Työpaikkailmoituspalvelu Atkins ry:lle

Jani Mäenpää

Haaga-Helia ammattikorkeakoulu
Tradenomi
Opinnäytetyö
2022

Tiivistelmä

Tekijä(t)
Jani Mäenpää

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Työpaikkailmoituspalvelu Atkins ry:lle

Sivu- ja liitesivumäärä
31

Toiminnallisen opinnäytetyön tavoite oli luoda Haaga-Helian tietojenkäsittelyn opiskelijoiden ainejärjestö Atkins ry:lle uusi automatisaatiota lisäävä työpaikkailmoituspalvelu. Atkins ry:n vanhan työpaikkailmoituspalvelun ongelmana oli, että ilmoituksen saaminen näkyville vaati paljon manuaalista työtä Atkins ry:n työntekijöiltä. Tämän toiminnallisen opinnäytetyön tarkoitus oli ratkaista kyseinen ongelma.

Opinnäytetyön tietoperusta jakaantuu kolmeen lukuun. Ensimmäinen teorialuvuista kertoo ohjelmistokehityksen elinkaarimallista ja valitsemastani vesiputousmallisesta ohjelmistotuotantoprosessista. Toinen teorialuku syventyy projektin pääteknologiaan, React-kirjastoon perustuvaan, Next.js-sovelluskehykseen. Tässä yhteydessä vertaillaan muun muassa erilaisia web-palvelun renderointitapoja. Kolmannessa teorialuvussa käydään lopuksi läpi muita tärkeitä projektin web-teknologioita. Esimerkiksi TypeScriptin ja Tailwind CSS:n käyttöönoton etuja web-kehityksessä esitellään.

Opinnäytetyön empiirisessä osassa käydään projektin työvaiheet ja tulokset läpi tietoperustassa esitetyn vesiputousmallin mukaisesti. Lopuksi pohdintaosuudessa mietitään teknologiavalintojen onnistumista, työpaikkapalvelun jatkokehitysideoita sekä arvioidaan opinnäytetyöprosessista saatuja kokemuksia.

Asiasanat
Next.js, sovelluskehys, web-kehitys, ohjelmistokehitys

Sisällys

1 Johdanto

Tämä on toiminnallinen opinnäytetyö, joka toteutetaan Haaga-Helian tietojenkäsittelyn ainejärjestö Atkins ry:lle. Atkins ry tarjoaa tietojenkäsittelyn opiskelijoille esimerkiksi tapahtumia, edunvalvontaa ja mahdollisuuksia työllistyä laajan yritysyhteistyön avulla, johon kuuluu muun muassa erilaiset yritysvierailut ja työpaikkailmoitusten jakaminen.

IT-alalla on suuri tarve uusille työntekijöille, jonka takia IT-alan yritykset haluavat tehdä yhteistyötä alan ainejärjestöjen kanssa ja jopa maksaa siitä. Atkins ry saa viikoittain useita työpaikkailmoituksia, jotka laitetaan Atkins ry:n verkkosivuille. Jokainen työpaikkailmoitus vaatii, että joku Atkins ry:n hallituksen jäsen syöttää manuaalisesti työpaikkailmoituksen WordPress-alustan kautta työpaikkasivulle. Tämä on mekaaninen ja aikaa vievää prosessi, mistä syntyi idea tälle toimeksiannolle.

Opinnäytetyön tavoitteena on luoda Atkins ry:lle työpaikkailmoituspalvelu, jossa työpaikkailmoitusten prosessointi on automatisoitu mahdollisimman pitkälle. Atkins ry:n opiskelijajäsenet voivat selailla työpaikkailmoituksia ja eri yritysten edustajat voivat lisätä työpaikkailmoituksia palveluun. Yritysten ilmoitukset tulevat hyväksyttäväksi hallintapaneeliin, jossa Atkins ry:n edustajat voivat nappia painamalla hyväksyä, hylätä tai muokata ilmoituksia. Jos ilmoitus hyväksytään, se näkyy heti muiden ilmoitusten seassa.

1.1 Käytettävät teknologiat

Palvelun ratkaisu toteutetaan suurimmaksi osaksi TypeScriptilla, joka on JavaScriptin pohjalta kehitetty ohjelmointikieli ja joka hyödyntää JavaScriptin ekosysteemiä. TypeScriptin etu on, että se käyttää vahvaa tyypitystä JavaScriptin dynaamisen tyypityksen sijasta. TypeScript on hyvin käytetty moderneissa web-pohjaisissa käyttöliittymäprojekteissa. Käytän työssäni web-kehittäjänä paljon JavaScriptia käyttöliittymäkerrosten rakentamiseen, jonka takia valitsin tähän opinnäytetyön TypeScriptin, koska haluan kehittyä kielessä enemmän ja en pääse töissäni sitä muuten oikeastaan käyttämään.

Hyödynnän JavaScriptin ekosysteemiä tässä käyttämällä JavaScript-pohjaista React-kirjastoa. Lisäksi käytän Reactiin pohjautuvaa Next.js-sovelluskehystä. Next.js-sovelluskehys tarjoaa valmiita työkaluja palvelun front-end- ja back-end-ratkaisujen rakentamiseen, mikä tekee sovelluksen kehityksestä nopeaa.

Tietokannaksi valitsin PostgreSQL-tietokannan, koska olen ollut sen kanssa aikaisemminkin tekemisissä ja sen käyttö tuntuu luontevalta. Tietokannan hallintaan käytän Prisma-nimistä objekti-relaatiokartoituskirjastoa, jolla datan lukeminen, lisääminen, muokkaaminen ja poistaminen sujuu yksinkertaisesti.

Käyttöliittymän tyylisovelluskehyksenä toimii Tailwind CSS. Halusin oppia käyttämään Tailwind CSS -kehystä, koska se on ollut suosittu kehittäjien keskuudessa ja halusin tietää mistä on kyse, minkä takia valitsin sen teknologiaksi. Lopullinen ratkaisu tullaan lopuksi julkaisemaan jossain pilvipalvelussa, kuten DigitalOcean Droplettina.

1.2 Käsitteet

Tässä kappaleessa käydään lyhyesti läpi tässä opinnäytetyössä käytettäviä käsitteitä.

API	API:n (application program interface) eli ohjelmointirajapinnan avulla kaksi eri ohjelmaa voivat jakaa tietoa keskenään.
Backend	Web-sovelluksen osa, joka vastaa käyttäjälle näkymättömästä sovelluksen bisneslogiikasta.
Fetch-rajapinta	JavaScript-ohjelmointirajapinta, jonka avulla voidaan tehdä http-kutsuja eri kohteisiin.
Frontend	Web-sovelluksen osa, joka vastaa sovelluksen käyttöliittymästä. .
GraphQL	Avoimen lähdekoodin tietojen kysely- ja käsittelykieli ohjelmointirajapinnoille.
Heroku	Pilvipalvelu, jossa voidaan isännöidä erilaisia sovelluksia.
JavaScript	Ohjelmointikieli, jolla pystyy kehittämään niin selaimessa kuin palvelimella toimivia sovelluksia.
Next.js	React-pohjainen avoimen lähdekoodin sovelluskehys, jolla pystyy rakentamaan kokonaisia web-sovelluksia erilaisia renderöintimenetelmiä käyttäen.
Node.js	Node.js on asynkroninen tapahtumapohjainen JavaScript-ajonaikainen (runtime) ympäristö, jonka avulla voidaan rakentaa muun muassa skaalautuvia verkkosovelluksia
ORM	Ohjelmointitekniikka, jolla voidaan kutsua ja manipuloida dataa tietokannasta käyttäen olio-ohjelmointia tukevaa ohjelmointikieltä.
PostgreSQL	Avoimen lähdekoodin tietokannan hallintajärjestelmä.

