LAB University of Applied Sciences

Henkilökohtaisen rahan ja rahoitusten full-stack seurantaohjelman kehitys

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikkan
2024
Omar Kraidié

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Omar Kraidié	Opinnäytetyö, AMK	2024
	Sivumäärä 33	

Työn nimi

Henkilökohtaisen rahan ja rahoitusten full-stack seurantaohjelman kehitys

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikan koulutus

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Kraidie Consulting Oy

Tiivistelmä

Opinnäytetyön aiheena oli full-stack verkkokehitys Next.js:n, Shadcn/ui:n, Tailwind CSS:n ja saavutettavuuden näkökulmasta. Työn tavoitteena oli luoda rahan ja rahoitusten seurantaohjelma. Opinnäytetyössä käsitellään myös hakukoneoptimointia ja jatkuvaa julkaisua, erityisesti Next.js:n tarjoamien ominaisuuksien kautta.

Teoriaosuudessa katsotaan Shadcn/ui -käyttöliittymäkirjaston periaatteita ja erottumista perinteisistä kirjastoista. Lisäksi pohditaan Tailwind CSS:n käyttöä käyttöliittymän tyylittelyssä ja saavutettavuuden merkitystä modernin web-sovelluksen kehityksessä.

Käytännön osuudessa esitellään, miten Shadcn/ui voidaan käyttää ja asentaa projektiin. Samalla painotetaan Tailwind CSS:n roolia käyttöliittymän tyylittelyssä. Lisäksi tuodaan esille hakukoneoptimoinnin ja jatkuvan julkaisun näkökulmat, erityisesti Next.js:n tarjoamien ominaisuuksien avulla.

Lopputuloksena on asiakkaan tilauksen mukainen full-stack web-sovellus. Sovellus on jatkokehitettävissä ja asiakas on tyytyväinen.

Asiasanat

rahoitus, verkkokehitys, Next.js, Shadcn/ui, Tailwind CSS, saavutettavuus, jatkuva julkaisu

Abstract

Author(s)	Type of Publication	Published
Omar Kraidié	Thesis, UAS	2024
	Number of Pages	
	33	

Title of Publication

Development of a full-stack personal finance tracker

Degree, Field of Study

Bachelor of Engineering, Information and Communications Technology

Organisation of the client (if the thesis work is commissioned by another party)

Kraidie Consulting Oy

Abstract

The topic of the thesis was full-stack web development from the perspective of Next.js, Shadcn/ui, Tailwind CSS, and accessibility. The goal of this work was to create a money and finance tracking program. It also covers search engine optimization and continuous deployment, especially through the features provided by Next.js.

In the theoretical part, the principles, and distinctions of the Shadcn/ui user interface library are explained, along with the discussion of using Tailwind CSS for styling the interface and the significance of accessibility in modern web application development.

The practical part introduces how Shadcn/ui can be used and installed in a project, emphasizing the role of Tailwind CSS in styling the interface. Additionally, it highlights perspectives on search engine optimization and continuous deployment, particularly through the features offered by Next.js.

The result is a full-stack web application tailored to the client's order. The application is extendable, and the client is satisfied.

Keywords

finance, web development, Next.js, Shadcn/ui, Tailwind CSS, accessibility, continuous deployment

Sisällysluettelo

1 Johdanto

Opinnäytetyön aiheena on modernin full-stack web-sovelluksen kehittäminen ja työn tavoitteena on rakentaa Kraidie Consulting Oy:lle pilottiversio full-stack web-sovelluksesta. Rakennettava sovellus tulee olla React.js ja Next.js -pohjainen rahan ja rahoitusten seurantaohjelma henkilökohtaisten talousasioiden organisointiin ja analysointiin. Asiakkaalle on annettava mahdollisuus ladata oma talousdata ulos sovelluksesta.

Käyttäjän henkilökohtainen data on suojattava tunnistautumisella, eikä se saa paljastua muille käyttäjille. Sovelluksen analysointi ja seurantatyökaluja ei voi käyttää, ellei asiakas ole tunnistautunut. Tunnistautuminen tulee toteuttaa NextAuth.js:än Oauth-teknologialla ja sen Oauth-tarjoajista vähintään kaksi tulisi käyttöönottaa.

Kaikki asiakkaan data tallennetaan itse hallinnoituun tietokantaan rajapintakutsujen ja serverless-funktioiden avulla. Palvelimen ja asiakkaan sovelluksen tilaa tulisi hallita erillisillä teknologioilla. Asynkronisten palvelinkutsujen aikana näytetään käyttöliittymässä erilaisia latauselementtejä.

Valmis sovellus tulee julkaista Vercel.com-palvelimille GitHub:in kautta. Sovelluksen jatkokehittäminen tulisi olla sujuvaa, joka mahdollistaa Kraidie Consulting Oy:lle uusien toiminnallisuuksien tehokkaan lisäämisen.

Työn teoriaosuudessa käsitellään web-sovelluksen osa-alueita, kuten taustajärjestelmää (backend) ja käyttöliittymää (frontend), sekä projektissa käytettyjä työkaluja ja teknologioita, kuten esimerkiksi React.js, Next.js, TypeScript ja TailwindCSS. Käytännön osuudessa toteutetaan full-stack web-sovellus, ja käydään läpi sen keskeiset vaiheet, kuten projektin luonti, tietokannan käyttöönotto, käyttäjän tunnistautuminen, käyttöliittymän kehittäminen ja sovelluksen julkaisu.

2 Full-stack web-sovellus

2.1 Taustajärjestelmä

Taustajärjestelmä eli backend on web-sovelluksen osa, joka toimii taustalla ja vastaa käyttäjän eli asiakkaan pyyntöihin, kuten tietokantakutsuihin. Se hallinnoi sovelluksen toimintaa ja varmistaa tiedon eheyden ja turvallisuuden. Taustajärjestelmä on näkymätön käyttäjälle, mutta olennainen osa kaikissa full-stack web-sovelluksissa. Taustajärjestelmän valinta on tärkeä osa kehitysprosessia, sillä se vaikuttaa merkittävästi sovelluksen suorituskykyyn ja laajennettavuuteen. (Enes 2023.)

Next.js on JavaScript-kehys, joka tarjoaa tavan kehittää full-stack web-sovellusta, painottaen erityisesti palvelinpuolen (server) ja asiakaspuolen (client) renderointia sekä useita hyödyllisiä ominaisuuksia käyttöliittyymäkehityksessä. Next.js 13, versio 13.4, esittelee useita ominaisuuksia, jotka voi tehdä siitä sopivan valinnan myös backend-rajapintojen kehittämiseen. (Enes 2023.)

Next.js sisältää valmiiksi integroidun tuen serverless-funktioille, mikä mahdollistaa päätepisteiden luomisen ilman taustapalvelimen asentamista, mutta samalla tarjoaa myös mahdollisuuden hyödyntää erillistä taustapalvelinta. Serverless funktiot aktivoituvat loppukäyttäjän tarpeen mukaan, mikä tekee niistä kustannustehokkaita. (Enes 2023.)

Lisäksi Next.js tukee TypeScriptiä. Tämä helpottaa tyyppiturvallisen taustapalvelimen koodin kirjoittamista ja virheiden havaitsemista varhaisessa vaiheessa. (Enes 2023.)

Next.js tuo mukanaan tiedostopohjaisen reitityksen, joka mahdollistaa päätepisteiden rakentamisen luomalla selkeän tiedostorakenteen, jossa on ryhmiteltynä sovelluksen päätepisteet omiin hakemistoihin. Tämä yksinkertaistaa reittien hallintaa ja voi auttaa uusia kehittäjiä perehtymään sovelluksen koodipohjaan. Lisäksi se mahdollistaa dynaamisten päätepisteiden käytön. (Enes 2023.)

Next.js 13 integroituu Vercelin alustaan, jonne tämä projekti tulee viime kädessä julkaista omalla alidomainilla. Se tarjoaa tehokkaan ratkaisun sekä pienimuotoisille pilotti- ja prototyyppisovelluksille että mittaville tuotantosovelluksille. (Enes 2023.)

Tämän projektin taustajärjestelmäksi on valittu Next.js 13. Se täyttää asiakkaan asettamat vaatimukset, ja sen käyttöönotto sekä julkaisu Verceliin on nimenomaisesti asiakkaan toiveiden mukainen.

2.2 Käyttöliittymäkerros

Frontend, eli käyttöliittymäkerros, on oleellinen osa modernia web-sovellusta. Se on osa sovelluksesta, jonka loppukäyttäjä näkee ja tulee käyttämään. Käyttöliittymäkerros vastaa siitä, miten sovellus esitetään käyttäjälle selaimessa. Käyttöliittymä voi olla rakennettu erilaisilla teknologioilla ja kielillä ja se voi sisältää kirjastoja, kehyksiä tai ulkoisia paketteja. Käyttöliittymän kehityksessä tulisi valita ne työkalut, jotka sopivat kyseiseen tehtävään helpottaakseen käyttöliittymän rakentamista ja hallintaa. (Hanekcud 2024.)

React on avoimen lähdekoodin JavaScript-kirjasto, joka on suunniteltu käyttöliittymien rakentamiseen keskittyen erityisesti dynaamisten ja tehokkaiden verkkosovellusten kehittämiseen frontendissä. Kun React julkaistiin, se toi mukanaan innostusta mahdollisuudesta kehittää yksisivuisia sovelluksia, mihin React soveltuu erityisen hyvin. React toimii siten, että se toimittaa koko sovelluspaketin käyttäjän selaimelle ja mahdollistaa logiikan kirjoittamisen sovellukseen. (Gupta 2023.)

Kun projektiin lisätään useita ulkoisia paketteja tai sen koodipohja kasvaa, samalla projektin koko kasvaa, mikä saattaa aiheuttaa haasteita käyttäjille. Next.js esittelee käsitteen "koodin jakaminen", joka mahdollistaa pakettien jakamisen ja myöhäisen lataamisen (lazy loading). Tämä keventää dataliikennettä ja tarjoaa mahdollisuuden tallettaa joitakin tiedostoja käyttäjän laitteelle, joka parantaa sovelluksen nopeutta ja suorituskykyä. (Gupta 2023.)

Kun koodipohja kasvaa, suorituskyky yleensä heikkenee. JavaScript-tiedoston lataaminen vaatii aina koodin analysointia ja tulkitsemista selaimessa ennen sen suorittamista, ja mitä suurempi koodipohja on, sitä enemmän aikaa tähän kuluu. Next.js tarjoaa ratkaisun tähän ongelmaan, ja sen vaikutus käyttöliittymäkehitykseen on merkittävä. Tämä kehys mahdollistaa raskaan käyttöliittymälogiikan, kuten datan hakemisen ja monimutkaisen JavaScript-logiikan, siirtämisen palvelinpuolelle, mikä tuo useita etuja käyttöliittymän kannalta. (Gupta 2023.)

Kun raskaat laskennalliset toiminnot ja datan hakeminen siirtyvät palvelinpuolelle, seurauksena on kevyempi käyttöliittymä, joka toimitetaan suoraan käyttäjän selaimelle. Tämä tarkoittaa nopeampaa sivujen latautumista ja parempaa käyttökokemusta, joka on tärkeää nykyaikaisessa web-kehityksessä. Lisäksi Next.js tarjoaa joukon valmiita toiminnallisuuksia, kuten Server-Side Generation (SSG), Server-Side Rendering (SSR) ja Client-Side Rendering (CSR). Nämä ominaisuudet voivat olla helposti osa kehitysprosessia ilman syvällistä teknistä osaamista. Tämä mahdollistaa kehittäjille selkeämmän ja helpommin ymmärrettävän koodin tuottamisen verrattuna aiempiin Next.js-versioihin. Next.js tarjoaa siis useita uusia teknologioita, jotka voivat hyödyttää käyttöliittymäkehittäjiä. (Gupta 2023.)

3 Työkalut

3.1 TypeScript

TypeScript on ohjelmointikieli, joka nousee merkittävään rooliin modernin full-stack web-sovelluksen kehityksessä. Se tarjoaa monia etuja, erityisesti silloin, kun pyritään rakentamaan tehokasta ja luotettavaa sovellusta. Tässä luvussa käsitellään TypeScriptin käyttöä ja sen roolia sovelluksen kehityksessä. (TypeScript.)

TypeScript on JavaScriptiin perustuva ohjelmointikieli, joka tarjoaa staattista tyypitystä. Staattinen tyypitys mahdollistaa virheiden havaitsemisen jo koodia kirjoittaessa, mikä tekee koodin laadunvalvonnasta helpompaa ja tehokkaampaa. (Dreimanis 2023) Tämä on erityisen tärkeää monimutkaisissa järjestelmissä.

TypeScript tarjoaa myös dokumentaatiota ja koodin selkeyttämistä, mikä voi auttaa muita kehittäjiä ymmärtämään koodia ja sen toimintaa. Tämä on erityisen tärkeää suurissa projekteissa, joissa useat kehittäjät työskentelevät yhdessä. (Dreimanis 2023.)

