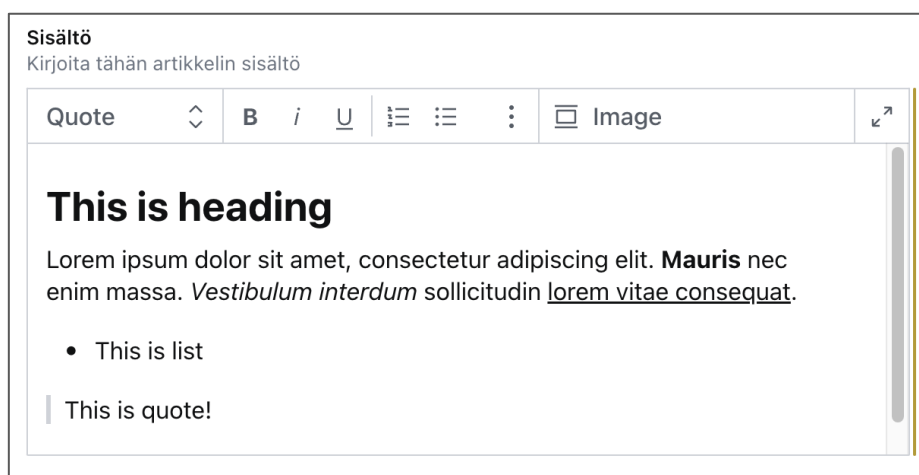
Hyvin rakennetun dynaamisen reitityksen ansiosta sovelluksen skaalautuvuus pysyy myös hyvin hallussa. Uusia ohjelmia, moduuleita sekä artikkeleita voidaan lisätä rajaton määrä ilman, että tarvitsee muuttaa olemassa olevaa koodirakennetta tai reitityslogiikkaa.

3.5 Portable text -muotoilu

Portable text on Sanityn kehittämä määrittely, joka tarjoaa kehittäjille joustavan tavan käsitellä rikasta tekstisisältöä. Sen perusidea on tallentaa teksti lohkoina, jotka voivat olla joko perinteisiä tekstikappaleita tai mukautettuja sisältötyyppejä. Kukin lohko Portable Text -järjestelmässä voi sisältää erilaisia elementtejä, kuten tekstin jaksoja, kuvia tai vaikkapa upotettuja videoita.

Erityisen vahva puoli Portable Textissä on sen kyky rikastaa tekstiä erilaisilla merkinnöillä. Tekstijaksoihin voidaan liittää merkintöjä, jotka voivat viitata esimerkiksi korostukseen, linkkeihin tai muihin määriteltyihin toimintoihin. Tämä tekee sisällöstä dynaamista ja helposti muunneltavaa, sekä mahdollistaa monipuolisen sisällön esittämisen ja käsittelyn.

Tässä opinnäytetyössä Portable Textiä hyödynnetään juuri portaalin sisällön muotoiluun, jolloin sivuston mallin (template) ei tarvitse ottaa kantaa siihen, miten jokin osa esimerkiksi tekstisisällöstä on muotoiltu tai missä kohtaa kuvia esitetään. Sisällöntuottaja voi määritellä kaiken tämän itse Sanity Studiossa. Kuvassa 11 nähdään seuraavaksi miltä Sanity Studion Portable Text -käyttöliittymä näyttää sisällöntuottajalle.



Kuva 11. Sanity Studion Portable Text -kenttä

Tekstiä voidaan siis muotoilla monilla eri tavoilla sekä lisätä eri tasoisia otsikoita tai esimerkiksi listoja ja lainauksia. Myös kuvat voidaan upottaa suoraan Portable Text -kenttään, ja ne esiintyvät tällöin tekstin seassa juuri siellä, missä sisällöntuottaja niiden haluaa näkyvän. Jotta Portable Text saadaan