Prisma	Uuden sukupolven Node.js ja TypeScript ORM.
REST	Ohjelmointirajapintojen toteuttamiseen tarkoitettu arkkitehtuurimalli.
React	JavaScript-pohjainen kirjasto, jolla voidaan rakentaa deklaratiivisia käyttöliittymiä komponenttipohjaisesti.
Renderointi	Prosessi, joka muuttaa web-sovelluksen koodin käyttäjälle näkyväksi sivuksi.
Tailwind CSS	CSS-kehys, jolla voidaan rakentaa HTML-dokumentin tyylit HTML:n sisällä käyttäen class-attribuuttia.
TypeScript	Ohjelmointikieli, joka lisää JavaScriptiin vahvan tyypityksen.
WordPress	Avoimen lähdekoodin PHP-ohjelmointikielellä tehty sisällönhallintaohjelmisto, joka käyttää MySQL-tietokantaa tiedon tallentamiseen.

2 Ohjelmistokehityksen elinkaari ja vesiputousmalli

Ohjelmistokehityksen saralla on kehitetty monenlaisia eri malleja, joiden avulla kehittää ohjelmistoja. Nykyään suosituimmat käytössä olevat mallit ovat ketterää kehitystä, joissa tuotteen omistaja voi muuttaa vaatimuksia kesken projektin. Tässä projektissa päätin ottaa käyttöön niin sanotun vesiputousmallin, koska projektin vaatimukset eivät oikeastaan tule muuttumaan kesken projektin.

Ohjelmistosuunnittelussa ja tuotekehityksessä vesiputousmalli on lineaarinen, peräkkäinen lähestymistapa ohjelmistokehityksen elinkaareen (SDLC). Vesiputousmallissa korostetaan loogista askelkulkua. Jokainen kehitysvaihe sisältää selkeät päätepisteet tai tavoitteet, joihin ei voi palata valmistumisen jälkeen, samalla tavalla kuin vesi virtaa kallion reunan yli. Tohtori Winston W. Royce loi termin vuonna 1970 julkaistussa tutkimuksessa, ja se on edelleenkin käytössä teollisuudessa. (Lewis 2019.)

Vesiputousmallissa voi olla viidestä seitsemään eri vaihetta, riippuen projektin monimutkaisuuden asteesta. Tässä tapauksessa kuusi projektissa käytettävää vaihetta ovat:

- Vaatimusmäärittely
- Suunnittelu
- Toteutus
- Testaus
- Julkaisu
- Ylläpito

Näiden 6 vaiheen pitäisi kattaa koko luotavan projektin elinkaari.

Vesiputouskehityksen edut ovat, että se mahdollistaa projektin osittamisen ja hallinnan. Kullekin kehitysvaiheelle voidaan asettaa aikataulu ja määräajat ja tuote voi edetä kehitysprosessin mallivaiheet yksitellen. Vesiputouskehityksen haittana on, että se ei salli paljon pohdintaa tai tarkistamista. Kun sovellus on testausvaiheessa, on erittäin vaikeaa palata takaisin ja muuttaa jotain, jota ei ollut hyvin dokumentoitu tai ajateltu suunnitteluvaiheessa. (Tutorials Point s.a.)

2.1 Vaatimusmäärittely

Vesiputousmalli vaatii runsaasti ennakkoon tehtyä dokumentaatiota. Ensimmäiseksi aloitetaan keräämällä tietoja asiakkailta tai loppukäyttäjiltä heidän tarpeistaan ja tuloksista, joita he odottavat ohjelmistolta tai tuotteelta. Näistä tarpeista ja tuloksista syntyy dokumentti, jossa listataan mitä projektilta vaaditaan. (Waseem 2022.)

Vaatimusmäärittely on suunnitteluvaihe ja viimeinen tilaisuus olla yhteydessä asiakkaisiin ennen projektin alkamista. Haluat kirjoittaa muistiin mahdollisimman paljon tietoa varmistaaksesi, että olet oikealla reitillä ja että kaikki on suunniteltu. (Waseem 2022.)

Vaatimusmäärittelyvaihe on kriittinen, koska se luo pohjan seuraaville vaiheille. Tästä syystä vaatimusten keräämiseen on käytettävä paljon aikaa. Vaatimusmäärittelyssä pitäisi pystyä kuvailemaan jokaista seuraavaa vaihetta yksityiskohtaisesti ensimmäisen vaiheen loppuun mennessä. Tämä sisältää projektin aikataulut, hinnat, riskit, oletukset ja riippuvuudet. (Waseem 2022.)

2.2 Suunnittelu

Toisessa vaiheessa, eli suunnitteluvaiheessa, tarkastellaan vaatimusmäärittelyssä luotuja vaatimuksia ja kehitetään suunnitelma niiden täyttämiseksi. Tässä vaiheessa päätetään se polku, jota kuljetaan, jotta ratkaisu ja asiaankuuluvat tekniset tiedot saadaan toimitettua. (Waseem 2022.)

Suunnitteluvaihe jakautuu usein kahteen osaan: loogiseen suunnitteluvaiheeseen ja fyysiseen suunnitteluvaiheeseen. Loogisen suunnittelun alivaiheessa mietitään kaikki teoreettiset ratkaisut, joilla on potentiaalia saavuttaa asiakkaan tavoitteet. (Waseem 2022.)

Fyysinen suunnittelu, edellyttää konkreettisempia eritelmiä. Tässä alivaiheessa määritetään laitteistot, ohjelmistot, arkkitehtuuri, tietolähteet ja palvelut, joita käytetään projektin aikana. Tässä vaiheessa ei tehdä kuitenkaan vielä ohjelmointia. (Waseem 2022.)

2.3 Toteutus

Toteutusvaiheessa alkaa projektin ohjelmointi. Tässä vaiheessa projektitiimi toteuttaa aikaisemmissa vaiheissa suunnitellut vaatimukset. Koska suunnitteluvaiheeseen on käytetty paljon aikaa ja toteutettavat ominaisuudet löytyvät vaatimusmäärittelydokumentista, on tämä vaihe mahdollisesti kohtalaisen lyhyt. (Waseem 2022.)

Suuret ohjelmistot jaetaan pienempiin ohjelmiin monimutkaisemmissa projekteissa. Tiimit käyttävät yksikkötestausta, jossa rakennetaan ja testataan yksi yksikkö kerrallaan ennen niiden yhdistämistä lopputulokseen. (Waseem 2022.)

2.4 Testaus

Testausvaiheessa varmistetaan, että kaikki vaatimukset täyttyvät ja että tuote ei vaadi vianmääritystä. Ennen kuin tuote saavuttaa asiakkaan, laadunvarmistustiimi tarkistaa toimitettavan tuotteen huolellisesti. Hanke palautetaan ensimmäiseen vaiheeseen, jos siinä havaitaan vakavia puutteita tai vaatimuksia ei ole täytetty. Pienet virheet voidaan korjata toistamalla suunnitteluvaihe uudelleen. (Waseem 2022.)