TypeScript on yhteensopiva Reactin ja Nextin kanssa. Se mahdollistaa komponenttien ja logiikan kirjoittamisen vahvalla tyypityksellä, mikä tekee koodista helpommin ylläpidettävää ja vähentää koodivirheitä. TypeScript tehostaa myös IDE-työkalujen ja koodin yhteensopivuutta. (Dreimanis 2023.)

3.2 React.js

Tässä osiossa tarkastellaan React.js:n roolia modernin full-stack web-sovelluksen kehityksessä. React.js on suosittu JavaScript-kirjasto, jonka tavoitteena on mahdollistaa dynaamisten ja interaktiivisten käyttöliittymien kehittämisen web-sovelluksille. Se toimii komponenttipohjaisen arkkitehtuurin periaatteella, ja sen toimintaperiaatteet korostavat selkeyttä, uudelleenkäytettävyyttä ja tehokkuutta käyttöliittymien kehityksessä. (Nextjs a.)

Reactin keskeinen käsite on "komponentti." Komponentit ovat itsenäisiä, modulaarisia koodiosia, jotka voivat sisältää HTML-rakenteita, tyyliohjeita ja toiminnallisuutta. Komponentit mahdollistavat käyttöliittymän jakamisen pienempiin ja helpommin hallittaviin osiin. Kuvassa 1 esitetään kuinka selaimeen voi lisätä autotalli komponentin (Garage), joka sisältää toisen autokomponentin (Car). Modulaarisuus tekee koodista selkeämpää ja helpottaa sovelluksen rakentamista ja ylläpitoa. (W3Schools.)

```
function Car() {
  return <h2>I am a Car!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

Kuva 1. Yksinkertainen esimerkki React.js-komponentin lisäämisestä DOMiin (W3Schools)

Lisäksi React.js toimii tehokkaana kehyksenä Nextin kanssa, joka tarjoaa monia etuja modernien web-sovellusten kehittämisessä kuten parannettu SEO, palvelinpuolen renderaus ja yksinkertaistettu asennus. React.js + Next.js mahdollistaa React- ja React DOM -kirjastojen hallinnan paikallisesti käyttämällä Node Package Manager. (Nextjs a.)

3.3 Next.js

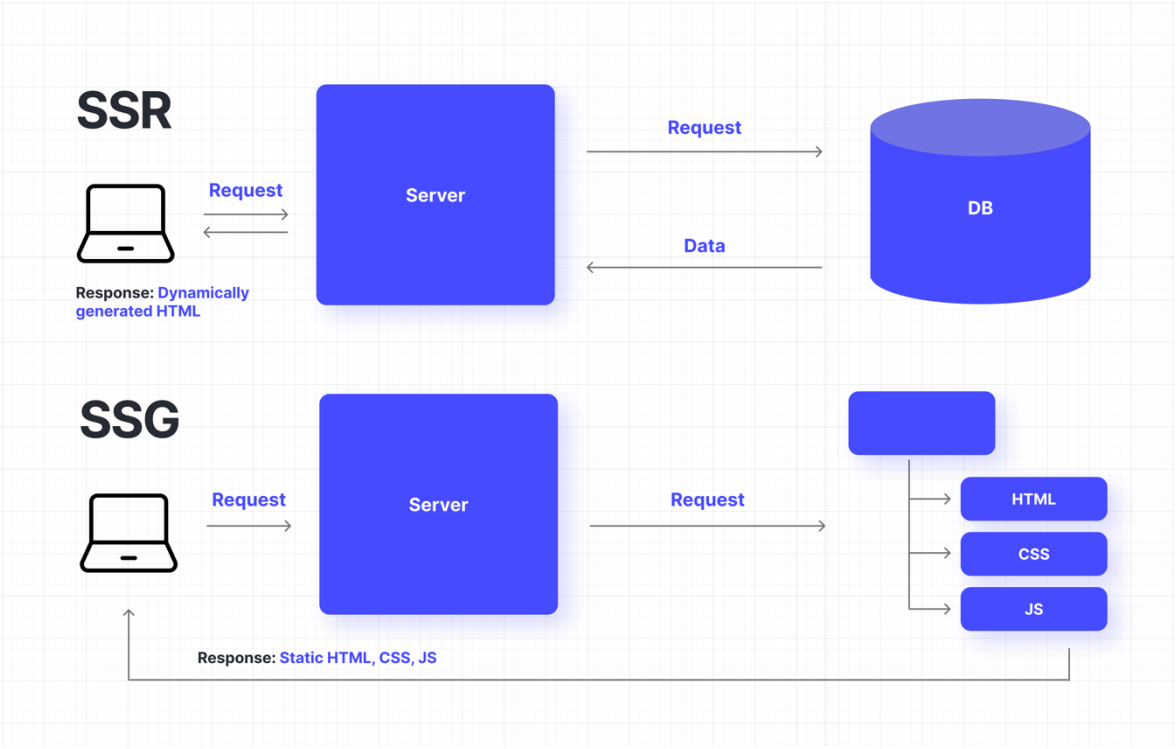
Tässä luvussa käsitellään lyhyesti Next.jsin perusperiaatetta. Samalla vertaillaan sitä palvelinpohjaiseen renderöintiin (SSR) ja staattiseen sivun generointiin (SSG) nähden.

Next.js on moderni web-sovelluskehys, joka perustuu React.js -kirjastoon. Se tarjoaa monia ominaisuuksia, jotka yleensä tekevät siitä sopivan vaihtoehdon sekä loppukäyttäjille että projektin kehittäjille. (Welearncode.)

Next.js on niin sanottu Reactin metakehys (metaframework), mikä tarkoittaa, että se on kehys, joka laajentaa Reactin tarjoamia toimintoja. Next.js tarjoaa kehittäjille lisäominaisuuksia, kuten palvelinpohjaisen renderöinnin, reitityksen ja TypeScript tuen (Welearncode). Lisäksi se tuottaa automaattisesti HTML-rakenteen, ja samalla poistaa tarpeen kirjoittaa tai ylläpitää HTML-pohjaa manuaalisesti. (Nextjs a.)

Palvelinpohjainen renderöinti (SSR) ja staattinen sivun generointi (SSG) ovat kaksi keskeistä tekniikkaa web-sovellusten kehityksessä. SSR:ssä sivun sisältö generoidaan

dynaamisesti palvelimella jokaisen käyttäjän pyynnön yhteydessä mahdollistaen reaaliaikaisen sisällön päivityksen ja hyvän hakukoneoptimoinnin. SSG:ssä sivut luodaan etukäteen ja tallennetaan valmiina tiedostoina palvelimelle, mikä johtaa nopeampiin latausaikoihin ja parempaan suorituskykyyn. Kuvio 1 esittää SSR:n ja SSG:n toimintaperiaatetta ja kuinka ne eroavat toisistaan. (Sajdok 2023.)



Kuvio 1. SSR:n ja SSG:n toimintaperiaate (Sajdok 2023)

Valinta SSG:n ja SSR:n välillä riippuu käyttötapauksesta. SSG sopii sisältöön, joka ei vaadi jatkuvaa päivitystä, kun taas SSR on tehokas valinta dynaamiseen sisältöön. Useissa projekteissa näitä kahta lähestymistapaa voidaan yhdistää tarpeiden mukaan. On tärkeää harkita huolellisesti, kumpi lähestymistapa sopii parhaiten projektin tarpeisiin. (Sajdok 2023.)

3.4 Tailwind CSS

Tailwind CSS on käyttöliittymäkomponenttien kehittämiseen tarkoitettu CSS-kehys, joka tarjoaa nopean ja helpon tavan tyylittää räätälöityjä sovelluksia ilman oman CSS:n kirjoittamista. Tailwind CSS kehittäjät voivat käyttää apuluokkia käyttöliittymärakenteen, värin, välien, typografian ja varjojen tyylittelyyn luodakseen täysin räätälöityjä komponentteja ilman turhaa CSS-koodia. (Fitzgerald 2022.)

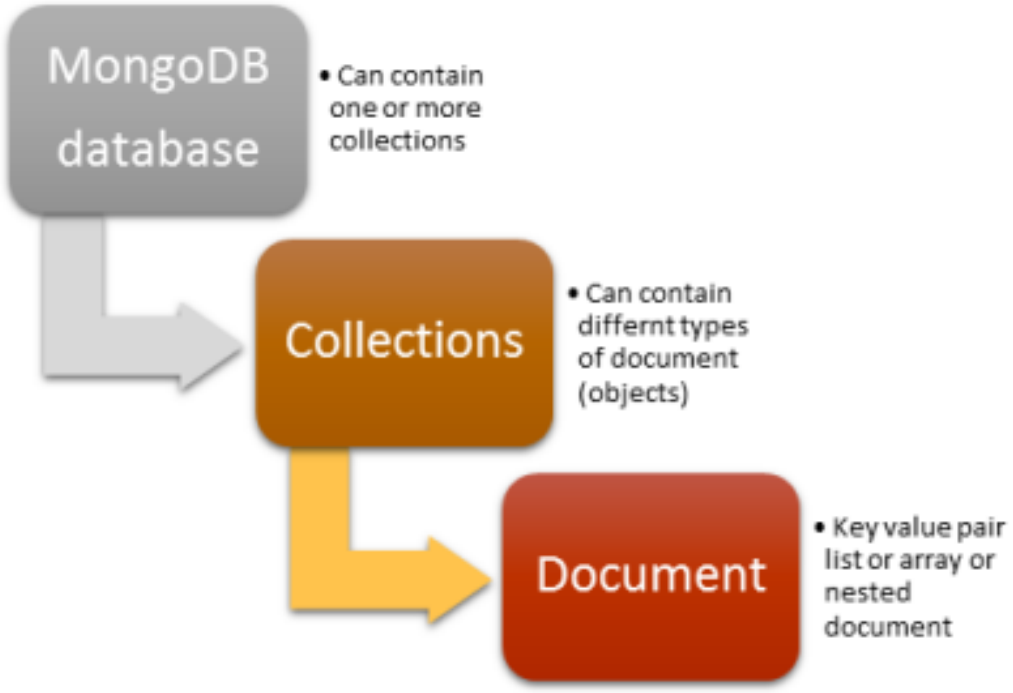
Tailwind CSS eroaa monista muista CSS-kehystyksistä, kuten Bootstrapista ja Materializesta, koska se ei tarjoa täysin muotoiltuja komponentteja, kuten nappeja, alasvetovalikkoja ja navigaatiopalkkeja. Sen sijaan se tarjoaa hyötyluokkia, joiden avulla voi luoda

omia uudelleenkäytettäviä komponentteja. Tämä antaa kehittäjälle enemmän vapauksia sovelluksen ulkoasuun. (Fitzgerald 2022.)

Tailwind CSS on helppo integroida React.js + Next.js-projektiin. Sen voi lisätä projektiin asentamalla sen NPM-paketin ja mukauttamalla projektin asetuksia tarpeiden mukaan. Tailwindin käyttäminen Next.js:ssä ei vaadi monimutkaisia konfigurointeja, joka voi nopeuttaa kehitystyötä ja se integroituu hyvin muiden komponenttikirjastojen kanssa. (Fitzgerald 2022.)

3.5 MongoDB

MongoDB on dokumenttipohjainen NoSQL-tietokanta, jota käytetään suurten datamäärien tallentamiseen. Sen toimintaperiaate poikkeaa perinteisistä relaatiotietokannoista, kuten SQL-tietokannoista. MongoDB hyödyntää kokoelmia ja olioita. Oliot koostuvat avain-arvo-pareista, ja ne muodostavat perusdatayksikön MongoDB:ssä. Kokoelmat puolestaan sisältävät joukon dokumentteja ja toimivat relaatiotietokantojen taulukoiden vastineena. Kuvio 2 esittää MongoDB tietokannan perusrakenteen. (Srivastava 2022.)



Kuvio 2. MongoDB tietokannan rakenne (Srivastava 2022)

Vaikka MongoDB on erityisen hyödyllinen suurten datamäärien ja korkean suorituskyvyn tarpeisiin, se sopii myös pieniin harraste tai pilotti projekteihin. Lisäksi horisontaalinen skaalaus, eli uusien palvelinten lisääminen, on helppoa MongoDB:ssä, koska se on skeematon tietokanta. Skaalauksen hallinta tapahtuu automaattisesti sovelluksen tasolla ilman tarvetta manuaalisille toimenpiteille. Toisin sanoen MongoDB skaalautuu hyvin. MongoDB

tietokannan käyttöönnotto onnistuu Next.js ja React.js sovelluksessa Prisma ORM:in avulla. (Srivastava 2022.)

3.6 Prisma ORM

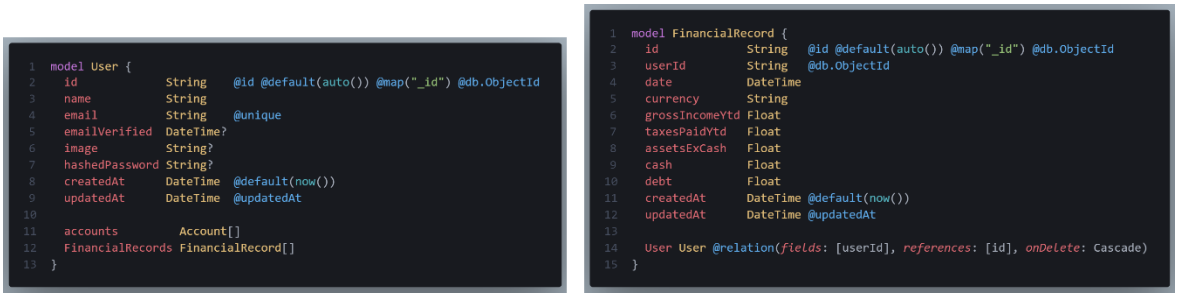
Prisma ORM on moderni Object-Relational Mapping -työkalu, joka helpottaa olioihin perustuvan ohjelmoinnin ja relaatiotietokantojen välistä vuorovaikutusta. ORM-työkalut toimivat välikerroksena, joka mahdollistaa tietokannan ja ohjelman välisen yhteyden. Prisma on yksi näistä ORM-työkaluista. (Shankar 2023.)

ORM:ien käyttö nopeuttaa kehitystyötä ja vähentää kustannuksia tarjoamalla selkeän rajapinnan tietokantaan. Prisma tarjoaa useita etuja, kuten automaattisesti generoidun ja tyypitetyn kyselykielen Node.js- ja TypeScript-sovelluksille sekä integroidun tietokantataulujen hallinnan. (Shankar 2023.)

Yksi Prisman keskeisistä ominaisuuksista on sen TypeScript tyyppiturvallisuus ja selkeä kyselykieli. Prisma tarjoaa Prisma Clientin, Prisma Migraten ja Prisma Studion kaltaisia työkaluja, jotka helpottavat sovelluksen kehittämistä ja hallintaa. (Shankar 2023.)

React-js ja Next.js sovelluksen alustaminen Prisman kanssa on yksinkertaista. Käytännössä tämä edellyttää tarvittavien kirjastojen ja työkalujen asentamista sekä Prisman konfiguroimista projektiin. Tämän jälkeen määritellään tietomallit ja niiden väliset suhteet Prisma Schema -tiedostossa. (Shankar 2023.)

Kuvassa 2 näytetään User- ja FinancialRecord-skeemat. Skeemassa määritellään kokoelmien avain-arvo-parit, kuten id, joka on jokaiselle MongoDB-oliolle uniikki, luomisen ja muokkauksen aikaleimat sekä relaatiot muihin kokoelmiin. Esimerkiksi User-skeemassa rivillä 12 viitataan FinancialRecord-kokoelmaan ja FinancialRecord skeemassa rivillä 14 viitataan User-kokoelmaan. Tässä esimerkissä relaatio avainpari on userId FinancialRecord skeemassa ja kun yksittäisen FinancialRecord:in omistaja (User) poistetaan tietokannasa niin samalla poistetaan sen kaikki talousdata (FinancialRecords).



Kuva 2. Prisman Schema.json-tiedosto

Kun tietokanta on alustettu ja mallit on määritelty, voidaan suorittaa CRUD (Create, Read, Update, Delete) -toimintoja Prisman avulla. Prisma Client tarjoaa selkeän ja tyyppiturvallisen tavan suorittaa nämä toiminnot. (Shankar 2023.)

Kuvassa 3 esitetään miltä tyypillinen Next.js päätepisteen koodi näyttää. Kuvan koodi on selitetty seuraavasti:

- Rivillä 1 määritetään päätepisteen toiminto (DELETE) käyttäen Next.jssän omaa nimeämisstandardia.
- Rivillä 3 käytetään getAuthSession -serverless-funktiota, joka hakee käyttäjän istunnon tiedot.
- Rivillä 5 tarkistetaan istunnon tiedoilla, onko käyttäjä tunnistautunut sovellukseen.
- Rivillä 9 suoritetaan käyttäjän taloustiedon poisto käyttäen Prisman omaa syntaksia. Lopuksi palautetaan poistettu talousolio.
- Koko funktio on try catch koodin sisällä, joka mahdollistaa virhetilanteiden käsittelyn.