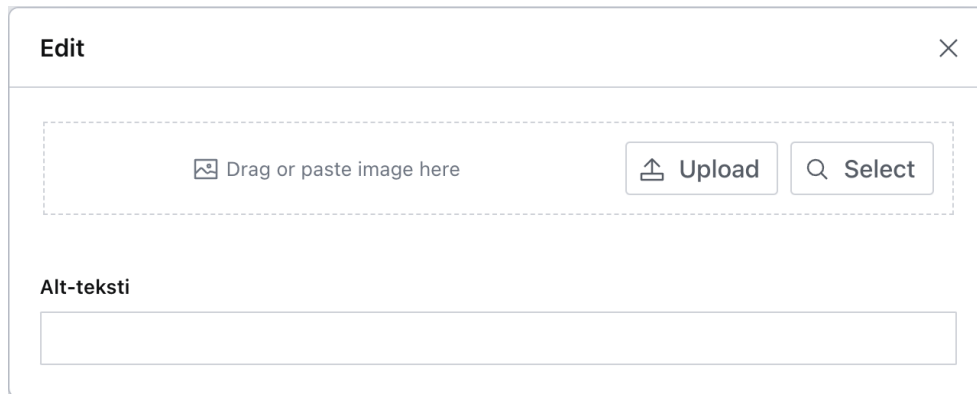
käyttöön, tarvitsee kehitysympäristöön asentaa kaksi lisäosaa. Ensimmäinen on "@portabletext/react", joka on suunniteltu renderöimään Portable Text -sisältöä React-komponentteina. Tämä kirjasto mahdollistaa myös mukautettujen renderöintisääntöjen määrittelyn erilaisille Portable Text -datatyypeille, kuten otsikoille, kappaleille, listoille ja kuvien upotuksille. Toinen lisäosa, "@sanity/image-url", hoitaa Sanity Studiosta palvelimelle lähetettyjen kuvien URL-osoitteiden muodostamisen. Tämän paketin avulla voidaan lisäksi määrittää kuvan URL-osoitteeseen liittyviä parametreja, kuten skaalausta, rajoituksia ja muita kuvankäsittelytoimintoja, mikä on erityisen hyödyllistä kuvien dynaamiseen käsittelyyn verkkosivustoilla.

Projektissa käytetään Tailwindin CSS -tyyliluokkia, joita voidaan kätevästi hyödyntää myös Portable Textin muotoilussa. Kuvassa 12 nähdään, kuinka erilaisille otsikkotyypeille on määritelty erityiset Tailwind CSS -tyylit. Esimerkiksi, kun käyttäjä valitsee Sanity Studiossa otsikkotasoksi Heading2/H2, määritellään tämän <h2>-elementin tyyliluokaksi tailwindin *text-2xl* ja *font-bold*. Nämä tailwind-määrytykset on tehty Next.js:ssä erillisessä RichTextComponents-komponentissa, jossa jokainen elementtityyppi saa omat tyylinsä.

```
block: {
  h1: ({ children }: any) => (
    <h1 className="text-3xl font-bold">{children}</h1>
  ),
  h2: ({ children }: any) => (
    <h2 className="text-2xl font-bold">{children}</h2>
  ),
  h3: ({ children }: any) => (
    <h3 className="text-xl font-bold">{children}</h3>
  ),
  h4: ({ children }: any) => (
    <h4 className="text-lg font-bold">{children}</h4>
  ),
  blockquote: ({ children }: any) => (
    <blockquote className="border-l-4 border-purple-500 italic my-4 pl-8">
      {children}
    </blockquote>
  ),
},
```

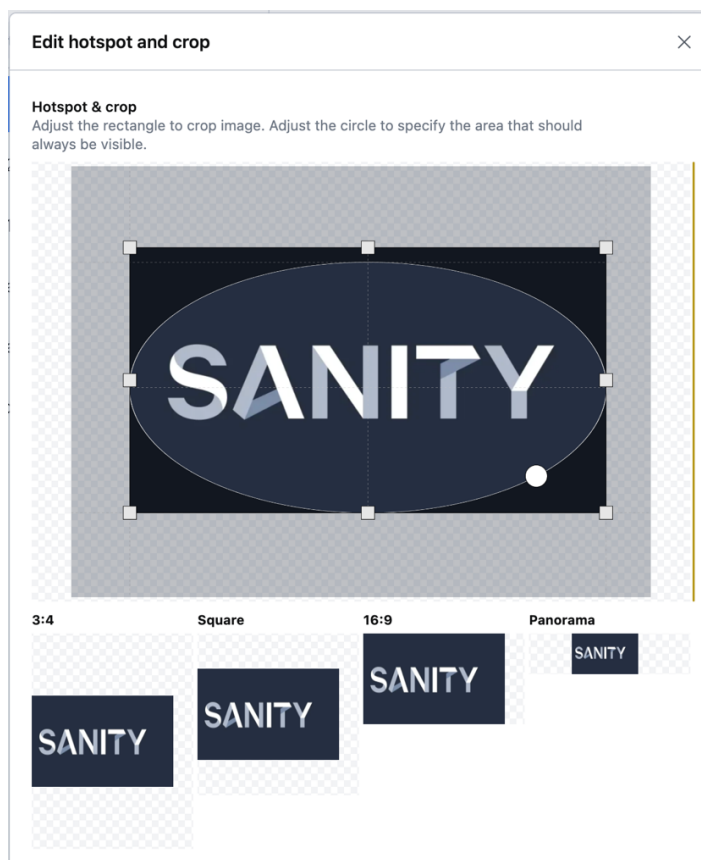
Kuva 12. Portable Textin tyylimäärittelyä