2.5 Julkaisu

Lopullinen käyttöönottoprosessi alkaa, kun ohjelmiston testausvaihe on suoritettu ja järjestelmässä ei havaita vikoja. Lopullinen ohjelma julkaistaan ja tarkistetaan mahdollisten käyttöönotto-ongelmien varalta projektipäällikön palautteen perusteella. (Martin 2022.)

2.6 Ylläpito

Ylläpitovaihe alkaa, kun tuote on julkaistu ja käyttäjät alkavat käyttää sitä. Jotkut puutteet voidaan havaita vasta, kun asiakas on ottanut tuotteen käyttöön. Ylläpitoa varten tarvitaan ylläpitotiimi, joka pystyy käsittelemään kaikki kehittyvät viat tai rikkinäiset ominaisuudet. Tämä vaihe päättyy, kun asiakas on täysin tyytyväinen, tai se voi jatkua, jos asiakas tarvitsee usein päivityksiä. (Waseem 2022.)

3 Next.js

Next.js on Vercelin kehittämä joustava React-kirjastoon pohjautuva sovelluskehys, jonka avulla voidaan luoda nopeita verkkoapplikaatioita. Next.js-sovelluskehyksellä voidaan luoda applikaation käyttöliittymä (front-end) Reactia hyödyntäen ja sovelluksen tiedonsiirtokerros (back-end) hyödyntäen Node.js-ajonaikaista ympäristöä. Next.js-sovelluskehyksellä kehitettävää applikaatiota ohjelmoidaan JavaScript-ohjelmointikielellä. Lisäksi Next.js-sovelluskehyksessä on tuki vahvasti tyypitetylle TypeScript-ohjelmointikielelle. (Next.js s.a. a.)

Next.js tarjoaa kehittäjälle paljon valmiita työkaluja, mitä React ei itsessään tarjoa. Perinteisessä React-applikaatiossa applikaation sisältö renderoituu asiakaspuolella (client-side rendering). Tämän ongelma on, että ennen kuin sisältö tulee näkyväksi käyttäjälle, täytyy JavaScriptin latautua ja sovelluksen päättää mitä käyttäjälle näytetään. Lisäksi applikaation hakukonenäkyvyys on huonompi, koska hakukoneet osaavat indeksoida staattisia sivuja paremmin. Next.js ratkaisee tämän renderoimalla sisällön etukäteen palvelimen puolella (server-side rendering), jolloin sivu latautuu nopeammin ja hakukoneet pystyvät indeksoimaan sivun tarkemmin. (FreeCodeCamp 2019.)

Muita FreeCodeCampin mainitsemia Next.js:n pääominaisuuksia ovat muun muassa:

- Hot Code Reloading: Next.js uudelleenlataa sivun, kun se huomaa muutoksen kovalevyllä
- Automaattinen reititys: Kaikki URL-osoitteet yhdistetään tiedostojärjestelmään, jos ne ovat pages-kansiossa, eikä tätä varten tarvitse ylimääräistä konfigurointia.
- Yhden tiedoston komponentit: Komponentin tyylit voivat olla samassa tiedostossa, jolloin muodostuu vain yksi komponenttitiedosto.
- Ekosysteemin yhteensopivuus: Next.js toimii hyvin JavaScript-, Node- ja React-ekosysteemejen kanssa.
- Automaattinen koodinjako: Sivut renderoidaan juuri niiden tarvitsemilla kirjastoilla ja JavaScriptilla. Sen sijaan, että luotaisiin yksi JavaScript-tiedosto, joka sisältää kaiken sovelluskoodin, Next.js hajottaa sovelluksen automaattisesti useihin eri resursseihin.
- Dynaamiset komponentit: JavaScript-moduulit ja React-komponentit voidaan tuoda dynaamisesti.

3.1 Renderointitavan valinta

Renderointi web-kehityksen kontekstissa tarkoittaa prosessia missä web-sivun koodi, eli yleensä HTML, CSS ja JavaScript, muutetaan käyttäjälle näkyviksi interaktiivisiksi sivuiksi. Tämän prosessin suorittaa selaimen oma renderointimoottori (rendering engine). Renderointimoottori vastaa web-sivun sisällön hakemisesta ja sivun maalaamisesta käyttäjälle. (Seobility Wiki s.a.)

Next.js tarjoaa kolme eri tapaa renderoida sovellus. Nämä vaihtoehdot ovat staattinen renderointi (static rendering), palvelinpuolen renderointi (server-side rendering) ja asiakaspuolen renderointi (client-side rendering). Next.js antaa mahdollisuuden joko käyttää pelkästään yhtä renderointitapaa tai sitten yhdistellä eri tapoja samassa sovelluksessa.

3.1.1 Staattinen renderointi

Staattisessa renderoinnissa (static rendering) sivun renderointi tapahtuu sovelluksen rakennusaikana (build-time). Tämän ansiosta sivun piirtyminen käyttäjälle on nopeaa ja käyttäjä pääse nopeasti vuorovaikutukseen sivun kanssa, mikäli JavaScriptin määrä on maltillinen. (web.dev 2019.)

Staattista renderointia kannattaa käyttää aina, jos mahdollista, koska se on käyttäjälle kaikkein nopeinta, koska sivu on valmiiksi rakennettu ja se voidaan laittaa välimuistiin. Staattista renderointia ei kuitenkaan välttämättä kannata käyttää, jos eri sivuja on todella paljon tai jos data sivulla muuttuu usein. (Next.js s.a. b.)

```
import React from "react";

const PostList = ({ posts }) => {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
};

export async function getStaticProps() {
  const response = await fetch("https://example.api/api/posts");
  const posts = response.json();

  return {
    props: { posts: posts },
  };
}

export default PostList;

```

Kuva 1. Esimerkki Next.js-komponentti, jossa käytetty getStaticProps-funktiota

Next.js tekee sivuista automaattisesti staattisia, jos muuta ei ole määritelty. Next.js:n staattisiin sivuihin voidaan hakea dataa esimerkiksi REST-ohjelmointirajapintaa käyttäen. Kuvassa 1 on esimerkki, miten haettu data saadaan välitettyä komponentille. Määrittämällä funktion *”export async function getStaticProps()”,* Next.js ymmärtää, että kyseinen komponentti halutaan renderoida staattisesti. Funktion sisällä voidaan hakea data REST-ohjelmointirajapinnalta käyttäen esimerkiksi fetch-ohjelmointirajapintaa. Kun haluttu data on noudettu, palautetaan se oliona funktion lopussa propsina, jolloin komponentti voi käyttää kyseistä dataa.

3.1.2 Palvelinpuolen renderointi

Palvelinpuolen renderoinnissa (server-side rendering) palvelin generoi koko web-sivun HTML:n käyttäjän pyytäessä sivua. Tällöin vältytään ylimääräisiltä edestakaisilta operaatioilta, koska datan noutaminen ja HTML-mallintaminen tehdään palvelinpuolella valmiiksi, ennen kuin käyttäjän selain saa vastauksen. Tämä nopeuttaa sivun piirtymistä käyttäjän selaimeen, koska selaimen ei tarvitse tehdä raskaita JavaScript-operaatioita, sillä JavaScript-koodia ei tule käyttäjälle niin paljoa. (web.dev 2019).