```
1  export const DELETE = async (req: Request, { params }: { params: { id: string } }) => {
2    try {
3      const session = await getAuthSession();
4
5      if (!session) {
6        return NextResponse.json({ error: 'Not logged in' }, { status: 401 });
7      }
8
9      const financialRecord = await db.financialRecord.delete({
10       where: { id: params.id },
11     });
12
13     return NextResponse.json({ data: financialRecord });
14   } catch (error) {
15     NextResponse.json({ error: 'Something went wrong' }, { status: 500 });
16   }
17 };
```

Kuva 3. Yksittäisen talousdatan poistamisen päätepistekoodi käyttäen Next.js:ää ja Prisma Clienttia

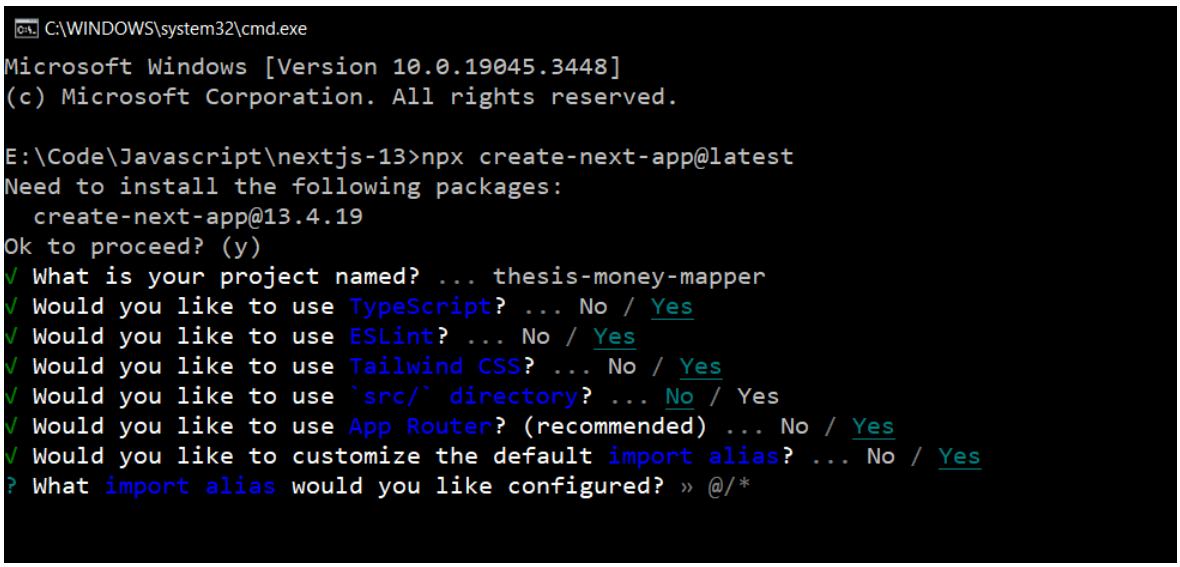
4 Next.js-projekti

4.1 Projektin luonti

Sovelluksen luominen on yksi ensimmäisistä vaiheista sovelluksen kehittämisessä. Se asettaa perustan projektin rakentamiselle ja sillä määritetään tietyt asetukset. Valitut asetukset tukevat sovelluksen käyttötarkoitusta, mutta ne voi myöskin rajoittaa sovelluksen jatkokehitystä, joten harkintaa on käytettävä projektin luomisessa.

Create Next App on Next.js:än yksi oleellisista työkaluista ja se mahdollistaa Next.js-pohjaisten sovellusten luomisen. Tämän työkalun käyttöönotto on ratkaiseva vaihe uuden sovelluksen kehittämisessä, sillä siinä määritellään tärkeitä tulevan sovelluksen asetuksia. Create Next App tarjoaa interaktiivisen määrittelyn ja mahdollistaa sovelluksen perustamisen vain yhdellä komennolla. (Lotanna 2020.)

Uuden projektin luominen on mahdollista Next.js komennolla. Tämä komento käynnistää interaktiivisen prosessin, jossa käyttäjälle esitetään kysymyksiä sovelluksen asetusten määrittämiseksi uutta projektia varten, kuten kuvassa 4 esitetään. Tämä vaihe on kriittinen, sillä siinä valitaan mm. otetaanko projektissa TypeScript käyttöön, käytetäänkö siinä Next.js:n uutta app routeria, käytetäänkö src hakemistoa ja muita tärkeitä asetuksia. (Lotanna 2020.)



Kuva 4. Next.js npx create-next-app komento terminaalissa

Create Next App -pohjaisen projektin voi aloittaa nopeasti, ja siihen ei liity lainkaan ulkopuolisia riippuvuuksia. Aiemmin Next.js sovelluspohjassa oli epävirallisen version kanssa noin 5,4 megatavua riippuvuuksia, mutta optimoinnin jälkeen sovelluksen koko on kutistunut yli 4,7 megatavua. Tämä tarkoittaa, että Create Next App -sovelluksen koko on nyt alle

604 kilotavua asennuksen yhteydessä. Create Next Appin avulla voi siis nopeasti ja helposti aloittaa uuden Next.js-pohjaisen projektin ja hyödyntää kaikkia sen tarjoamia parannuksia ja ominaisuuksia. (Lotanna 2020.)

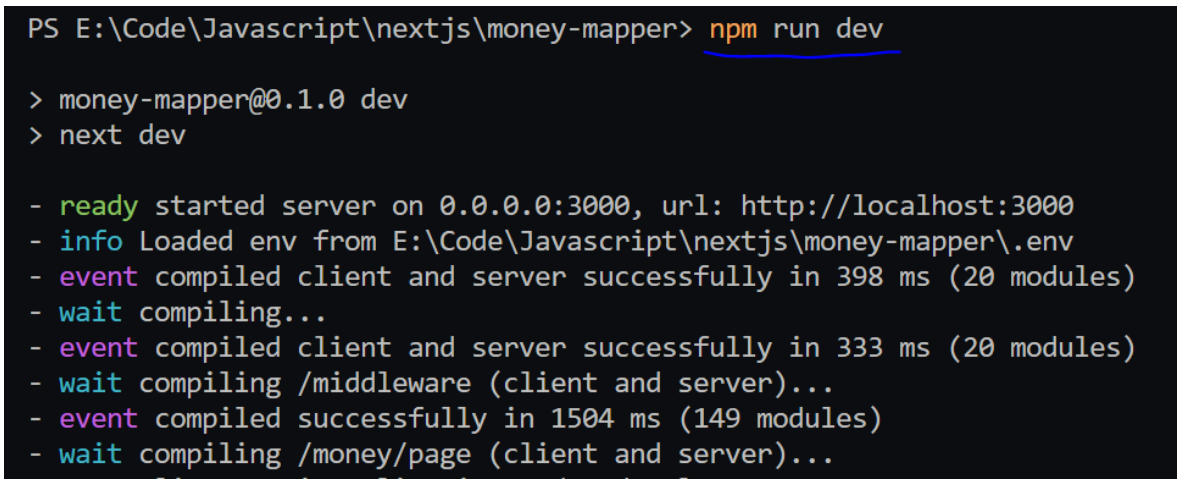
4.2 Kehitys

Kehitysympäristö on oleellinen osa web-sovellusta. Tässä osiossa käsitellään sovelluksen kehitysympäristöä, sen käynnistämistä, ja siihen liittyviä keskeisiä toimintoja.

Kehitystila (development mode) on tila, jossa Next.js-sovellusta suoritetaan paikallisesti kehitystyön aikana. Tämä tila on suunniteltu helpottamaan sovelluksen kehittämistä ja testaamista. Kehitystila mahdollistaa useita hyödyllisiä ominaisuuksia ja toimintoja, jotka helpottavat kehittäjien työtä kuten hot-code reloading, reaaliaikainen esikatselu ja virheraportointi. (Nextjs b.)

Hot-code reloading uudelleenlataus mahdollistaa muutosten näkymisen välittömästi ilman manuaalista päivittämistä tai sovelluksen uudelleenkäynnistämistä, mikä nopeuttaa kehitystyötä ja auttaa havaitsemaan virheet nopeasti. (Nextjs b) Virheraportointi tarjoaa selkeät raportit, jotka auttavat kehittäjiä paikantamaan ja korjaamaan koodissa ilmenevät ongelmat. Lisäksi reaaliaikainen esikatselu mahdollistaa sovelluksen välittömän tarkastelun paikallisesti, mikä tehostaa kehitystyötä ja nopeuttaa projektin etenemistä. (Nextjs b.)

Next.js-sovelluksen voi käynnistää kehitystilassa omalla komennolla, kuten kuvassa 5 esitetään. Tämä komento käynnistää sovelluksen oletuksena 3000 portilla. Portin voi vaihtaa tarvittaessa käyttämällä komennon -p-valitsinta. Portin vaihto on hyödyllistä, jos kehitysympäristössä on muita palveluita käynnissä samalla portilla, ja se mahdollistaa konfliktien välttämisen. Lisäksi portin voi asettaa myös ympäristömuuttujan kautta. (Nextjs b.)



Kuva 5. Next.js npm run dev komento terminaalissa

4.3 Lint

Lint on staattinen koodianalyysityökalu, ja sen tarkoituksena on havaita ohjelmointivirheet, mahdolliset virheet ja korjata koodityylin ongelmat. Lint esiteltiin ensimmäisen kerran vuonna 1978, ja siitä lähtien se on ollut olennainen osa ohjelmistokehitystä. (Devfaysalkhan 2023.)

ESLint on avoimen lähdekoodin JavaScript-työkalu, jonka avulla voidaan asettaa standardityyli projektissa käyttämällä tiettyä joukkoa itsenäisiä sääntöjä. ESLint on täysin mukautettavissa, mikä tarkoittaa, että kehittäjä voi mukauttaa ja konfiguroida sen kehitystarpeiden mukaan. (Devfaysalkhan 2023.)

ESLint toimii kehittäjän apuna virheiden aikaisessa tunnistamisessa kehitysprosessin alkuvaiheissa. Se suorittaa koodianalyysin, havaitsee mahdolliset ongelmat, kuten virheellisesti syötetyt parametrit tai puuttuvat koodimääritykset. Ilman ESLintiä kehittäjä saattaisi olla tietämätön virheistä kehityksen aikana, mikä puolestaan voisi johtaa suoritusaikavirheisiin. ESLint mahdollistaa näiden ongelmien varhaisen havaitsemisen edistäen siten koodin laadun parantamista. Tämä työkalu myös edistää yhdenmukaisten koodauskäytänteiden ylläpitämistä, havaitsee potentiaaliset virheet ja nostaa koodin yleistä laatua. ESLintin integrointi on olennainen osa kehitysprosessia, joka varmistaa, että lopullinen sovellus toimii sujuvasti ilman ennalta-arvaamattomia virheitä. (Devfaysalkhan 2023.)

Projektin kehitysvaiheessa on suositeltavaa suorittaa ESLint-testejä säännöllisesti varmistaakseen, että koodi noudattaa sovittuja käytäntöjä ja että mahdolliset virheet havaitaan ja korjataan ajoissa. ESLintin käyttäminen on yksi askel kohti luotettavan ja laadukkaan ohjelmiston kehitystä. (Devfaysalkhan 2023.)

Next.jssän lint-komento tarkistaa kaikki tiedostot ja niiden koodit ESLintin avulla Next.js-projektissa. Kansiot, kuten pages/, app/, components/, lib/ ja src/, käydään rivi riviltä läpi varmistaen, että koodi noudattaa ESLintin määrittämiä tarkastussääntöjä. Kun lint komento on ajettu onnistuneesti kehityksessä voi kehittäjä olla varma, että projektin rakennus vaiheessa ei ilmene yllättäviä ongelmia. (Nextjs a.)

Kuvassa 6 suoritetaan lint-komento, joka varoittaa kehittäjää puuttuvista koodiriippuvuuksista (form). Riippuvuus puuttuu tiedostoista CreateFinancialRecordForm.tsx riviltä 80, sarakkeesta 5, ja EditFinancialRecordForm.tsx riviltä 64, sarakkeesta 5.