Kun sisällöntuottaja upottaa kuvan Portable Textin sekaan, lähetetään haluttu kuva Sanityn palvelimelle tai valitaan kuva jo palvelimella olevista kuvatiedostoista. Tämän lisäksi kuvalle määritellään alt-teksti saavutettavuuden parantamiseksi. Kuvassa 13 nähdään Sanity Studion kuvanlisäysnäköymä, jossa nämä määrytykset tehdään.



Kuva 13. Sanity Studio Portable Text ja kuvan lisäys

Kuvassa 14 nähdään, miten palvelimelle lisättyä kuvaa voidaan esimerkiksi rajata. Myös kuvan hotspot-kohdan määrittäminen on mahdollista, näin tiettyä kuvan kohtaa voidaan korostaa näkymään paremmin esimerkiksi pienemmissä thumbnail-tyyppisissä kuvissa.



Kuva 14. Sanity Studion kuvan rajausta ja hotspot määrittely

RichTextComponents-komponentissa kuvien käsittely aloitetaan tarkistamalla, että tarvittavat tiedot (value) ovat saatavilla. Jos tietoja ei ole, kuvaa ei rende-

röidä. Onnistuneen tarkastuksen jälkeen käytetään `urlForImage` funktiota luomaan URL-osoite, joka viittaa suoraan kuvan sijaintiin Sanityn palvelimella. Kuvassa 15 nähdään, miltä `RichTextComponents`in `image`-tyypin osio näyttää koodissa.

```
image: ({ value }: any) => {
  if (!value) {
    return null;
  }

  const imageUrl = urlForImage(value).url();
  if (!imageUrl) {
    return null;
  }

  return (
    <div className="relative w-full max-w-md mx-auto">
      <Image
        className="rounded-md"
        src={imageUrl}
        alt={value.alt || 'Kuva'}
        width={value.asset?.metadata?.dimensions?.width || 1000}
        height={value.asset?.metadata?.dimensions?.height || 1000}
      />
    </div>
  );
},
```

Kuva 15. Kuvan käsittely `RichTextComponents`-komponentissa

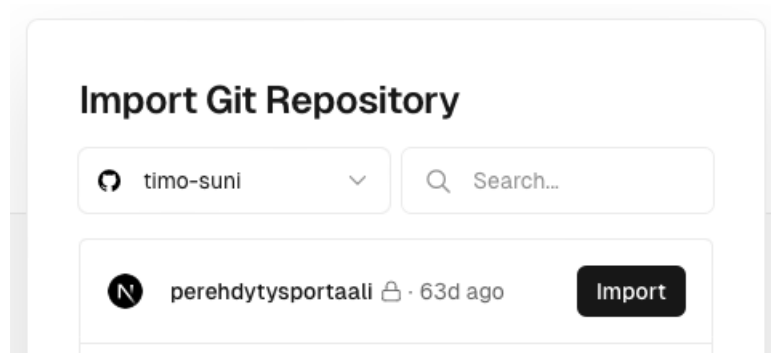
Kuvan renderöintiin käytetään Next.js:n `Image`-komponenttia. Komponentin `src`-attribuutti saa arvokseen aiemmin luodun `imageUrl`-osoitteen, ja `alt`-attribuutti asetetaan kuvan `alt`-tekstille tai käytetään oletusarvoa. Kuvan leveys ja korkeus määritetään sen alkuperäisten mittasuhteiden perusteella tai käytetään oletusarvoja, jos tietoja ei ole. Näin varmistetaan kuvan oikeanlainen ilmentyminen ja integroituminen tekstisisältöön.