Palvelinpuolen renderointia kannattaa käyttää, kun sivun data muuttuu useasti tai sivua kutsuttaessa halutaan antaa pyynnölle esimerkiksi tunnistautumiseen liittyviä http-otsikoita. (Next.js s.a. c.) Palvelinpuolen renderointi on hyvä hakukonenäkyvyydelle, koska hakukonerobotit pystyvät indeksoimaan sivuja parhaiten, kun sisältö on näkyvissä renderoidussa HTML:ssa. (Google Search Central s.a.)

```
1   import React from "react";
2
3   const PostList = ({ posts }) => {
4     return (
5       <ul>
6         {posts.map((post) => (
7           <li key={post.id}>{post.title}</li>
8         ))}
9       </ul>
10    );
11  };
12
13  export async function getServerSideProps() {
14    const response = await fetch("https://example.api/api/posts");
15    const posts = response.json();
16
17    return {
18      props: { posts: posts },
19    };
20  }
21
22  export default PostList;
23
```

Kuva 2. Esimerkki Next.js-komponentti, jossa käytetty getServerSideProps-funktiota

Next.js-komponentin saa renderoitumaan palvelimella määrittämällä komponenttiin kuvassa 2 näkyvän *”export async function getServerSideProps*()”-funktion. Komponentille datan syöttämisen logiikka on sama kuin staattisessa renderoinnisssa, eli noudetaan data esimerkiksi fetch-ohjelmointirajapintaa käyttäen ja syötetään rajapinnan saama vastaus JSON-oliona propseihin. Nyt dataa voidaan käyttää PostList-komponentissa.

3.1.3 Asiakaspuolen renderointi

Asiakaspuolen renderointi tarkoittaa sivun renderoimista suoraan käyttäjän selaimessa käyttäen JavaScriptia. Tämä tarkoittaa käytännössä siis sitä, että kaikki sovelluksen logiikka, datan noutaminen ja mallintaminen tapahtuu käyttäjän puolella selaimessa, eikä sitä tehdä etukäteen palvelimella. (web.dev 2019.)

Asiakaspuolen renderoinnin ongelmia on, että kun sovelluksen koko kasvaa, kasvaa myös JavaScriptin määrä, mikä hidastaa sovellusta, koska JavaScript pitää prosessoida, ennen kuin sivun sisältö voidaan näyttää käyttäjälle. Koodin jakamisella (code splitting) tätä seurausta voidaan kuitenkin lieventää, koska käyttäjälle annetaan vain osa koodista kerralla käyttöön. (web.dev 2019.)

```jsx
import React, { useEffect, useState } from "react";

const PostList = (props) => {
  const [posts, setPosts] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    fetch("https://example.api/api/posts")
      .then((response) => response.json())
      .then((data) => {
        setPosts(data);
        setIsLoading(false);
      });
  }, []);

  if (isLoading) return <div>Loading...</div>;
  if (!posts) return <div>No posts found...</div>;

  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
};

export default PostList;
```

Kuva 3. Esimerkki miten toteuttaa asiakaspuolen renderointi Next.js-komponentissa (Next.js s.a. d)

Kuvassa 3 voidaan nähdä, että data haetaan nyt komponentin sisällä useEffect-hookia käyttämällä. Tämä käytännössä tarkoittaa, että sivu renderoituu ensin ja vasta sitten komponentti alkaa noutamaan dataa. Tämän takia tarvitaan erilaisia tiloja, kuten kuvassa näkyvä ”posts” ja ”isLoading”. Dataa ei ole sivun renderoinnin alussa olemassa, joten tarvitsemme ”isLoading”-tilaa, jotta

voimme näyttää latausnäkymää datan noudon aikana. Käytännössä prosessi on seuraavanlainen:

1. Näytetään latausnäkymä
2. Noudetaan data
3. Jos dataa ei löydy, näytetään virhenäkymä
4. Jos data löytyi, näytetään löydetty data

3.2 Sidosteknologiat

3.2.1 React

React on Facebookin kehittämä JavaScript-pohjainen kirjasto deklaratiivisten käyttöliittymien rakentamiseen. Reactin avulla voidaan luoda monimutkaisia käyttöliittymiä pienistä ja eristetyistä komponenteiksi kutsutun koodin osista. (React s.a.)

```
1   import React from "react";
2
3   const HelloWorld = () => {
4     const text = "Hello World!";
5
6     return (
7       <div>
8         <h2>{text}</h2>
9         <p>This is a paragraph.</p>
10      </div>
11    );
12  };
13
14  export default HelloWorld;
15
```

Kuva 4. Esimerkki React-komponentista

Kuvassa 4 voidaan nähdä esimerkki HelloWorld-nimisestä React-komponentista. Komponentti on muuten normaalia JavaScriptia, mutta siinä on muutama Reactille tyypillinen erikoisuus. Rivillä 1 React tuodaan (import) käyttöön. Tätä ei kuitenkaan tarvitse tehdä enää erikseen uusimmassa React-versiossa (18 ja suurempi). Rivillä 3 komponentti määritetään nuolifunktiolla, joka on tyypillinen tapa tehdä nykyaikaista JavaScriptia *”function”*-avainsanan sijasta. Komponentin funktion sisällä on määritelty muuttuja *”text”*.

Mielenkiintoisin asia tässä komponentissa Reactia ajatellen on se, mitä funktio palauttaa. React-komponenteissa palautetaan niin sanottua JSX-syntaksia (JavaScript Syntax Extension), joka on kuin HTML-merkintäkieltä, mutta siinä voidaan käyttää JavaScriptia aaltosulkeiden sisällä, samaan

tapaan kuin kuvan 4 komponentissa rivillä 8. Kun komponentti renderoidaan tulostuu *”{text}”*-syntaksin sijasta ”Hello World!”.

3.2.2 JavaScript

JavaScript on komento- tai ohjelmointikieli, jolla voidaan rakentaa web-sivuille monimutkaisia ominaisuuksia, jota perinteiset staattiset HTML-sivut eivät tue. JavaScriptin avulla voidaan esimerkiksi dynaamisesti päivittää verkkosivun sisältöä, hallinnoida multimediaa tai animoida kuvia. JavaScript, HTML-merkintäkieli ja CSS-tyylikieli muodostavat tasot, jonka avulla voidaan rakentaa nykyaikaisia verkkosivuja. (MDN Web Docs 2022.)

JavaScriptissä on dynaaminen tyypitys, eli siinä ei suoraan määritellä tyyppejä, vaan kieli yrittää tehdä tyyppimuutoksia automaattisesti parhaansa mukaan. Tämän takia JavaScriptissä on vahvasti tyypitettyjen ohjelmointikielien näkökulmasta erikoisuuksia.

```
1
2   const number1 = 5;
3   const number2 = "5";
4
5   function calculateSum(a, b) {
6     return a + b;
7   }
8
9   const sum = calculateSum(number1, number2)
10
11  console.log(sum)
12
```

Kuva 5. JavaScript-esimerkki tyyppivirheestä

Kuvassa 5 on JavaScript-esimerkki, jossa on määritetty riviltä 5 alkaen *”calculateSum”*-niminen funktio, jonka konstruktori ottaa vastaan luvut a ja b. Rivillä 9 on määritelty *”sum”*-muuttuja, joka kutsuu kyseisen funktion muuttujilla ”*number1”* ja ”*number2”*. *”number2”*-muuttuja on kuitenkin merkkijono, eikä luku, joten rivi 11 tulostaa 55 vaikka haluttaisiin tulokseksi 10. Tätä virhettä ei huomata välttämättä vasta kuin tuotannossa. koska se ei ole JavaScriptin mielestä virhe.