```
PS E:\Code\Javascript\nextjs\money-mapper> npx next lint
- info Loaded env from E:\Code\Javascript\nextjs\money-mapper\.env

./app/money/components/CreateFinancialRecordForm.tsx
80:5  Warning: React Hook useEffect has a missing dependency: 'form'. Either include it or remove the dependency array.  react-hooks/exhaustive-deps

./components/Modals/EditFinancialRecordForm.tsx
64:5  Warning: React Hook useEffect has a missing dependency: 'form'. Either include it or remove the dependency array.  react-hooks/exhaustive-deps

info  - Need to disable some ESLint rules? Learn more here: https://nextjs.org/docs/basic-features/eslint#disabling-rules
PS E:\Code\Javascript\nextjs\money-mapper>
```

Kuva 6 Next.js npx next lint komento terminaalissa

4.4 Build

Build-komento on tärkeä osa web-sovelluksen kehitystä, ja sen avulla luodaan optimoitu tuotantoversio sovelluksesta. Rakentaessa sovelluksen Next.js:llä, saadaan tuotantoversio, joka on optimoitu tehokkaaseen toimintaan. Build komento ajetaan, kun sovellus halutaan rakentaa julkaisua varten. (Nextjs c.)

Yksi tärkeä tieto, jonka build-komento tarjoaa, on sovelluksen koon tarkastelu. Koko ilmaisee niiden resurssien määrän, jotka ladataan, kun siirrytään projektin sivulta toiselle asiakaspuolella (client). Tämä sisältää kunkin reitin koon ja siihen liittyvät riippuvuudet. Lisäksi komento tarjoaa tiedon ensimmäisen latauksen JavaScript-resursseista, jotka ladataan palvelimelta, kun käyttäjä vierailee sivustolla ensimmäistä kertaa, kuten kuvassa 7 listataan. (Nextjs c.)

```
Route (app)                              Size     First Load JS
┌ λ /                                    4.25 kB         100 kB
├ λ /api/auth/[...nextauth]              0 B                0 B
├ λ /api/financial-records               0 B                0 B
├ λ /api/financial-records/[id]          0 B                0 B
├ λ /dashboard                           114 kB          241 kB
├ ○ /favicon.ico                         0 B                0 B
└ λ /money                               187 kB          342 kB
+ First Load JS shared by all            85.4 kB
  ├ chunks/596-fe55617523f65a71.js       25.8 kB
  ├ chunks/677-6fd8604f93b63042.js       7.04 kB
  ├ chunks/fd9d1056-7afd34c3ff6bbcfc.js  50.5 kB
  ├ chunks/main-app-1d2e6ea6533d4520.js  218 B
  └ chunks/webpack-d3a840934cbf8330.js   1.79 kB

Route (pages)                            Size     First Load JS
─ ○ /404                                 183 B            76 kB
+ First Load JS shared by all            75.8 kB
  ├ chunks/framework-43665103d101a22d.js 45.1 kB
  ├ chunks/main-82944172a0be9bbe.js      28.8 kB
  ├ chunks/pages/_app-6b79a29ad0d63b21.js 199 B
  └ chunks/webpack-d3a840934cbf8330.js   1.79 kB

ƒ Middleware                             52.9 kB

λ  (Server)  server-side renders at runtime (uses getInitialProps or getServerSideProps)
○  (Static)  automatically rendered as static HTML (uses no initial props)
```

Kuva 7. Sovelluksen rakenne terminaalissa onnistuneen build komennon jälkeen

Näitä arvoja pakataan gzip-algoritmilla, ja ensimmäisen latauksen laatu ilmaistaan väreillä: vihreä, keltainen tai punainen. Vihreä viittaa suorituskykyiseen sovellukseen, kun taas keltainen ja punainen voivat viitata optimointitarpeisiin. (Nextjs c.)

5 Tietokannan käyttöönotto

5.1 MongoDB

Modernin full-stack web-sovelluksen kehityksessä tietokanta on keskeinen osa, joka mahdollistaa käyttäjän datan tallentamisen ja tehokkaan hallinnan. Käytännössä tietokanta tarjoaa paikan, jossa sovellus voi pysyvästi tallentaa ja organisoida monimutkaista tietoa, joka liittyy käyttäjän toimintaan ja sovelluksen toiminnallisuuteen.

Tietokannan luominen MongoDB:ssa on ensisijaisesti sidoksissa valittuun kehitysalustaan, tässä tapauksessa React.js ja Next.js -pohjaiseen sovellukseen. Ensimmäinen vaihe on ottaa käyttöön MongoDB Atlas UI, joka tarjoaa visuaalisen käyttöliittymän tietokannan hallintaan. Tässä vaiheessa kehittäjä rekisteröityy Atlas-tilille, joka tarjoaa ilmaisen tason klusterin perustoiminnallisuuksilla. Tämä klusteri tarjoaa olennaiset resurssit sovelluksen alkuvaiheessa tarvittavan tietokannan käyttöönottoon. Aktiivisen klusterin hallintapaneelista on mahdollista luoda tietokanta, jonka jälkeen sen voi yhdistää sovellukseen. (MongoDB.)

Kun tietokanta on luotu niin se on valmis vastaanottamaan ja tallentamaan dataa. Se muodostaa perustan Prisma ORM:n tulevalle integraatiolle. Prisma ORM kykenee liittymään MongoDB-tietokantaan ja tarjoamaan sovellukselle tehokkaat työkalut datan hallintaan ja manipulointiin.

5.2 Prisma

Full-stack web-sovelluksen kehityksessä tietokantayhteyksiä on hallittava jollain tapaa. Yksi tehokas tapa hallita tietokantaintegraatioita on käyttää ORM (Object-Relational Mapping) -työkaluja, joista yksi merkittävä vaihtoehto on Prisma. Tämä osio käsittelee Prisman käyttöä ja sen integroimista MongoDB-tietokantaan.

ORM-järjestelmät helpottavat tietokantaintegraatioita tarjoamalla abstraktiotason tietokantakyselyille ja -toiminnoille. Niiden avulla kehittäjät voivat käsitellä tietokantaa olio-suuntautuneesti, joka voi tehdä koodista ylläpidettävämpää ja helpottaa tietokantamallin muutosten hallintaa.

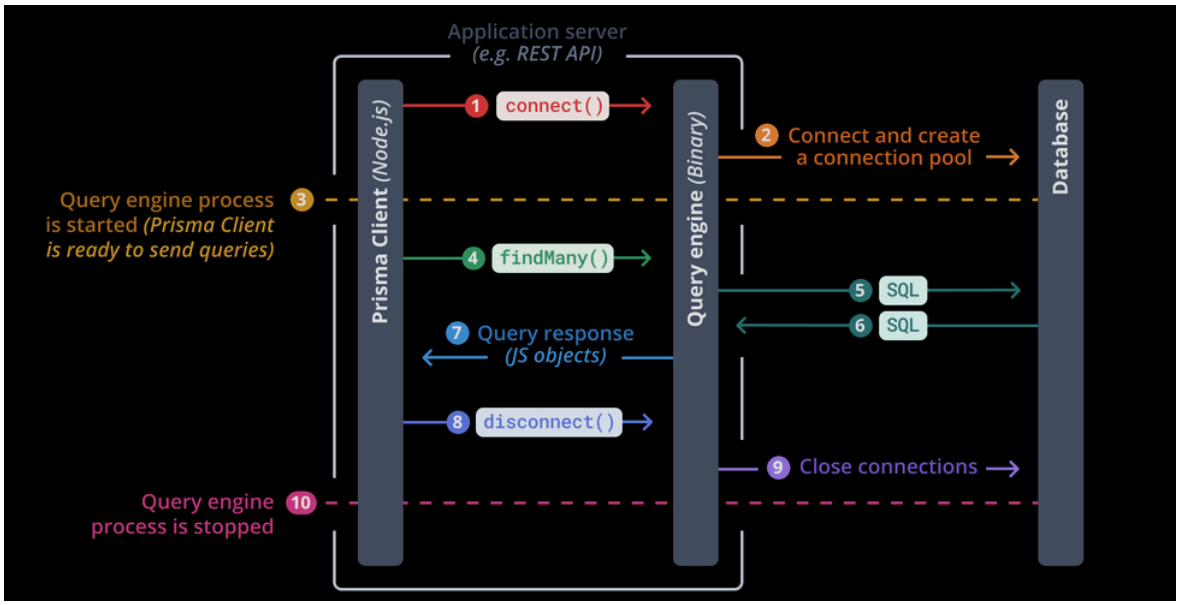
MongoDB-tietokantaan Prisman avulla yhteyden muodostaminen edellyttää ensin tietokannan yhteysosoitteen määrittämistä Prisma-skeemassa (Prisma a). Osoite kannattaa tallentaa ympäristömuuttujaan, jotta salaista yhteysosoitetta ei vahingossa näytetä ulkopuolisille. Ympäristömuuttujia voi määrittää esimerkiksi projektin juuressa olevassa .env-tiedostossa. Yhteysosoitteen muoto riippuu käytetystä tietokantajärjestelmästä. MongoDB:n osalta se noudattaa kuvassa 8 käytettyä rakennetta.