4 JULKAISU JA LOPPUTULOS

4.1 Sovelluksen julkaisu ja päivitys-hookit

Sovelluksen julkaisuun käytetään Vercelin omaa julkaisualustaa. Koska sovelluksella on oma GitHub-repositorio, on helpointa tehdä julkaisu suoraan tästä repositoriosta. Tällöin Vercelin julkaisualusta rakentaa ja julkaisee automaattisesti sovelluksen uudelleen, kun repositoriota päivitetään. Näin sivuston pitäminen ajan tasalla on nopeaa ja vaivatonta.

Jotta Vercelillä olisi pääsy tämän sovelluksen repositorioon, on ensin linkitettävä käytössä oleva GitHub-tili Vercelin tilin kanssa. Tämän jälkeen on vielä lisättävä lupa Vercelille GitHubin integraatio asetuksissa, jotta Vercelille saadaan pääsy sovelluksen repositorioon. Kun nämä vaiheet on tehty, voidaan sovelluksen repositorio tuoda Vercelin julkaisualustalle kätevästi Import-painikkeella, kuten kuvassa 16.



Kuva 16. Git repositorion tuonti Vercel-julkaisualustalle

Repositorion tuonnin yhteydessä Vercel pyytää lisäämään ympäristömuuttujat manuaalisesti. Muuttujien lisäyksen jälkeen alkaa projektin rakennusvaihe eli buildaus. Onnistuneen rakennusvaiheen jälkeen projekti julkaistaan (deploy) automaattisesti Vercelin palvelimilla uniikkiin osoitteeseen kuten "esimerkki.vercel.app". Sovellus on tällöin käytettävissä kyseisessä osoitteessa. Mikäli sovellukselle haluttaisiin lisätä oma domain, kuten "esimerkki.fi", on se lisättävissä helposti projektikohtaisten domain asetusten kautta Vercelin julkaisualustalla.

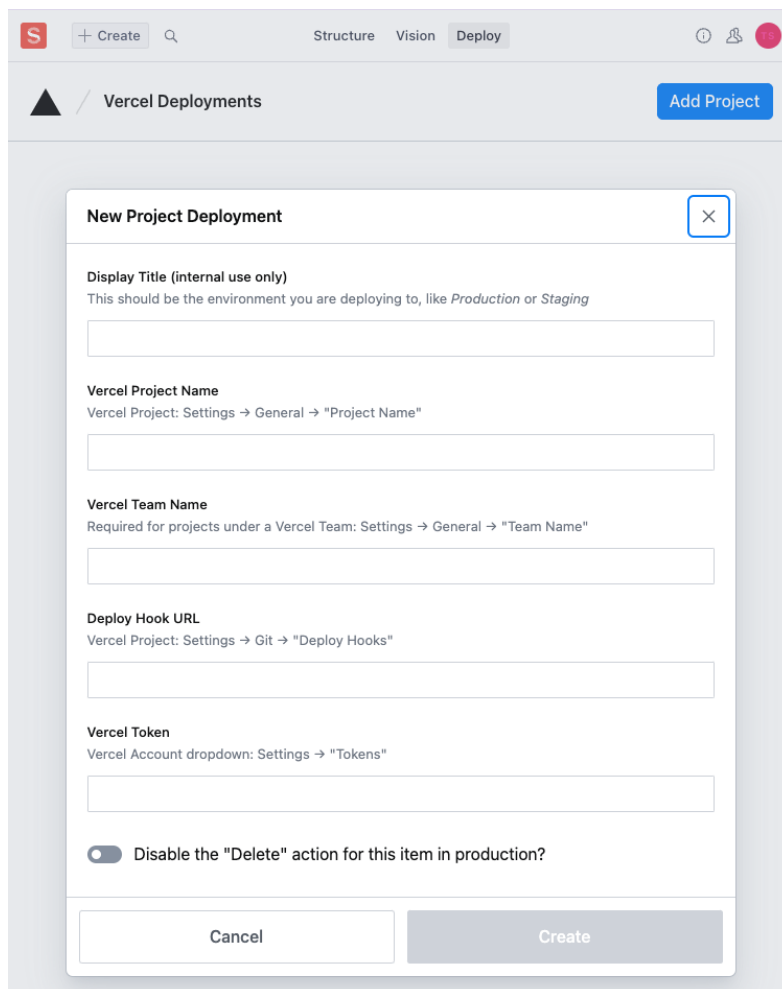
Sanityn kautta sovelluksen sisältöön tehtyjen muutosten päivitys tapahtuu Sanityn "vercel-deploy" pluginin avulla. Plugin asennetaan komennolla "npm i sanity-plugin-vercel-deploy". Plugin otetaan käyttöön Sanityn konfiguraatiodokumentossa "sanity.config.ts" lisäämällä vercelDeployTool plugineihin, kuten kuvassa 17.


```
import { visionTool } from '@sanity/vision'
import { defineConfig } from 'sanity'
import { structureTool } from 'sanity/structure'
import { vercelDeployTool } from 'sanity-plugin-vercel-deploy'
import { apiVersion, dataset, projectId } from './sanity/env'
import schemas from './sanity/schemas/index'

export default defineConfig({
  basePath: '/studio',
  projectId,
  dataset,
  schema: { types: schemas },
  plugins: [
    structureTool(),
    visionTool({defaultApiVersion: apiVersion}),
    vercelDeployTool(),
  ],
})
```