3.2.3 TypeScript

TypeScript on saavuttanut suurta suosiota viime aikoina kehittäjien keskuudessa niin työelämässä kuin harrastusmielessäkin. TypeScript on ohjelmointikieli, joka rakentuu JavaScriptin päälle. TypeScript lisää JavaScriptiin ylimääräistä syntaksia sallien tyypityksen. Kun TypeScript-koodi on kirjoitettu, se syötetään TypeScript-kääntäjälle (compiler), joka kääntää TypeScript-koodin JavaScript-tyyppiseksi koodiksi. TypeScript-tiedostot ovat ”.ts”-päätteisiä, kun JavaScript-tiedostot ovat ”.js”-päätteisiä. (TypeScript Tutorial s.a.)

TypeScriptin tarkoitus on parantaa kehittäjän tuottavuutta vähentämällä koodissa esiintyviä bugeja. Jos TypeScript-ohjelmassa esimerkiksi funktion tai muuttujan tyyppi on määritelty, ja kun sille yritetään sijoittaa väärän tyyppistä tietoa, TypeScript-kääntäjä varoittaa tästä heti, eikä suostu kääntämään tätä koodia. Näin vältytään ongelmalta, joka JavaScriptissä voitaisiin huomata vasta koodin ajoaikana (runtime). (TypeScript Tutorial s.a.)

```
1
2   const number1 = 5;
3   const number2 = "5";
4
5   function calculateSum(a: number, b: number) {
6     return a + b;
7   }
8
9   const sum = calculateSum(number1, number2);
10
11  console.log(sum);
12
```

Kuva 6. TypeScript-esimerkki tyyppivirheestä

Kuvassa 6 on riviltä 5 määritelty *”calculateSum”*-niminen funktio, joka ottaa konstruktoriin kaksi muuttujaa *”a”* ja *”b”*, joiden molempien tyypiksi on asetettu luku. Nyt kun rivillä 9 yritetään kutsua kyseistä funktiota, mutta *”number2”*-muuttuja on tässä tapauksessa väärää tyyppiä, varoittaa TypeScript tästä, jolloin virhe voidaan heti huomata ja korjata, täten säästäen kallisarvoista kehittäjän aikaa.

3.2.4 Npm-paketinhallinta

Npm-paketinhallinta on tietovarasto, jossa voidaan julkaista avoimen lähdekoodin Node.js-projekteja. Näitä projekteja tietovarastossa kutsutaan paketeiksi. (Node.js 2011) Yksi esimerkki tämmöisestä projektista on Next.js. Lisäksi Npm tarjoaa komentoliittymätyökalun, jolla sen tietovarastossa olevia projekteja voi asentaa. Se tarjoaa myös apuja paketin version- ja riippuvuuksienhallintaan. (Node.js 2011.)

Jotta npm-paketinhallintaa voidaan käyttää, pitää projektissa olla *”package.json”*-tiedosto. Tämä on lista riippuvuuksista, joita paketissa pitää olla. Pakettiin voi lisätä riippuvuuksia Npm:n tietovarastosta komennolla *”npm install <paketin_nimi>”.* Paketin riippuvuudet voidaan taas asentaa komennolla *”npm install”,* jolloin riippuvuuksien tiedostot siirtyvät projektin *”node_modules”*-kansioon ja projekti voi alkaa niitä käyttämään. (Node.js 2011.)

4 Muut merkittävät teknologiat

4.1 PostgreSQL

PostgreSQL on ilmainen avoimen lähdekoodin oliorelaatiotietokanta, joka tuo SQL:n lisäksi monia muita ominaisuuksia turvalliseen datan varastointiin ja skaalautumiseen. (PostgreSQL s.a.)

PostgreSQL sisältää monia ominaisuuksia, joiden tarkoituksena on auttaa kehittäjiä rakentamaan sovelluksia, järjestelmänvalvojia suojelemaan tietojen eheyttä ja rakentamaan vikasietoisia ympäristöjä sekä auttamaan sinua hallitsemaan tietojasi riippumatta siitä, kuinka suuri tai pieni tietojoukko on. PostgreSQL on myös erittäin hyvin laajennettavissa. PostgreSQL-tietokannassa voi esimerkiksi määrittää omia tietotyyppejä, rakentaa mukautettuja funktioita ja kirjoittaa koodia eri ohjelmointikielistä kääntämättä tietokantaasi uudelleen. (PostgreSQL s.a.)

4.2 Prisma

Prisma on ilmainen avoimeen lähdekoodiin perustuva uuden sukupolven ORM-kirjasto (object-relational mapping). Prisma pohjautuu Node.js-ajonaikaiseen ympäristöön ja TypeScript-ohjelmointikieleen. Prisma tuo tietokannan päälle abstraktiokerroksen, jonka avulla voidaan tehdä vaivattomasti eri CRUD-operaatiota, eli luoda, lukea, päivittää ja poistaa tietoja tietokannasta. (Prisma s.a.)

Prisma jakaantuu seuraaviin osiin:

- Prisma Client: automaattisesti luotu ja tyyppiturvallinen kyselynrakennustyökalu Node.js:lle ja TypeScriptille
- Prisma Migrate: siirtojärjestelmä (migration)
- Prisma Studio: graafinen käyttöliittymä, jolla tietokannassa olevia tietoja voidaan tarkastella ja muokata (Prisma s.a.)

Prisma Client -työkalua voidaan käyttää missä tahansa tuetussa Node.js- tai TypeScript -back-end-sovelluksessa, joka voi olla esimerkiksi REST- tai GraphQL-ohjelmointirajapinta. (Prisma s.a.)

4.3 Tailwind CSS

Tailwind CSS on JavaScript-pohjainen CSS-sovelluskehys mukautettujen käyttöliittymien nopeaan rakentamiseen. Se on erittäin muokattavissa oleva matalan tason CSS-sovelluskehys, joka antaa kehittäjälle kaikki rakennuspalikat, joita tarvitaan räätälöityjen mallien rakentamiseen ilman itsepäisiä valmiita tyylejä, joita vastaan pitäisi taistella. (GeeksforGeeks 2021.)

Tailwind CSS ei pakota suunnitteluspesifikaatiota tai miltä sivuston pitäisi näyttää, vaan siinä yhdistetään pieniä komponentteja yhteen ainutlaatuisen käyttöliittymän luomiseksi. Tailwind CSS yksinkertaisesti ottaa raa'an CSS-tiedoston, käsittelee tämän CSS-tiedoston määritystiedoston kautta ja tuottaa tulosteen. (GeeksforGeeks 2021.)

```
1   import React from "react";
2
3   const HelloWorld = () => {
4     return (
5       <div className="flex justify-center items-center">
6         <h2 className="text-lg">Hello World!</h2>
7         <p className="m-2">This is a paragraph.</p>
8       </div>
9     );
10  };
11
12  export default HelloWorld;
13
```

Kuva 7. Esimerkki keskitetystä React-komponentista käyttäen Tailwind CSS -kehystä

Kuvassa 7 on määritetty HelloWorld-niminen React-komponentti. Komponentin palautusarvossa voidaan nähdä div-HTML-elementti, jonka sisällä on muita HTML-elementtejä. Reactissa *"className"*-attribuutti tarkoittaa samaa asiaa, kun HTML:ssä *"class"*-attribuutti, eli käytettävää CSS-luokan nimeä.