```
mongodb://USERNAME:PASSWORD@HOST:PORT/DATABASE
```

Kuva 8. MongoDB yhteysosoitteen rakenne (Prisma a)

Kun Prisma-skeema on valmiina, Prisma Client tulee keskeiseksi komponentiksi tietokantakäsittelyssä. Prisma Client tarjoaa turvallisen rajapinnan skeeman perusteella generoiduille tietokantakyselyille ja se täytyy asentaa projektiin. Prisma Clientin avulla kehittäjä voi suorittaa tietokantatoimintoja sovelluksen vaatimusten mukaisesti. (Prisma b) Kuviossa 3 on esitetty Prisman toimintamalli vaiheittain. Kuvion vaiheet on selitetty seuraavasti:

1. Käyttöliittymäkoodissa kutsutaan Prisma Clientin connect funktiota
2. Kyselymoottori käynnistyy ja se muodostaa yhteydet tietokantaan.
3. Prisma Client on valmis lähettämään kyselyjä tietokantaan.
4. Prisma Client lähettää findMany() -kyselyn joka on Prisma ORMin omaa syntaksia kyselymoottorille.
5. Kyselymoottori kääntää kyselyn ja lähettää sen tietokantaan.
6. Kyselymoottori vastaanottaa SQL-vastauksen tietokannasta.
7. Kyselymoottori palauttaa tuloksen tavallisina JavaScript-objekteina Prisma Clientille.
8. Prisma disconnect() funktiota kutsutaan käyttöliittymäkoodissa.
9. Kyselymoottori sulkee tietokantayhteydet.
10. Kyselymoottori pysäytetään.



Kuvio 3. Prisma Clientin toimintamalli (Prisma b)

Kun Prisma Client on asennettu ja konfiguroitu, voidaan suorittaa tietokantakyselyjä sovelluksessa. Prisma Client toimii sovelluksen ja tietokannan välisenä väliohjelmistona, ja se mahdollistaa vuorovaikutuksen tietokannan kanssa. Kehittäjä muodostaa kyselyt käyttäen Prisma Clientin tarjoamia funktioita ja malleja. Nämä kyselyt perustuvat Prisma-skeemassa määriteltyihin tietokantamalleihin. Kyselyt suoritetaan asynkronisesti parantaen sovelluksen suorituskykyä ja responsiivisuutta. Prisma Clientin avulla saadut tulokset käsitellään ja integroidaan suoraan sovellukseen. (Prisma b.)

Kuvassa 9 esitetään miten Prisma Clienttia käytetään Next.js päätepisteen koodissa. Kuvan koodi on selitetty seuraavasti:

- Rivillä 1 määritetään päätepisteen toiminto (GET) käyttäen Next.jssän omaa nimeämisstandardia.
- Rivillä 3 haetaan käyttäjän istuntotiedot getAuthSession funktiolla.
- Rivillä 5 tarkistetaan, onko käyttäjä tunnistautunut sovellukseen.
- Rivillä 9 haetaan käyttäjän kaikki taloustiedot nousevassa luontipäivämääräjärjestyksessä Prisma Client syntaksilla.
- Rivillä 14 lisätään taloustietoihin käyttäjän nettovarallisuus ja kokonaisvarat suorittamalla laskutoimituksia. Lopuksi palautetaan manipuloitu käyttäjädata.
- Koko funktio on try-catch koodin sisällä virhetilanteiden käsittelyä varten.

```
1  export const GET = async (req: Request) => {
2    try {
3      const session = await getAuthSession();
4
5      if (!session) {
6        return NextResponse.json({ error: 'Not logged in' }, { status: 401 });
7      }
8
9      const financialRecords = await db.financialRecord.findMany({
10       where: { userId: session?.user.id },
11       orderBy: { date: 'asc' },
12     });
13
14     const financialRecordsWithNetWorth: (FinancialRecord & { netWorth: number })[] =
15       financialRecords.map((collection) => {
16         const netWorth = collection.assetsExCash + collection.cash - collection.debt;
17         const totalAssets = collection.assetsExCash + collection.cash;
18         return { ...collection, totalAssets, netWorth };
19       });
20
21     return NextResponse.json({ data: financialRecordsWithNetWorth });
22   } catch (error) {
23     NextResponse.json({ error: 'Something went wrong' }, { status: 500 });
24   }
25 };
```

Kuva 9: Prisma Clientin avulla suoritettu käyttäjän talousdatan tietokantakyselyn koodi

6 Tunnistautuminen

6.1 NextAuth

Tunnistautuminen on olennainen osa full-stack sovelluksen kehitystä, ja sen tavoitteena on varmistaa käyttäjän turvallinen ja luotettava pääsy sovellukseen. Tunnistautuminen on prosessi, jossa käyttäjä vahvistaa oman henkilöllisyytensä. Tämä on erityisen tärkeää raha- ja rahoitusten seurantaohjelmassa, kuten web-sovelluksessa, jossa käyttäjän henkilökohtaista ja välillä jopa salaista talousdataa käsitellään.

NextAuth.js on avoimen lähdekoodin tunnistautumisratkaisu Next.js-sovelluksille. Se on suunniteltu erityisesti tukemaan Next.js- ja Serverless-arkkitehtuureja tarjoten monipuolisen ja joustavan lähestymistavan käyttäjien tunnistautumiseen. Tämä ratkaisu mahdollistaa turvallisen tavan lisätä tunnistautumistoiminnallisuus sovellukseen. (NextAuth.js a.)

NextAuth.js:än Prisma Adapter -teknologia mahdollistaa käyttäjätietojen tallentamisen MongoDB-tietokantaan tunnistautumisen yhteydessä. Lisäksi NextAuth.js tukee OAuth-teknologiaa käyttäjän tunnistautumisessa. (NextAuth.js a.)

OAuth eli avoin valtuutus on laajalti käytetty valtuutuskehys, joka mahdollistaa suostumuksen antamisen tietyn sovelluksen kommunikoinnille toisen sovelluksen kanssa käyttäjän puolesta ilman salasanan paljastamista. Tämä onnistuu tarjoamalla pääsytunnisteita kolmannen osapuolen palveluille, kuten Google tai GitHub, paljastamatta käyttäjän tunnistetietoja. Tunnistautumisen yhteydessä käyttäjän tiedot tallennetaan MongoDB-tietokantaan.

6.2 Prisma adapter

Adapterit ovat ohjelmistokomponentteja, jotka mahdollistavat eri osien yhteensopivuuden. Ne toimivat liittymäkerroksena eri ohjelmistokomponenttien välillä, kuten tunnistautumisen ja tietokantayhteyksien hallinnan välillä. Tunnistautumisadapterit ovat keskeisiä toteutettaessa turvallisia ja skaalautuvia web-sovelluksia.

Prisma-adapteri on erityinen adapteri, joka mahdollistaa NextAuth.js:n ja Prisma-database-sovittimen yhteistyön. Se tarjoaa integroidun tavan tallentaa käyttäjän tunnistautumistiedot Prisma-tietokantaan tunnistautumisen yhteydessä, varmistaen samalla tietoturvan ja järjestelmän eheyden.

Prisma-adapteri toimii liittymäkohtana NextAuth.js:n ja Prisma-database-sovittimen välillä. Sen avulla käyttäjän tunnistautumistiedot tallennetaan tietokantaan turvallisesti. Tämä mahdollistaa sovelluksen hyödyntää Prisma-database-sovittimen tarjoamia etuja, kuten tietokantakyselyjen helppous ja skaalautuvuus. (NextAuth.js b.)

Prisma-adapterin käyttöönotto tapahtuu asentamalla tarvittavat kirjastot, kuten @prisma/client ja @auth/prisma-adapter. Adapteri määritellään sitten NextAuth.js-konfiguraatiossa, liittämällä se siihen ja tarjoamalla sille PrismaClient-instanssi. (NextAuth.js b.)

Kuvassa 10 esitetään NextAuth.js konfiguraatiokoodi, jossa alussa ensin haetaan tarvittavat resurssit. Seuraavaksi alustetaan uusi PrismaClient liitettäväksi tunnistautumisen määrittely olioon. Lopuksi luodaan NextAuth olio, jonka adapter avaimen arvoksi määritetään PrismaAdapter ja providers listaan lisätään GoogleProvider, joka mahdollistaa OAuth tunnistautumisen.

pages/api/auth/[...nextauth].js

```
import NextAuth from "next-auth"
import GoogleProvider from "next-auth/providers/google"
import { PrismaAdapter } from "@auth/prisma-adapter"
import { PrismaClient } from "@prisma/client"

const prisma = new PrismaClient()

export default NextAuth({
  adapter: PrismaAdapter(prisma),
  providers: [
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
    }),
  ],
})
```

Kuva 10. Prisma adapterin konfiguraatio projektissa (NextAuth.js b)

6.3 OAuth tunnnistautuminen

OAuth (Open Authorization) on avoimen standardin mukainen protokolla, joka mahdollistaa käyttäjän tunnistautumisen ja valtuuttamisen eri sovelluksissa ja palveluissa. Se tarjoaa turvallisen tavan jakaa käyttäjän tietoja eri palveluiden välillä ilman, että salasanat tai muita arkaluontoisia tietoja tarvitsee paljastaa. Tämä luku käsittelee OAuth-tunnistautumista ja sen merkitystä modernin full-stack web-sovelluksen kehityksessä.

OAuth-tunnistautuminen mahdollistaa käyttäjän kirjautumisen sovellukseen kolmannen osapuolen palveluntarjoajan kautta, kuten Google tai GitHub. Se on osa nykyaikaisten web-sovellusten kehitystä, sillä se tarjoaa turvallisen ja standardoidun tavan hallita käyttäjän tunnistautumista ilman, että sovellus tarvitsee säilyttää käyttäjän salasanoja. Tämä parantaa tietoturvaa ja käyttäjien luottamusta sovellukseen. Lisäksi tämä parantaa käyttäjäkokemusta ja vähentää tarvetta luoda ja muistaa erillisiä käyttäjätunnuksia ja salasanoja jokaiselle sovellukselle.

Tunnistautumisen tarjoajat (providers) ovat palveluita tai kirjastoja, jotka tarjoavat tavan autentikoida käyttäjät ja hallita heidän kirjautumistietojaan. Ne voivat integroitua eri kirjautumistapoihin, kuten OAuth, Google-tili, GitHub ja muut vastaavat palvelut. Näiden tarjoajien tehtävänä on auttaa sovellusta varmentamaan käyttäjän identiteetti ja tarjoamaan turvallisen tunnistautumisen. (NextAuth.js c.)

NextAuth tukee useita valmiiksi määriteltyjä tarjoajia, kuten Google ja GitHub, sekä mahdollisuuden luoda omia mukautettuja OAuth-konfiguraatioita. OAuth-tunnistautumisen käyttö Next.js:ssä edellyttää, että kehittäjä rekisteröi sovelluksensa tarjoajan kehittäjäportaalissa ja määrittelee oikeat päätepisteet, kuten ohjauksen ja tokenin hankinnan. Tämä varmistaa, että sovellus voi luotettavasti kommunikoida tarjoajan kanssa turvallisesti ja tehokkaasti. (NextAuth.js c.)

6.4 Reittien suojaus

Tämän projektin yksi keskeisistä osa-alueista on web-sovelluksen tietoturvan varmistaminen, erityisesti käyttäjän henkilökohtaisen talousdatan suojaaminen. Koska sovellus tarjoaa käyttäjälle mahdollisuuden tallentaa ja hallita arkaluonteista taloudellista tietoa, on hyvä varmistaa, että vain oikeutetut eli tunnistautuneet käyttäjät voivat selata salaisia sivuja ja käyttää niitä.

Käyttäjän talousdataa voi tarkastella kahdella eri suojatulla sivulla, 'Money'- ja 'Dashboard'-sivuilla. Money-sivulla käyttäjä voi lisätä, poistaa tai muokata omaa dataansa, ja se on esitetty taulukkomuodossa. Dashboard-sivulla kaikki Money-sivun tiedot, eli käyttäjän data, on esitetty kaavioina eri luokkiin jaoteltuina. Sovelluksen sivuja voi selata käyttäen navigointipalkkia. Lisäksi, käyttäjän tiedot ja toiminnot kuten uloskirjautuminen näytetään samassa navigointipalkissa.

Reittien suojaaminen on osa web-sovelluskehitystä, erityisesti kun käsitellään henkilökohtaista dataa tai halutaan rajoittaa pääsyä tiettyihin toiminnallisuuksiin. Sovelluksessa käyttäjät voivat tallentaa henkilökohtaista taloustietoa, ja on kriittistä estää tunnistamattomia

käyttäjiä pääsemästä käsiksi esimerkiksi toisen käyttäjän taloustietopaneeliin tai sen toimintoihin.

Next.js tarjoaa useita tapoja suojata sovelluksen reittejä. Reittejä voi suojata käyttöliittymäpuolella sekä palvelinpuolella ja käyttää välikerrospohjaista suojausta. Näiden eri menetelmien yhdistelmä mahdollistaa kattavan turvallisuuden varmistamisen sovelluksessa. (Chidera 2023.)

Käyttöliittymäpuolessa tapahtuva reittien suojaus on soveltuva skenaarioihin, joissa halutaan estää tunnistautumattomia käyttäjiä pääsemästä tietyille sivuille tai sovelluksen osille. Tämä voidaan toteuttaa esimerkiksi käyttämällä Reactin useLayoutEffect-koukkua yhdessä Next.js:n tarjoaman reitityksen kanssa. Tämä mahdollistaa nopean ja yksinkertaisen tavan suojata reittejä ilman monimutkaisia tunnistautumisprosesseja. (Chidera 2023.)

Palvelinpuolen suojaus taas on oletusarvo Next.js-komponenteille ja se on tapa varmistaa, että palvelimella renderöity sisältö on suojattu. Tätä käytetään tyypillisesti tilanteissa, joissa reittejä ei saa olla saatavilla tunnistautumattomille käyttäjille varmistaen, ettei herkkää käyttäjädataa paljastu. (Chidera 2023.)

Lisäksi välikerrospohjainen reittien suojaus tarjoaa tehokkaan tavan hallita ja valvoa pääsyä tiettyihin reitteihin sovelluksessa. Välikerros-funktioiden avulla voidaan vastaanottaa saapuvia pyyntöjä ja soveltaa räätälöityjä sääntöjä reittien saavutettavuuden suhteen. Tämä lähestymistapa soveltuu erityisesti monimutkaisempiin sovelluksiin, joissa tarvitaan hienovaraista kontrollia reittien käytettävyyden suhteen. (Chidera 2023.)

Välikerroksen käyttöönotto Next.js:sässä tapahtuu luomalla middleware-funktio, joka käsittelee saapuvia pyyntöjä ennen niiden siirtymistä varsinaiseen reitin käsittelyyn. Tämä funktio määritellään ja liitetään sovellukseen yleensä käyttäen reittien konfigurointiin tarkoitettua osaa. Kuvassa 11 esitetään sovelluksen middleware funktion koodia, jolla suojataan ”dashboard” ja ”money” reitit. Kuvan koodi on selitetty seuraavasti:

- Rivillä 1 määritetään asynkroninen Next.js värikerrosfunktio (middleware).
- Riveillä 2–4 alustetaan muuttujia, joita käytetään reittien tarkistuksessa.
- Riveillä 6–7 ensin haetaan käyttäjän istunnon tiedot, jonka jälkeen määritellään isAuth muuttujaan onko käyttäjä tunnistautunut vai ei.
- Lopuksi riveillä 9–11 ja 13–15, ohjataan joko tunnistautunut käyttäjä halutulle reitille tai tunnistautumaton käyttäjä kirjautumissivulle.