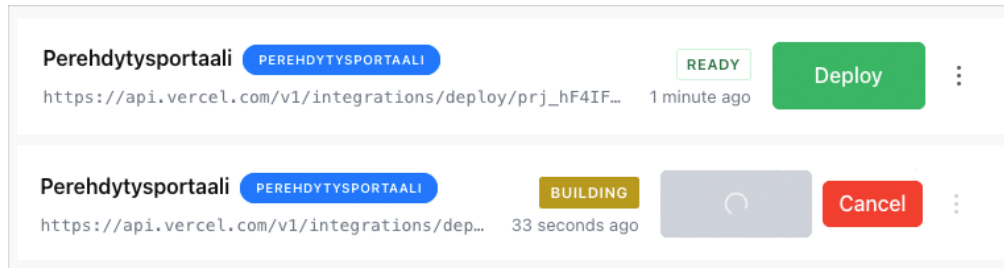
Kuva 17. Vercel Deploy pluginin käyttöönotto Sanityn konfiguraatiossa

Käyttöönoton jälkeen Sanity Studion käyttöliittymään ilmestyy erillinen Deploy-valinta. Deploy-valinnan alla päästään lisäämään deploy-hookki Vercel-projektiin. Kuvassa 18 nähdään deploy-hookin määrittystä varten tarvittavat tiedot sekä ohjeet niiden löytämiseen.



Kuva 18. Sanity - Vercel deploy-hookin määrittys

Kun hookki on luotu, voidaan deploy-prosessi käynnistää painamalla vihreää deploy-painiketta, jolloin Vercel alkaa rakentamaan sovellusta. Rakentamisen yhteydessä haetaan uusimmat sisällöt Sanityn palvelimelta. Lopuksi Vercel julkaisee vasta rakennetun sovelluksen. Kuvassa 19 nähdään deploy-valinnan alle määritetty projekti valmiina (yllä) ja rakennusvaiheessa (alla).



Kuva 19. Vercel-projektin kaksi eri tilänäkymää Sanity studiossa, ready (yllä) ja building (alla)

Tämä deploy-prosessi tulisi käyttäjän käynnistää aina, kun hän on tehnyt muutoksia tai lisäyksiä sisältöön. Näin varmistetaan, että sovelluksen sisällöt ovat aina ajan tasalla niiden päivittämisen jälkeen.