Normaalisti HTML-elementeille pitäisi määrittää tyylit erilliseen CSS-tiedostoon, mutta Tailwind CSS:n tapauksessa sama saadaan tehtyä lyhyillä avainsanoilla "class"-attribuutin sisällä. Div-elementin sisällä oleva määritys "flex justify-center items-center" tarkoittaa elementin keskitystä. Div-elementin sisällä olevassa h2-elementissä "text-lg" tarkoittaa oletuksena fonttikokoa 20 pikseliä. Alla olevan p-elementin määritys "m-2" tarkoittaa samaa kuin CSS:n "margin: 8px", eli 8 pikselin kokoista marginaalia joka suuntaan. Elementtien luokkanimistä voidaan huomata, että tyylimäärityksiä saa tehtyä todella kompaktisti ja ei tarvitse liikkua eri tiedostojen välillä.

5 Projektin elinkaari

5.1 Vaatimusmäärittely

Tässä projektissa oli tärkeää, että työpaikkailmoitusten käsittely saadaan automatisoida mahdollisimman pitkälle.

Atkins ry:n vanha työpaikkailmoituksen jättöprosessi on seuraavanlainen:

1. Yrityksen rekrytoija täyttää työpaikan tiedot lomakkeeseen tai lähettää tiedot Atkins-ainejärjestön hallituksen sähköpostiin
2. Ilmoituksen tiedot saapuvat sähköpostiin
3. Atkins ry:n työntekijä syöttää manuaalisesti ilmoituksen tiedot WordPress-järjestelmässä kahdesti eri kielille
4. Nyt ilmoitus näkyy sivuilla

Uusi prosessi sovelluksen julkaisun jälkeen on seuraavanlainen:

1. Yrityksen rekrytoija syöttää ilmoituksen tiedot lomakkeella järjestelmään
2. Ilmoitus siirtyy uuden järjestelmän hallintapaneeliin
3. Atkins ry:n työntekijä käy muuttamassa ilmoituksen tilaa yhtä nappia painamalla. Tilat ovat ODOTTAA, HYLÄTTY, HYVÄKSYTTY, YHTEISTYÖKUMPPANUUS
4. Nyt ilmoitus näkyy työpaikkapalvelussa
5. Ilmoitus poistuu oletusnäkymästä myös deadlinen mennessä

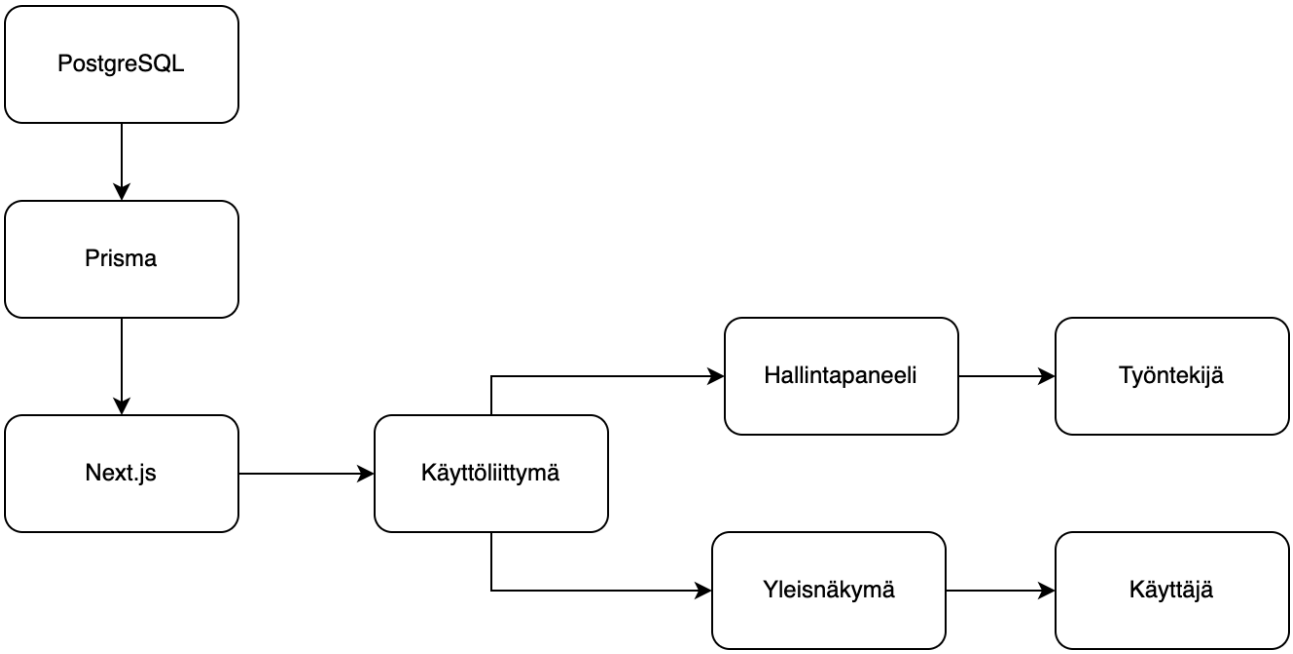
Projektin kirjatut vaatimukset ovat:

- Palvelun etusivulta pitää nähdä avoinna olevat työpaikat tiivistetysti
- Rekrytoijat voivat lisätä ilmoituksiaan lomakkeella, ylläpitäjän pitää vain hyväksyä ne
- Työpaikkailmoituksella pitää olla ominaisuudet
    - Otsikko
    - Työpaikkakuvaus
    - Rekrytoijan nimi ja sähköposti mahdollista yhteydenottoa varten
    - Työhön vaadittavat taidot
    - Ilmoituksen julkaisupäivämäärä
    - Ilmoituksen viimeinen hakupäivämäärä
    - Ilmoituksen tila (odottaa hyväksyntää, hyväksytty, yhteistyökumppanuus ja hylätty)
    - Linkki työpaikan hakuun
- Työpaikkailmoitusten suodatus työpaikan hakusanalla, yrityksen nimellä tai teknologioilla
- Yksittäinen työpaikkailmoitussivu, missä ilmoituksesta näkee tarkempia yksityiskohtia
- Hallintapaneeli, jossa ilmoituksia voi muokata, poistaa tai muuttaa tilaa