```
export async function middleware(request: NextRequest) {
  const sensitiveRoutes = ['/dashboard', '/money'];
  const publicRoutes = ['/'];
  const pathname = request.nextUrl.pathname;

  const token = await getToken({ req: request });
  const isAuth = !!token;

  if (isAuth && publicRoutes.some((route) => route === pathname)) {
    return NextResponse.redirect(new URL('/money', request.url));
  }

  if (!isAuth && sensitiveRoutes.some((route) => pathname.startsWith(route))) {
    return NextResponse.redirect(new URL('/', request.url));
  }
}
```

Kuva 11. Projektin middleware funktion koodi

Middlewaren käyttö varmistaa sovelluksen tietoturvallisuuden, erityisesti käsiteltäessä henkilökohtaista ja arkaluonteista talousdataa 'dashboard'- ja 'money'-sivuilla. Koodi on kirjoitettu sovelluksen juureen ”middleware.ts” tiedostoon. Next.js projektissa middleware.ts tai middleware.js tiedosto, joka tuo esiin middleware nimisen funktion toimii välikerroksena ilman ylimääräistä määrittelyä. Tässä tapauksessa välikerrosfunktio ajetaan palvelimella aina ennen uuden reitin avaamista.

7 Käyttöliittymä

7.1 Käyttöliittymäkirjasto

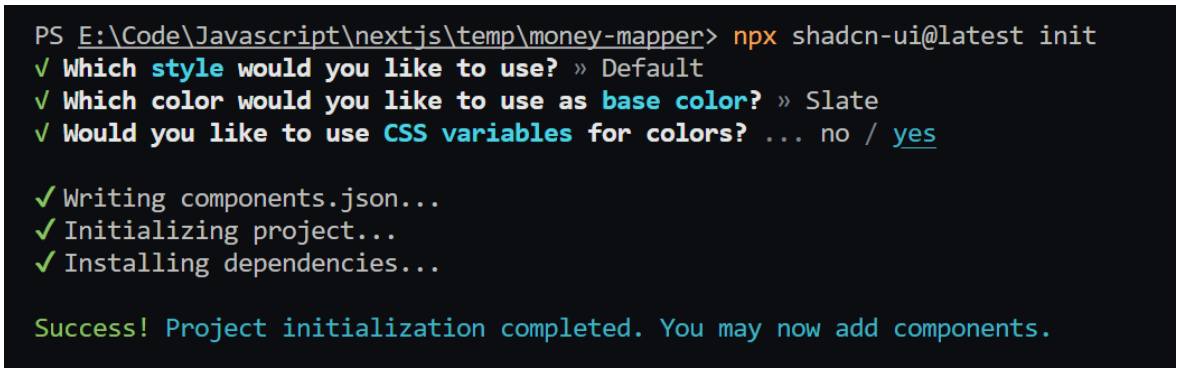
Jokainen web-sovellus tarvitsee jonkinlaisen käyttöliittymän. Käyttöliittymä on tärkeä osa web-sovelluksia, sillä se mahdollistaa käyttäjän vuorovaikutuksen ja antaa visuaalisen ilmeen sovellukselle. Hyvin suunniteltu ja toteutettu käyttöliittymä yleensä parantaa käyttäjäkokemusta.

Sovelluksen käyttöliittymä on rakennettu hyödyntäen shadcn/ui-komponenttikirjastoa ja yksittäiset elementit on tyylitetty TailwindCSS:än avulla. Shadcn/ui on uudentyyppinen käyttöliittymäkirjasto, joka erottuu perinteisistä kirjastoista kuten Material UI ja Chakra UI. Edellä mainitut kirjastot tarjoavat komponenttinsa valmiina esimääriteltyinä paketteina, kun taas Shadcn/ui tarjoaa mahdollisuuden ladata yksittäisten komponenttien lähdekoodin suoraan omaan koodikantaan. (James 2023.)

Kirjaston luoja määrittelee Shadcn/ui:n ennemminkin kokoelmaksi uudelleenkäytettäviä komponentteja kuin varsinaiseksi komponenttikirjastoksi tai käyttöliittymäkehykseksi. Shadcn/ui:n perustana toimivat Tailwind CSS ja Radix UI, ja se tukee tällä hetkellä Next.js, Gatsby, Remix, Astro, Laravel ja Vite -teknologioita. Kirjaston suosio on kasvanut nopeasti, ja se on saanut merkittävää kannatusta web-kehityksen ekosysteemissä. (James 2023.)

Shadcn/ui tarjoaa useita hyödyllisiä ominaisuuksia, kuten teemojen ja teemaeditorin, dark ja lightmode tuen, komentorivikäyttöliittymän (command line interface tai CLI) ja useita valmiita komponentteja, kuten Accordion, Skeleton, Table ja Popover. Lisäksi komponentteja voi tyylitellä Tailwind CSS:än avulla. (James 2023.)

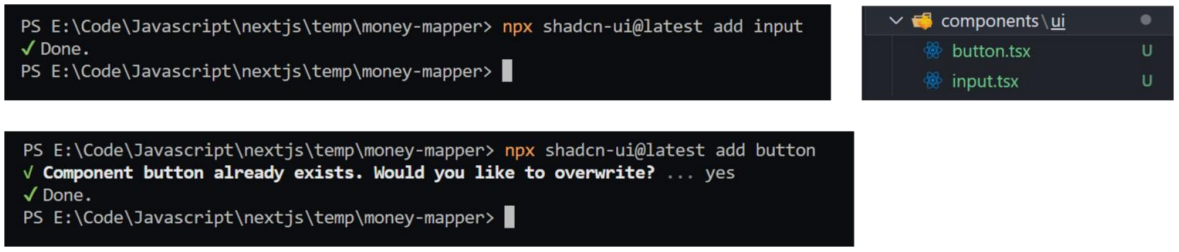
Shadcn-ui:n asentaminen projektiin on suoraviivaista ja se onnistuu npx komennolla, kuten kuvassa 12 esitetään. Asennuksen aikana kysytään muutamia konfiguraatiokysymyksiä, kuten millaisilla tyyleillä alustetaan shadcn/ui komponentit ja käytetäänkö CSS muuttajia komponenttien tyylittelyisessä. Kysymyksiin kannattaa vastata projektivaatimusten mukaisesti, jotta asennus onnistuu ilman yllättäviä ongelmia. (James 2023.)



Kuva 12. Terminaalinäkymä Shadcn-ui:n asentamisesta projektiin

Shadcn-ui:lla voidaan lisätä uusia komponentteja projektiin. Halutut komponentit, kuten painikkeet tai syötekentät, valitaan Shadcn-ui:n dokumentaatiosta ja ne asennetaan käyttämällä nille tarkoitettuja terminaali komentoja. Komento luo uudet tiedostot komponenteille kuten esimerkiksi 'button.tsx' ja 'input.tsx' -kansioon nimeltä '@/components/ui. (Craciun 2023.)

Kuvassa 13 näytetään Shadcn-ui:n button ja input komponenttien asennuskomennot. Jos asennettua komponenttia yrittää asentaa uudelleen sen voi halutessa päivittää. Lisäksi kuvassa on components hakemisto, jonne uudet Shadcn-ui komponentit asennetaan.



Kuva 13. Terminaalinäkymä Shadcn-ui:n komponenttien asennuskomennoista sekä asennettujen komponenttien components hakemisto

Shadcn-ui:n komponentteja voi muokata suoraan asennettujen komponenttien koodissa vastaamaan projektin visuaalisia vaatimuksia. Nämä komponentit voidaan sitten integroida sivuille, esimerkiksi lisäämällä ne 'page.tsx'-tiedostoon. Lisäksi komponenteista voi tehdä uudelleenkäytettäviä variaatioita käyttäen pohjana asennettuja Shadcn-ui komponentteja. Kuvassa 14 esitetään Shadcn-ui:n komponenttien käyttöä projektin koodissa sekä miltä ne näyttävät käyttöliittymässä.

Kuva 14. Shadcn/ui komponenttien käyttö koodissa Tailwind CSS:än kanssa

7.2 Tyylittely

Tämän projektin käyttöliittymän tulisi olla saavutettava, semanttinen, responsiivinen, visuaalisesti moderni ja intuitiivisesti helppokäyttöinen. Lisäksi käyttäjälle tulee ilmoittaa sovelluksen käytön aikana tapahtuvista virheistä ja onnistumisista. Viimeiseksi projektin HTML-koodin tulisi olla mahdollisimman hyvin hakukoneoptimoitu.

Sovelluksen tyylittelyä varten valittiin Tailwind CSS -kehys, joka tarjoaa modernin ja tehokkaan lähestymistavan käyttöliittymän rakentamiseen. Tailwind CSS on suosittu luokkapohjainen CSS-kehys, joka eroaa perinteisistä kehyksistä tarjoamalla kattavan joukon tyyliluokkia suoraan HTML-elementteihin liitettäviksi.

Luokkapohjainen lähestymistapa mahdollistaa hienovaraisen tyylien hallinnan ilman tarvetta kirjoittaa erillistä CSS-koodia. Jokainen luokka edustaa tiettyä CSS-ominaisuutta tai ominaisuuksien yhdistelmää, mikä helpottaa monimutkaisten käyttöliittymäkomponenttien rakentamista luomalla nämä luokat yhteen.

Tailwind CSS:än laajan tyyliluokkavalikoiman ansiosta suurin osa yleisimmistä tyylittelytarpeista voidaan toteuttaa ilman erillistä CSS-koodin kirjoittamista. Nämä tyyliluokat integroituvat suoraan projektissa käytettyyn komponenttikirjastoon. Tämä nopeuttaa kehitysprosessia ja vähentää mahdollisten virheiden riskiä. (Moses 2023.)

Tailwind CSS:n ja Next.js:n yhdistelmä mahdollistaa optimoidun CSS koodin, joka toimitetaan asiakkaan selaimeen. Next.js automaattisesti poistaa käyttämättömät CSS-tyylit rakennusprosessin aikana, mikä tarkoittaa, että lopulliseen sovellukseen sisältyy vain ne tyylit, joita todella käytetään. Tämä lähestymistapa keventää sovellusta ja tekee siitä tehokkaamman. (Moses 2023.)

Lisäksi Tailwind CSS:n käyttö tarjoaa hienojakoisen tason hallintaa tyyleihin poistaen tarpeen kirjoittaa erillistä CSS-koodia useimmissa tyylittelyskenaarioissa. Tämä säästää aikaa ja tehostaa kehitysprosessia. (Moses 2023) Kuvassa 15 on esimerkki Shadcn-ui:n

komponentin tyylittelystä, jossa on käytetty Tailwind CSS:ää. Tailwind CSS tyyliluokat kirjoitetaan className atribuuttiin. Kuvan koodi on selitetty seuraavasti:

- bg-blue asettaa elementin taustavärin siniseksi ja numeerisella arvolla 500 asetetaan sen kirkkautta.
- hover:bg-blue-700 asettaa elementin taustavärin siniseksi, kun sen päälle laittaa kursorin.
- text-white asettaa tekstin valkoiseksi ja font-bold lihavoi tekstin.
- py-2 ja px-2 asettavat padding-arvot Y- ja X-akseleilla. Padding lisää väliä elementin reunoista alkaen.
- rounded pyöristää elementin kulmat.