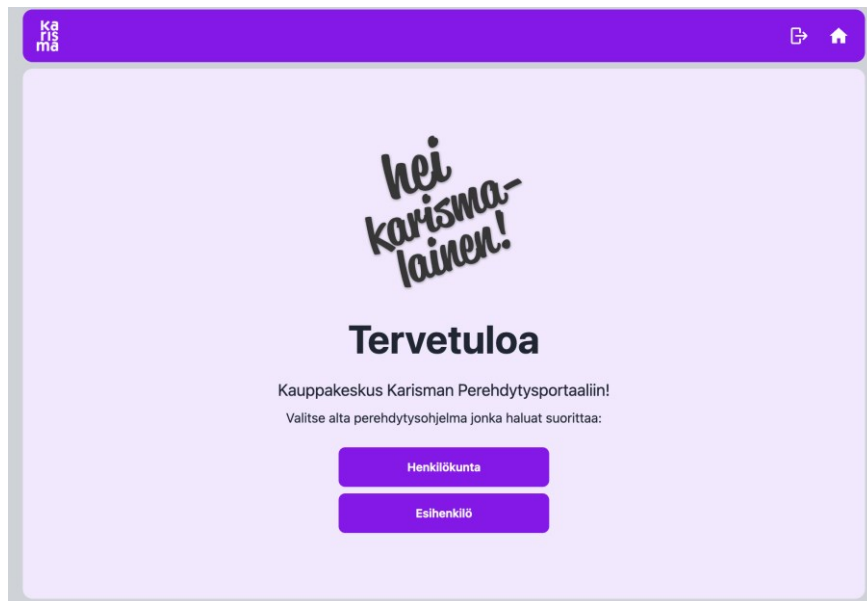
4.2 Lopputulos

Tämän opinnäytetyön ensisijainen tavoite oli kehittää käyttövalmis sovellus, joka tarjoaa mahdollisuuden kauppakeskuksen henkilökunnan perehdyttämiseen. Lisäksi oli tärkeää, että sovellus on helposti saatavilla ajasta ja paikasta riippumatta, ja että kauppakeskuksen johto pystyy itsenäisesti hallinnoimaan ja päivittämään perehdytysmateriaaleja. Kaikki nämä tavoitteet onnistuttiin saavuttamaan.

Sovelluksen paikasta ja ajasta riippumaton käyttömahdollisuus on varmistettu hostaamalla se Vercelin palvelimilla, mikä mahdollistaa pääsyn sovellukseen mistä tahansa internetyhteyden kautta. Tällä hetkellä sovellus on saatavilla väliaikaisessa Vercelin verkko-osoitteessa, mutta tulevaisuudessa siirtyminen kauppakeskuksen omistamaan domainiin on suunnitelmassa.

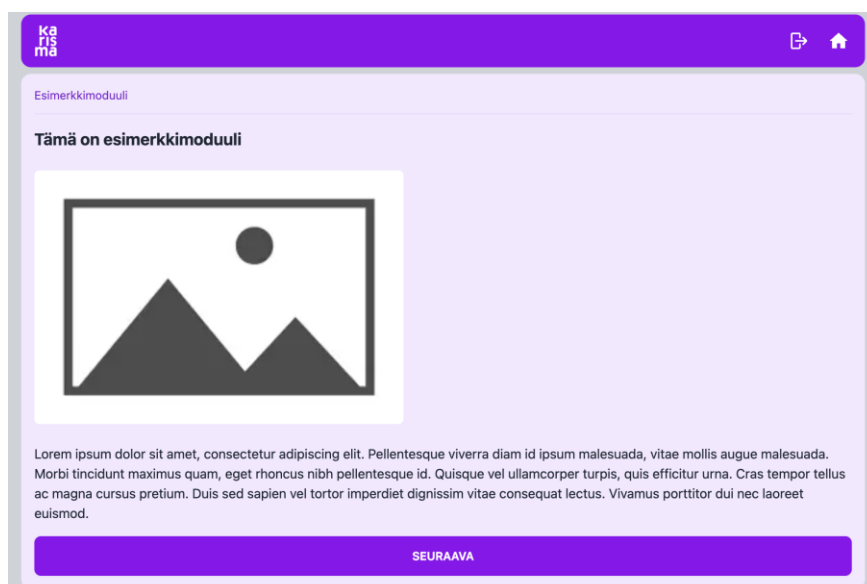
Kauppakeskuksen johto voi hallinnoida perehdytysportaalin sisältöä vaivattomasti Sanity Studion käyttöliittymän kautta. Tämä antaa heille mahdollisuuden päivittää ja muokata materiaaleja reaaliajassa ilman tarvetta teknisen tuen tai

kolmannen osapuolen palveluntarjoajan apuun. Studion käyttöliittymä on lähtökohtaisesti jo varsin suoraviivainen ja helppo käyttää, mutta käyttäjien tueksi sinne on lisätty tarvittavia opasteita käytön helpottamiseksi edelleen. Seuraavissa kuvissa esitellään perehdytysportaalin käyttöliittymää sen eri näkymissä. Kuvassa 20 nähdään perehdytysportaalin etusivu, joka avautuu kirjautumisen jälkeen. Tältä sivulta käyttäjä valitsee minkä ohjelman hän haluaa suorittaa.