- Kirjautuminen hallintapaneeliin

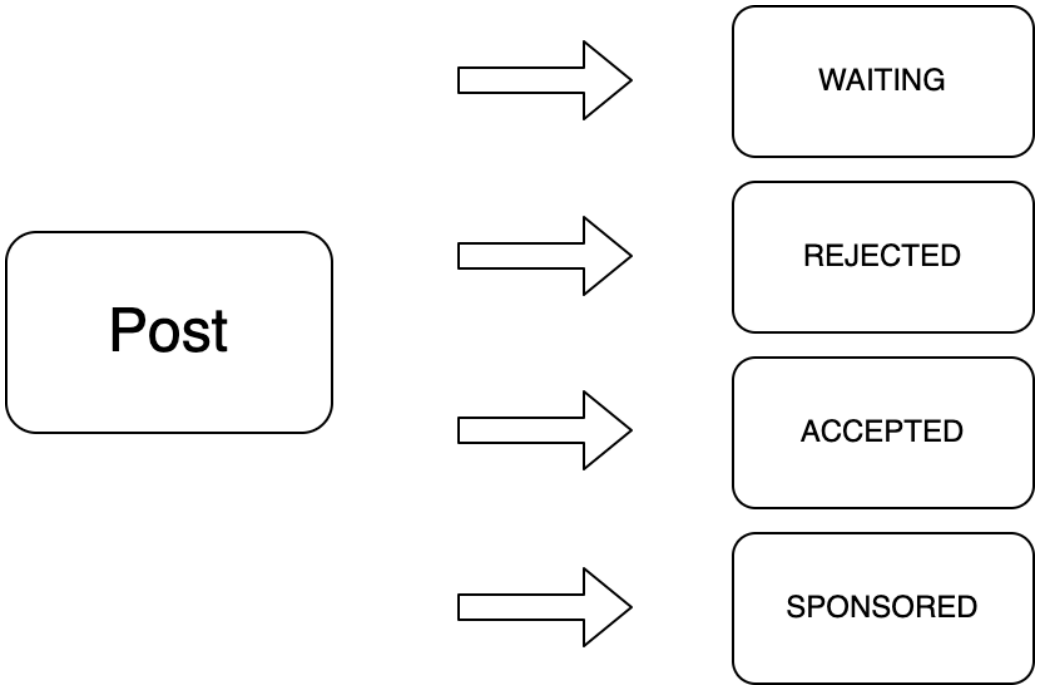
5.2 Suunnittelu

Tein suunnittelua varten arkkitehtuurikaavion ja kaavion työpaikkailmoituksen mahdollisista tiloista.



Kuva 8. Arkkitehtuurikaavio

Kuvasta 8 voidaan nähdä palvelun arkkitehtuuri. Sovellus rakentuu Next.js-sovelluskehyksen ympärille. Next.js yhdistää sovelluksen back-end- ja front-end-puolen. Next.js tekee tietokantaoperaatioita PostgreSQL-tietokantaan Prisman välityksellä. Näiden operaatioiden datat välitetään joko käyttöliittymän yleisnäkymään tai hallintapaneeliin. Hallintapaneeli on salasanasuojattu ja sinne on oikeus päästä vain Atkins ry:n työntekijöillä. Käyttöliittymän yleisnäkymän voi nähdä kuka tahansa käyttäjä.



Kuva 9. Työpaikkailmoituksen tila

Kuvassa 9 näkyy työpaikkailmoitus (Post) ja sen mahdolliset tilat, joita ovat ”WAITING”, ”REJECTED”, ”ACCEPTED”, ”SPONSORED” (ODOTTAA, HYLÄTTY, HYVÄKSYTTY, SPONSOROITU). Työpaikkailmoituksen oletustila on ”WAITING”, joka asetetaan ilmoitukselle automaattisesti ilmoitusta luotaessa. Hallintapaneelista käsin ilmoituksen tilaa pitäisi pystyä muuttamaan nappia painamalla.

5.3 Toteutus

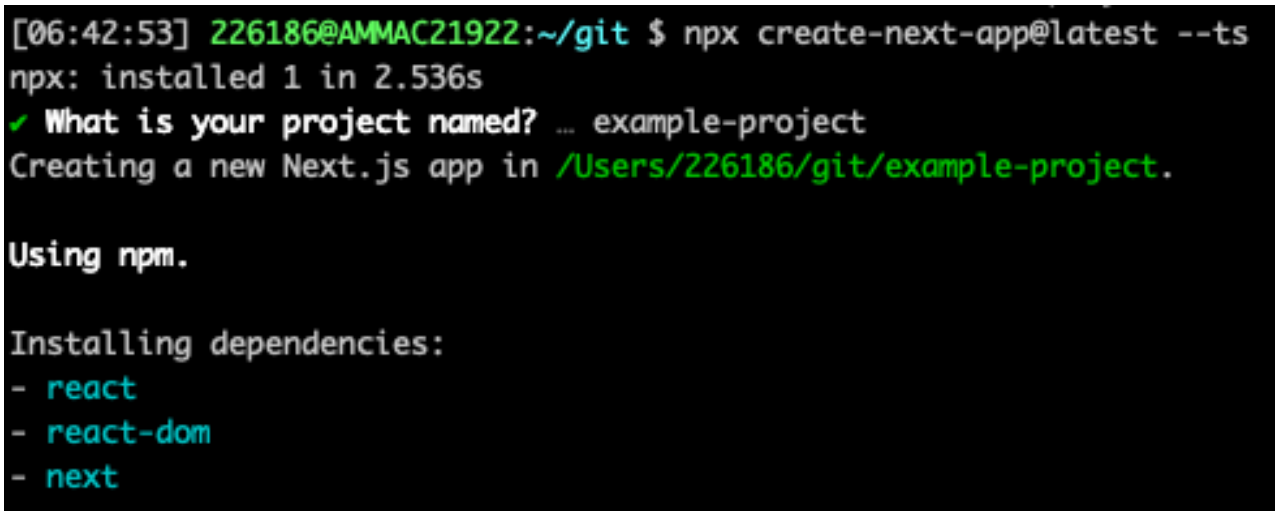
5.3.1 Versionhallinta

Aloitin projektin luomalla Git-tietovaraston (repository) GitHub-palveluun. Versionhallinnan luominen projektia varten on erityisen tärkeää, koska se mahdollistaa tiedostojen seurannan, jolloin esimerkiksi virheiden korjaaminen on helppoa sekä useampi henkilö voi rakentaa palvelua samaan aikaan.

GitHubiin Git-tietovaraston luominen on suoraviivaista. Tein omilla GitHub-tunnuksillani ”atkins-job-board”-nimisen Git-tietovaraston, johon pystyn nyt tallentamaan koodini.