```
<Button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
  Click me
</Button>
```

Kuva 15. Shadcn/ui:n button elementti tyylitetty Tailwind CSS:ällä

7.3 Saavutettavuus

Saavutettavuus on tärkeä osa modernin verkkosovelluksen kehitystä, ja sen merkitys korostuu entisestään, kun tavoitellaan inklusiivista ja monipuolista käyttäjäkuntaa. Maailmanlaajuisesti noin miljardi ihmistä kohtaa erilaisia fyysisiä tai kognitiivisia haasteita, jotka voivat vaikeuttaa perinteisten verkkopalveluiden käyttöä. Tästä syystä saavutettavuus on tärkeää ja sillä yritetään varmistaa, että kaikki käyttäjät, riippumatta heidän kyvyistään, voivat nauttia ja hyötyä verkkosisällöstä. (Chukwuka 2023.)

Verkkosivuston saavutettavuudella tarkoitetaan sen suunnittelua ja kehittämistä niin, että se on käytettävissä ja ymmärrettävissä kaikille, mukaan lukien ne, joilla on erilaisia fyysisiä, sensorisia tai kognitiivisia rajoituksia. Web Content Accessibility Guidelines (WCAG) määrittelee neljä keskeistä periaatetta saavutettavuudelle: havaittavuus, ohjattavuus, ymmärrettävyys ja kestävyys. (Chukwuka 2023.)

Yksi merkittävä tapa edistää saavutettavuutta on käyttää semanttista HTML:ää. Semantic HTML -elementit kertovat selaimille ja sen apuvälineille kuten ruudunlukijoille sivun rakenteesta ja tarkoituksesta. Tämä on erityisen tärkeää näkö- tai kognitiivisia apuvälineitä käyttäville käyttäjille. Semanttinen HTML parantaa käyttäjän kokemusta ja helpottaa sivuston ymmärrettävyyttä. (Chukwuka 2023.)

Shadcn/ui -komponenttikirjasto, joka on käytössä työn projektissa, tukee saavutettavuutta suoraan asennuksen jälkeen. Se tarjoaa valmiita komponentteja, jotka noudattavat saavutettavuusstandardeja, ja näin ollen varmistavat, että sovellus on käytettävissä kaikille käyttäjille. Esimerkiksi navigointipalkin käyttö, jossa käyttäjän tiedot ja toiminnot ovat näkyvillä, on intuitiivista käyttää ja ne ovat saavutettavissa esimerkiksi pelkästään näppäimistön käytöllä.

Saavutettavuus liittyy myös suoraan hakukoneoptimointiin (SEO). Kun sivusto on suunniteltu käyttäen semanttista HTML-rakennetta ja tarjoaa selkeät tekstivaihtoehdot multimediaelementeille, se parantaa merkittävästi hakukonenäkyvyyttä. Tämä ei ainoastaan auta hakukoneita ymmärtämään sivuston sisältöä paremmin, vaan voi myös lisää sivuston kävijöiden määrää. Kasvava kävijämäärä voi puolestaan edistää liiketoiminnan kasvua, sillä se luo mahdollisuuksia parempaan näkyvyyteen ja potentiaalisten asiakkaiden tavoittamiseen (Chukwuka 2023.)

Kiinnittämällä huomiota saavutettavuuteen ja sitä kautta hyvään hakukoneoptimointiin varmistetaan, että verkkosivusto tavoittaa laajemman yleisön ja tarjoaa positiivisen käyttäjäkokemuksen mahdollisimman monelle kävijälle. Nämä toimenpiteet voivat edistää liiketoiminnan menestystä pitkällä aikavälillä ja samalla tukea projektin menestymistä.

8 Hakukoneoptimointi

Hyvä hakukoneoptimointi varmistaa, että sovellus saavuttaa parhaan mahdollisen näkyvyyden hakukoneissa, houkuttelee kohdeyleisöä paremmin ja tukee käyttäjäkokemusta. Hakukoneoptimoinnilla on siis merkittävä rooli web-kehityksessä.

Next.js tarjoaa vahvan perustan hakukoneoptimoinnille, ja tekee siitä hyvän valinnan websovellusten kehittäjille. Sen sisäänrakennettu tuki palvelinpuolen renderaukselle, staattisen sivun tuottamiselle sekä React server komponenttiominaisuuksille tarjoaa tehokkaat työkalut, joilla parantaa sovelluksen hakukoneystävällisyyttä ja suorituskykyä. (Gary 2023.)

React Server Components (RSCs) mahdollistavat hienovaraisemman lähestymistavan sivujen renderöintiin sekä palvelimella että asiakaspuolella. Tämä tehostaa hakutulossivujen indeksointia, kun pyritään lähettämään vähemmän JavaScriptiä asiakkaalle. RSCt auttavat saavuttamaan paremman hakukonenäkyvyyden päivittämällä dynaamisia tietoja Reactkomponentissa samalla kun staattiset osat sivun sisällöstä pysyvät ennallaan.

Next.js 13:ssa otettiin käyttöön myös streaming UI chunks, joka mahdollistaa JavaScriptvapaan, täysin renderoidun sivun lähettämisen asiakkaalle. Tämä lähestymistapa parantaa verkkosuorituskykyä, koska se sallii ensimmäisen sisällön renderöinnin välittömästi ja sivun lopun, kuten ylimääräisen sisällön tai dynaamisen toiminnallisuuden, siirtämisen myöhemmäksi. Tällainen optimoitu lähestymistapa nopeuttaa sivun latautumista ja parantaa käyttökokemusta. (Gary 2023.)

Hakukoneoptimointia tukee myös päivitetty Next Image Component, joka hyödyntää natiivia HTML img elementin myöhäistä lataamista. Se optimoi kuvat automaattisesti ja tarjoaa paremman tuen SEO:lle, kuten vaaditun alt-tagin oletuksena. Lisäksi hakukoneoptimointia voi parantaa Next.js metadata-komponentin avulla. Next.js Metadata-komponentti korvasi aiemmin käytetyn Head-komponentin, tarjoten tehokkaamman tavan hallita metatageja ja dynaamista metadataa sovelluksen HTML dokumenteissa. (Gary 2023.)

9 Julkaisu

9.1 GitHub

GitHub on keskeinen työkalu ohjelmistokehittäjille projektien hallintaan ja ylläpitoon. Se tarjoaa versionhallinnan palvelun, joka mahdollistaa koodin seurannan, muutosten tekemisen ja tiimityön helpottamisen. Ennen kuin projekti julkaistaan Vercel.com-palvelimille, on tärkeää ymmärtää, miksi koodi on ensin lähetettävä GitHubiin.

Koodin lähettäminen GitHubiin ennen julkaisua tuo useita etuja. GitHub tarjoaa varmuuskopion projektista ja se tarjoaa tehokkaan alustan projektin koodin säilyttämiseen ja jakamiseen. Kaikki projektin versiot tallennetaan, mikä mahdollistaa helpomman paluun aiempiin versioihin, jos tarvetta ilmenee. Tämä tuo luotettavuutta kehitysprosessiin ja auttaa välttämään mahdollisia virheitä. Useamman kehittäjän työskennellessä samassa projektissa, GitHub mahdollistaa muutosten seuraamisen, konfliktien välttämisen ja tehokkaan yhteistyön eri osa-alueiden kesken. (McKenzie 2023.)

Kun projektin lähdekoodi on onnistuneesti lisätty GitHubiin, se on valmis julkaisua varten. Julkaisuvaiheessa on tärkeää varmistaa, että GitHubissa oleva lähdekoodi on ajantasainen ja sisältää kaikki tarvittavat muutokset ja päivitykset. Tämä varmistaa, että julkaistava versio on viimeisin ja se sisältää kaikki kehityksen aikana tehdyt muutokset. Kokonaisuudessaan GitHubin käyttö projektin hallinnassa ja valmistautumisessa julkaisuun tuo tehokkuutta ja selkeyttä kehitysprosessiin.

9.2 Vercel

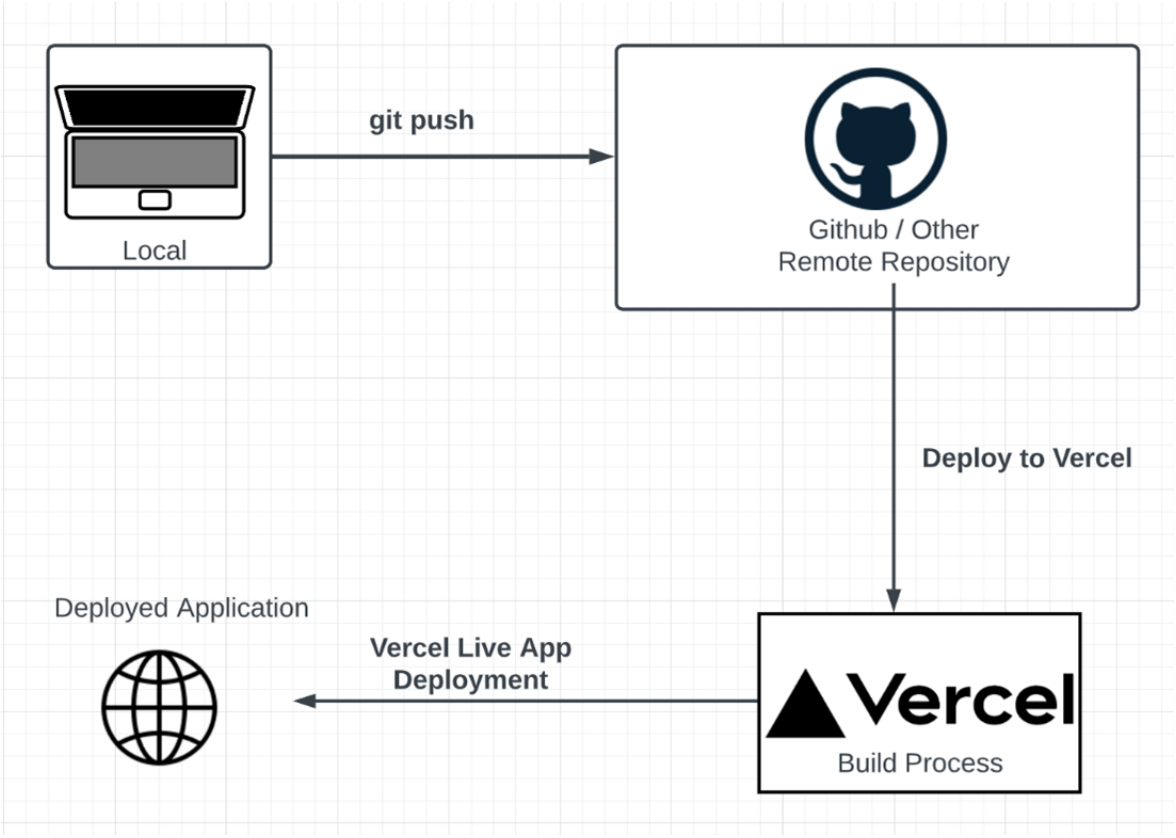
Web-sovellusten kehittäminen on monivaiheinen prosessi, ja yksi keskeisistä vaiheista on sovelluksen julkaisu eli deployaaminen. Tämä vaihe voi olla monimutkainen, mutta Vercelin käyttö helpottaa prosessia. Vercel on alusta, joka tukee kehysratkaisuja ja staattisia sivustoja. Se tarjoaa kehittäjille helpon tavan julkaista ja palvella sovelluksia käyttäen jatkuvan julkaisun- (Continuous Deployment tai CD) ja jatkuvan integraation (Continuous Integration CI) prosesseja.

CI/CD kehityskäytäntö voi nopeuttaa ohjelmiston julkaisuprosessia ja se muodostaa jatkuvan yhteyden sovelluksen lähdekoodiin. Tämä prosessi mahdollistaa ohjelmiston uusien versioiden nopean ja toistuvan käyttöönoton ilman manuaalisia toimenpiteitä. CI/CD:n perusajatuksena on varmistaa, että ohjelmisto on aina valmis tuotantokäyttöön ja että muutokset voidaan ottaa käyttöön mahdollisimman pikaisesti.

Continuous Integration liittyy usein kehittäjien tiiviiseen yhteistyöhön, jossa heidän koodinsa yhdistetään säännöllisesti yhteen keskitettyyn hakemistoon. Tämän myötä voidaan havaita mahdolliset konfliktit ja virheet varhaisessa vaiheessa. (Mitrais 2022.)

Continuous Deployment menee pidemmälle automatisoimalla myös ohjelmiston käyttöönoton tuotantoympäristöön. Kun uusi koodimuutos hyväksytään ja testataan onnistuneesti, se voidaan välittömästi julkaista tuotantoon ilman manuaalisia toimenpiteitä. Tämä nopeuttaa ohjelmiston toimitusprosessia ja vähentää virheitä. (Mitrais 2022.)

Vercelin käyttö Next.js-sovelluksen julkaisemiseen on suoraviivaista. Prosessin aikana annetaan Vercelille lupa sovelluksen koodin julkaisemiseen, ja sen jälkeen Vercel kytkeytyy automaattisesti versionhallintaan, kuten GitHubiin. Tämän jälkeen kehittäjä valitsee projektin, antaa tarvittavat oikeudet ja aloittaa julkaisun. Vercel tarjoaa yksinkertaisen käyttöliittymän, jonka avulla kehittäjät voivat seurata julkaisuvaiheen etenemistä ja ymmärtää helposti mahdolliset virheilmoitukset. (Mitrais 2022) Kuviossa 4 on esitetty projektin julkaisuprosessi Vercel.com-palvelussa.



Kuvio 4. Projektin julkaisu Vercel.com-palvelussa

9.3 Tuotanto

Kun sovellus hyväksyntätestattu se on valmis julkaistavaksi. Julkaistu sovellus on tuotantoversio sovelluksesta. Julkaisutetulle sovellukselle täytyy asettaa oma verkko-osoite. Tässä tapauksessa osoite on Vercel.com:in alidomain. Kuvassa 16 on esitelty tuotantoversion kirjautumis-, Money- ja Dashboardsivut ja miltä ne näyttävät eri laitteilla. Sovellus näyttää visuaalisesti erilaiselta riippuen näytetäänkö se light vai dark tilassa.