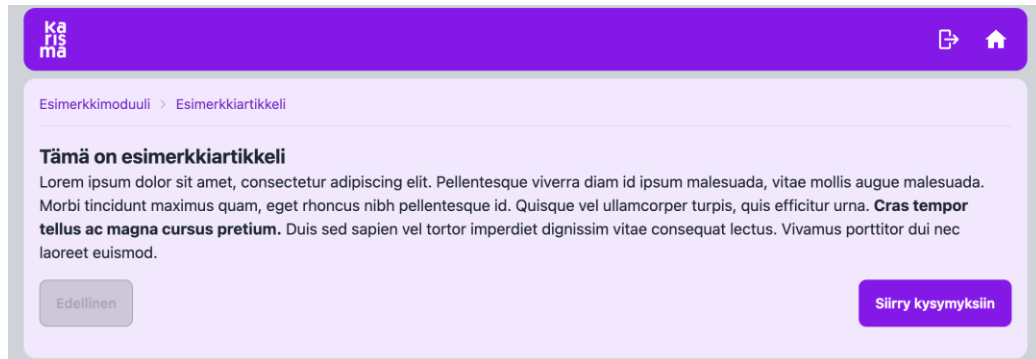
Kuva 20. Perehdytysportaalin etusivu

Kuvassa 21 on moduulin etusivu, jossa kerrotaan kyseisen moduulin aihealueesta yleisesti. Seuraava painikkeella käyttäjä siirtyy moduulin ensimmäiseen artikkeliin.



Kuva 21. Moduulin etusivu

Kuvassa 22 näkyy esimerkki mahdollisesta artikkelista. Jos artikkeli on viimeinen kyseisessä moduulissa, on käyttäjällä mahdollisuus siirtyä moduulin kysymysosioon.



Kuva 22. Artikkelisivu

Kauppakeskuksen ryhtyessä lisäämään materiaalia perehdytysportaaliin, saadaan käyttäjäpalautteen myötä todennäköisesti kehitysideoita tai havaitaan mahdollisia virhetilanteita. Portaalia on toistaiseksi testattu vain tilapäisellä testidatalla.

5 PÄÄTÄNTÖ

Opinnäytetyö on saatu päätökseen, ja sille asetettujen tavoitteiden täytyttyä voidaan sen arvioida onnistuneen kokonaisuutena hyvin. Opinnäytetyö kehittämistehtävän muodossa oli mielestäni mielenkiintoinen ja antoisa prosessi. Projektissa käytettyihin teknologioihin tuli syvennyttyä tällä tapaa enemmän sekä niiden taustoja tutkittua laajemmin joka tuki omaa kehitystä. Koska toimeksiantaja antoi projektin teknologioiden ja toteutuksen osalta vapaat kädet, piti valintoja niiden suhteen harkita huolella, jotta valinnat olisivat järkeviä sekä kestäisivät myös aikaa.

Jos aloittaisin projektin nyt uudelleen, kiinnittäisin erityistä huomiota aikataulutukseen. Nykyisellään projektissa ei asetettu tarkkoja määräaikoja edistymiselle ja valmistumiselle, mikä johti jossain määrin epäjärjestelmälliseen työskentelyyn. Selkeä aikataulu olisi auttanut projektin hallinnassa ja estänyt tarpeetonta poukkoilua eri kehitysvaiheiden välillä. Vaikka tämä lähestymistapa

ei tuottanut ongelmia itsenäisessä työskentelyssäni, ymmärrän, että moniammatillisissa projekteissa tarkka aikataulutus on välttämätöntä tehokkuuden ja koordinoinnin kannalta.

Perehdytysportaalin tulevaisuuden käytön osalta odotan innolla sen suorituskyvyn ja toimivuuden näkemistä käytännössä. On keskeistä arvioida, vaatiiko portaali teknisiä muutoksia tai päivityksiä todellisten perehdytysmateriaalien integroinnin myötä. Ensimmäinen vaihe portaalin julkaisun jälkeen onkin kerätä kehitysideoita Kauppakeskus Karisman johdolta. Tämän jälkeen, kun portaali on otettu aktiiviseen käyttöön henkilökunnan keskuudessa, on tarkoitus järjestää palautekysely myös henkilökunnalle. Tämä auttaa toimeksiantajaa sekä minua kehittäjänä ymmärtämään käyttäjäkokemuksia ja ohjaa sovelluksen jatkokehitystä käyttäjälähtöisesti.

Lopuksi haluan kiittää opinnäytetyön toimeksiantajaa Kauppakeskus Karismaa mahdollisuudesta soveltaa omaa osaamistani tämän mielekkään sovelluskehitysprojektin parissa. Haluan myös kiittää ohjaavaa opettajaani Janne Turusta, jonka kannustava ohjaus ja arvokkaat neuvot auttoivat minua eteenpäin opinnäytetyöprosessissa.

Tämä projekti on ollut minulle merkittävä oppimiskokemus, joka on vahvistanut ammatillista osaamistani ja valmistanut minua tulevaisuuden urahaasteisiin. Olen kiitollinen kaikesta, mitä olen oppinut ja innokas soveltamaan näitä oppeja tulevilla projekteillani.

LÄHTEET

Amsler, S. & Churchville, F. 2021. Content Management System (CMS). WWW-dokumentti. Päivitetty 1.2.2021. Saatavissa: <https://www.tech-target.com/searchcontentmanagement/definition/content-management-system-CMS/> [viitattu 24.1.2024].

Camden, R. & Rinaldi, B. 2017. Working with Static Sites. O'Reilly Media Inc. E-kirja. Saatavissa: <https://learning.oreilly.com/library/view/working-with-static/9781491960936> [viitattu 17.8.2023].

Darji, P. 2023. Traditional CMS vs. Headless CMS: Which Content Management System is Best?. Blogi. Päivitetty 12.4.2023. Saatavissa: <https://syndelltech.com/headless-cms-vs-traditional-%20cms/> [viitattu 16.8.2023].

Hawksworth, P. 2020. What is a Static Site Generator? And 3 ways to find the best one. WWW-dokumentti. Päivitetty 8.12.2009. Saatavissa: <https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/> [viitattu 1.6.2023].

Holmes, J. 2022. The benefits of headless CMSes. WWW-dokumentti. Päivitetty 10.10.2022. Saatavissa: <https://www.sanity.io/headless-cms/headless-cms-benefits> [viitattu 18.5.2023].

Jamstack.org. s.a. Headless CMS. WWW-dokumentti. Saatavissa: <https://jamstack.org/headless-cms/> [viitattu 31.8.2023].

Next.js documentation. s.a. Introduction. WWW-dokumentti. Saatavissa: <https://nextjs.org/docs#main-features> [viitattu 18.5.2023].

Nextjs.org. s.a. About React and Next.js. WWW-dokumentti. Saatavissa: <https://nextjs.org/learn/react-foundations/what-is-react-and-nextjs> [viitattu 2.2.2024].

Sanity Documentation. 2023a. Datasets. WWW-dokumentti. Päivitetty 8.3.2023. Saatavissa: <https://www.sanity.io/docs/datasets> [viitattu 14.9.2023].

Sanity Documentation. 2023b. Getting Started with Sanity Studio. WWW-dokumentti. Päivitetty 12.9.2023. Saatavissa: <https://www.sanity.io/docs/create-a-schema-and-configure-sanity-studio> [viitattu 13.9.2023].

Vercel. 2023a. Configuring a Build. WWW-dokumentti. Päivitetty 6.2.2023. Saatavissa: <https://vercel.com/docs/concepts/deployments/configure-a-build> [viitattu 1.6.2023].

Vercel. 2023b. Getting Started with Vercel. WWW-dokumentti. Päivitetty 6.2.2023. Saatavissa: <https://vercel.com/docs/concepts/get-started> [viitattu 1.6.2023].

Wiśniewski, K. 2021. Headless CMS vs Monolith CMS – What You Need to Know. Blogi. Saatavissa: <https://www.xfive.co/blog/headless-vs-monolith-cms/> [viitattu 17.8.2023].