Kuva 16. Money Mapper:in visuaalinen ilme eri laitteilla (iPhone, MacBook Pro ja Ipad)

Datan analyysi sovelluksissa on hyvä antaa käyttäjille mahdollisuus ladata kopio omasta datasta. Datan lataaminen mahdollistaa sen jatkoanalysoinnin. Kuvassa 17 näytetään käyttäjän data ladattuna ulos sovelluksesta .xlsx tiedostomuodossa.

	A	B	C	D	E	F	G	H
1	Date	Gross Income YTD	Taxes Paid YTD	Assets Ex Cash	Total Cash	Total Assets	Total Debt	Net Worth
2	31/03/2020	10250.75	2127.80	14923.45	4543.51	19466.96	8003.87	11463.09
3	30/06/2020	20501.50	4255.60	15382.25	4856.62	20238.87	7772.82	12466.05
4	30/09/2020	30752.25	6383.40	15849.13	5181.73	21030.86	7424.18	13606.68
5	31/12/2020	41003.00	8511.20	16324.38	5438.84	21763.22	15538.75	6224.47
6	31/03/2021	11268.90	2351.70	16808.40	5833.95	22642.35	12416.60	10225.75
7	30/06/2021	22537.80	4693.40	18317.49	6097.06	24414.55	11307.60	13106.95
8	30/09/2021	33806.70	7035.10	18841.07	6327.17	25168.24	10008.60	15159.64
9	31/12/2021	45075.60	9376.80	21051.13	6719.28	27770.41	13279.60	14490.81
10	31/03/2022	12506.35	2608.60	21635.23	5833.95	27469.18	12157.60	15311.58
11	30/06/2022	25012.70	5217.20	22233.08	6097.06	28330.14	11326.60	17003.54
12	30/09/2022	37519.05	7825.80	24114.95	6327.17	30442.12	9399.60	21042.52
13	31/12/2022	50025.40	10434.40	25449.16	8416.84	33866.00	10507.60	23358.40
14	31/03/2023	13312.75	2776.60	32859.16	9251.17	42110.33	9572.60	32537.73
15	30/06/2023	26625.50	5553.20	35859.16	9523.28	45382.44	8774.60	36607.84

Kuva 17. Money Mapper:istä ladattu käyttäjän talousdata .xlsx (Excel) muodossa

10 Yhteenveto ja pohdinta

Työn tavoitteena oli kehittää käyttäjäystävällinen ja monipuolinen rahan ja rahoitusten seurantaohjelma, joka hyödyntää moderneja full-stack webkehitystyökaluja. Sovelluksen käyttöliittymä rakentui Shadcn/ui:n komponenttikirjaston avulla, ja Tailwind CSS mahdollisti tehokkaan ja joustavan käyttöliittymän tyylittelyn. Saavutettavuus oli keskeinen näkökulma, ja Shadcn/ui tarjosi valmiita saavutettavia komponentteja, josta oli iso apu kehityksessä.

Web-sovelluksen tyylittelyssä käytettiin Tailwind CSS:ää, mikä tarjosi laajan valikoiman utiliteetteja suoraan HTML-elementteihin. Tämä lähestymistapa vähensi tarvetta erilliselle CSS-koodille ja mahdollisti hienovaraisen tyylivalvonnan. Tailwind CSS:n yhdistäminen Next.js:ään tuotti optimoidun tyylittely toteutuksen parantaen sovelluksen keveyttä ja tehokkuutta.

Hakukoneoptimointi oli olennainen osa kehitysprosessia, ja Next.js tarjosi tehokkaat työkalut tukien server-side renderingiä, static site generationia ja React server components -ominaisuuksia. Continuous Deployment ja Continuous Integration (CD/CD) mahdollisti automatisoidun julkaisuprosessin, ja uudet Next.js 13 ominaisuudet, kuten streaming UI chunks, loivat uudenlaisia mahdollisuuksia verkkosivujen suorituskyvyn parantamiseen.

GitHubin käyttö versionhallinnassa ja Vercelin rooli jatkuvassa julkaisussa antoivat vahvan perustan sovelluksen hallintaan ja julkaisuun. Yritykselle tarjottiin selkeä ja joustava ratkaisu, joka mahdollisti datan käsittelyn ja visualisoinnin muutamalla yksinkertaisella askeleella. Työn aikana käytiin läpi eri vaiheet, kuten koodin kommentointi, projektin rakenne, ja tarvittavat työkalut, mikä vahvisti lopullisen sovelluksen toimivuutta ja käytettävyyttä.

Asiakas on tyytyväinen sovellukseen ja sen jatkokehitys on mahdollista. Jatkokehitetty Money Mapper voisi sisältää käyttäjän talousdatan jakamisen verkkoko-osoitteella tai datan analysointi eri mittareilla. Lisäksi sovellukseen voisi lisätä uusia OAuth tunnistautumisen tarjoajia kuten GitLab, Discord, Facebook (Meta) tai Twitter (X).

Yhteenvetona opinnäytetyö onnistui tavoitteissaan tarjoten kattavan näkymän modernin web-kehityksen työkaluihin ja menetelmiin. Työn tuloksena syntyi vahva pohja, joka yhdistää käytettävyyden, saavutettavuuden ja tehokkuuden, vastaten yrityksen tarpeisiin monipuolisessa websovelluskehityksessä. Ajanhallinnan ja suunnittelun merkitys korostui, ja oppimiskokemus oli arvokas.

Lähteet

Chidera, E. 2023. How to Secure Routes in Next.js 13 – Client-Side, Server-Side, and Middleware-Based Protection. freeCodeCamp. Viitattu 11.11.2023. Saatavissa https://www.freecodecamp.org/news/secure-routes-in-next-js/.

Chukwuka, C. 2023. Next.js and accessibility - best practices for developing inclusive web applications. Bejamas. Viitattu 12.11.2023. Saatavissa https://bejamas.io/blog/next-js-and-accessibility/.

Craciun, D. 2023. Hate writing CSS in Next.js 13? Enter Shadcn-UI. Medium. Viitattu 12.11.2023. Saatavissa https://blog.stackademic.com/hate-writing-css-in-next-js-13-enter-shadcn-ui-a2f0ac16158d.

Dreimanis, G. 2023. Why You Should Choose TypeScript Over JavaScript. Serokell. Viitattu 28.10.2023. Saatavissa https://serokell.io/blog/why-typescript.

Devfaysalkhan. 2023. ESLint : Short introduction to developer. Medium. Viitattu 09.11.2023. Saatavissa https://nextjs.org/docs/pages/api-reference/next-cli.

Enes, S. 2023. Can We Use Next.js 13 as a Backend Framework? Medium. Viitattu 21.10.2023. Saatavissa https://medium.com/codex/can-we-use-next-js-13-as-a-backend-framework-b5f9479a2d.

Fitzgerald, A. 2022. Tailwind CSS: What It Is, Why Use It & Examples. Hubspot. Viitattu 08.11.2023. Saatavissa https://blog.hubspot.com/website/what-is-tailwind-css.

Gary, S. 2023. Next.js SEO for Developers – How to Build Highly Performant Apps with Next. freeCodeCamp. Viitattu 13.11.2023. Saatavissa https://www.freecode-camp.org/news/nextjs-seo/.

Gupta, P. 2023. Next.js 13.4: A Game-Changer for Front-End Development! Medium. Viitattu 28.10.2023. Saatavissa https://medium.com/@prashantg9912/how-nextjs-13-4-will-change-front-end-development-883546209786.

Hanekcud. 2024. What is Frontend? Viitattu 24.02.2024. Saatavissa https://medium.com/@hanekcud/what-is-frontend-4816aedb1f50.

James, N. 2023. Introducing Shadcn UI: A reusable UI component collection. LogRocket. Viitattu 12.11.2023. Saatavissa https://blog.logrocket.com/shadcn-ui-reusable-ui-component-collection/.

Lotanna, N. 2020. Introducing the new Create Next App. LogRocket. Viitattu 08.11.2023. Saatavissa https://blog.logrocket.com/introducing-the-new-create-next-app/.

McKenzie, C. 2023. How to git push an existing project to GitHub. TheServerSide. Viitattu 13.11.2023. Saatavissa https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-push-an-existing-project-to-GitHub.

Mitrais. 2022. How to Deploy Next.js with Vercel. Viitattu 13.11.2023. Saatavissa https://www.mitrais.com/news-updates/how-to-deploy-next-js-with-vercel/.

MongoDB. How to Create a Database. Viitattu 11.11.2023. Saatavissa https://www.mongodb.com/basics/create-database.

Moses, K. 2023. How to Set Up Tailwind CSS with NextJS. freeCodeCamp. Viitattu 12.11.2023. Saatavissa https://www.freecodecamp.org/news/how-to-set-up-tailwind-css-with-next-js/.

NextAuth.js. a. Introduction. Viitattu 11.11.2023. Saatavissa https://next-auth.js.org/getting-started/introduction.

NextAuth.js. b. @auth/prisma-adapter. Viitattu 11.11.2023. Saatavissa https://authjs.dev/reference/adapter/prisma.

NextAuth.js. c. OAuth. Viitattu 11.11.2023. Saatavissa https://next-auth.js.org/configuration/providers/oauth.

Nextjs. a. From React to Next.js. Viitattu 04.11.2023. Saatavissa https://nextjs.org/learn-pages-router/foundations/from-react-to-nextjs/getting-started-with-nextjs.

Nextjs. b. Next.js CLI. Viitattu 08.11.2023. Saatavissa https://nextjs.org/docs/pages/api-reference/next-cli.

Nextjs. c. Build. Viitattu 09.11.2023. Saatavissa https://nextjs.org/docs/app/api-reference/next-cli#build.

Prisma. a. Connect your database (MongoDB). Viitattu 11.11.2023. Saatavissa https://www.prisma.io/docs/getting-started/setup-prisma/start-from-scratch/mongodb/connect-your-database-node-mongodb.

Prisma. b. Query engine. Viitattu 11.11.2023. Saatavissa https://www.prisma.io/docs/concepts/components/prisma-engines/query-engine#the-query-engine-at-runtime.

Sajdok, N. 2023. SSR vs SSG in Next.js – a practical overview for CTOs and devs. The Software House. Viitattu 04.11.2023. Saatavissa https://tsh.io/blog/ssr-vs-ssg-in-nextjs/.

Shankar, B. 2023. Getting started with Prisma ORM. Medium. Viitattu 08.11.2023. Saatavissa https://medium.com/@binayakgourishankar/getting-started-with-prisma-orm-89c2fcda5026.

Srivastava, S. 2022. Why mongoDB ?? Knoldus. Viitattu 08.11.2023. Saatavissa https://blog.knoldus.com/why-mongodb/.

TypeScript. TypeScript for the New Programmer. TypeScript. Viitattu 28.10.2023. Saatavissa https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

Welearncode. A Complete Beginner's Guide to Next.js. Viitattu 04.11.2023. Saatavissa https://welearncode.com/beginners-guide-nextjs/.

W3Schools. React Components. Viitattu 04.11.2023. Saatavissa https://www.w3schools.com/react/react_components.asp.