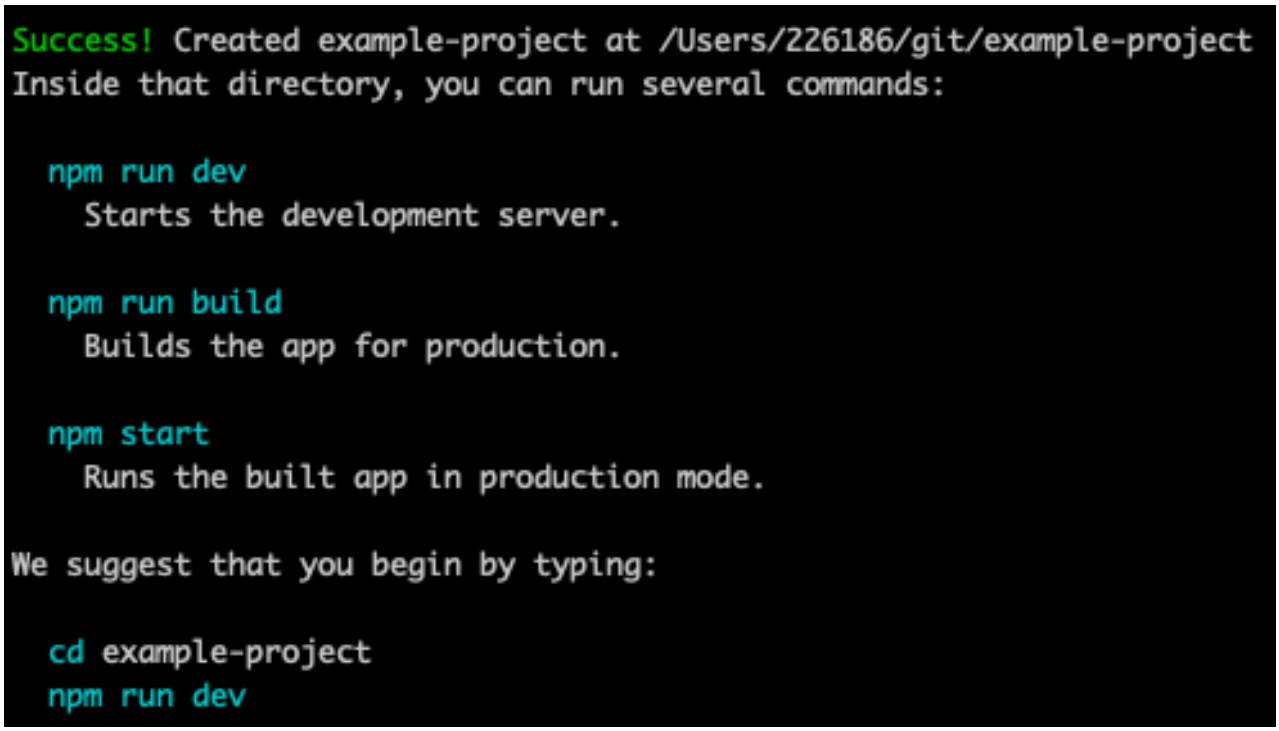
5.3.2 Next.js-projekti

Projekti pohjautuu teknologiallisesti vahvasti Next.js-sovelluskehykseen. Next.js-projekti voidaan luoda käyttämällä Npm-paketinhallitsijaa. Next.js:n kehittäjät ovat luoneet ”create-next-app”-nimisen komentoliittymätyökalun, joka on lisätty Npm:n pakettirekisteriin.



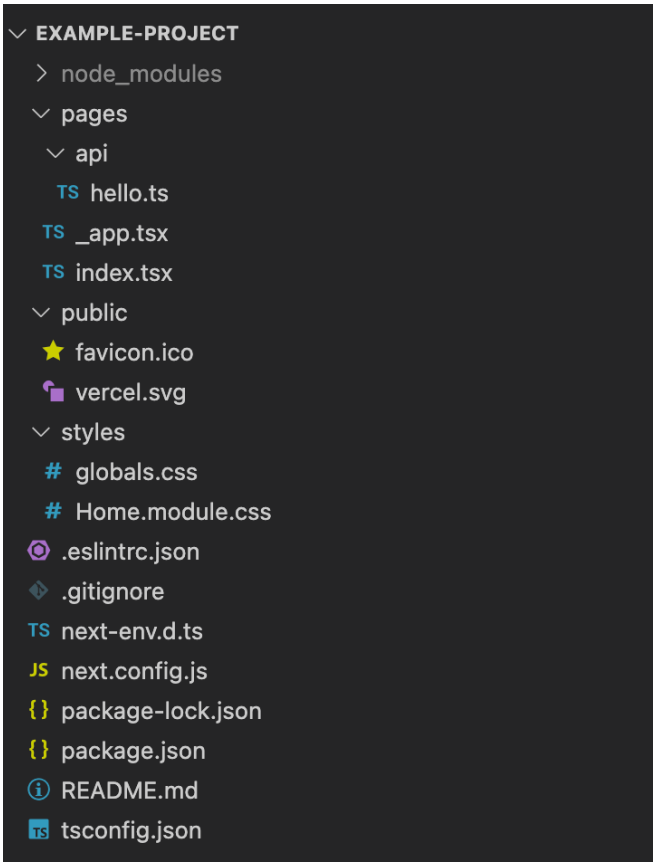
Kuva 10. Next.js:n asennus komentoliittymää käyttäen.

Next.js:n asennus onnistuu kuvan 10 mukaisen *npx*-komennon avulla *”npx create-next-app@latest –ts*”. Komento asentaa valmiin tuotantokelpoisen Next.js-projektipohjan, jolle voidaan antaa haluamansa nimi. ”—ts”-optio tarkoittaa tässä tapauksessa TypeScriptia, jota hyödynnän projektissa JavaScriptin sijasta.



Kuva 11. Onnistunut Next.js-asennus

Kuva 11 näyttää, miltä onnistunut asennus näyttää komentotulkissa. Nyt Next.js-projektia voidaan alkaa kehittämään tekstieditorissa ja sovellus saadaan pyörimään paikallisessa kehitysympäristössä ajamalla komentoliittymässä komento *”npm run dev”*.



Kuva 12. Valmis Next.js-projektipohja ja -rakenne

Kuvassa 12 näkyy, miltä näyttää Next.js-projektipohja. Nyt Next.js renderoi projektin juureen ”pages”-kansiossa olevan ”index.tsx”-tiedoston. Tätä muokkaamalla pääsen heti muokkaamaan sovelluksen aloitussivua. Samassa paikkaa on myös ”_app.tsx”-tiedosto, jossa määritetyt komponentit näkyvät kaikkien eri polkujen näkymissä. ”api”-kansiosta löytyy myös valmis päätepiste, johon voidaan tehdä http-kutsuja normaalin REST-ohjelmointirajapinnan tapaan.

5.3.3 Prisma-skeema

Sovelluksen Prisma-skeema on yksinkertainen. Sovellus vaatii vain yhden mallin, jonka nimesin Post-malliksi (ilmoitus).

```
13  model Post {
14    id                Int        @id @default(autoincrement())
15    title             String     @db.VarChar(255)
16    company           String     @db.VarChar(255)
17    name              String     @db.VarChar(255)
18    email             String     @db.VarChar(255)
19    description       String
20    requiredSkills    String[]
21    recommendedSkills String[]
22    url               String
23    createdAt         DateTime   @default(now())
24    updatedAt         DateTime   @updatedAt
25    published         DateTime?
26    deadline          DateTime?
27    status            PostStatus @default(WAITING)
28  }
29
30  enum PostStatus {
31    WAITING
32    APPROVED
33    REJECTED
34    SPONSORED
35  }
```

Kuva 13. Post-malli ja PostStatus-enumi

Kuvassa 13 esitetystä Post-mallista voidaan huomata, että sillä on id-attribuutti, eli yksilöivä luku-tyyppinen tunniste, joka korottuu automaattisesti uuden ilmoituksen luonnin yhteydessä. Post-mallissa on monia merkkijonotyyppisiä attribuutteja, joista osan koko on rajoitettu 255 merkkiin. *”String[]”* tarkoittaa tässä merkkijonotyyppistä taulukkoa, jotka on asetettu työpaikkailmoituksen työpaikan vaatimuksiksi. DateTime-tyypit tarkoittavat päivämäärää, joissa aika määritetään millisekunteina. Viimeinen attribuutti status, käyttää PostStatus-enumia, jolle on määritetty tilat WAITING, APRROVED, REJECTED ja SPONSORED, eli odottaa, hyväksytty, hylätty, yhteistyö. Uudelle Post-tietueelle annettaan odottaa-tila, joka on tässä määritetty *”@default”*-annotaatiolla.

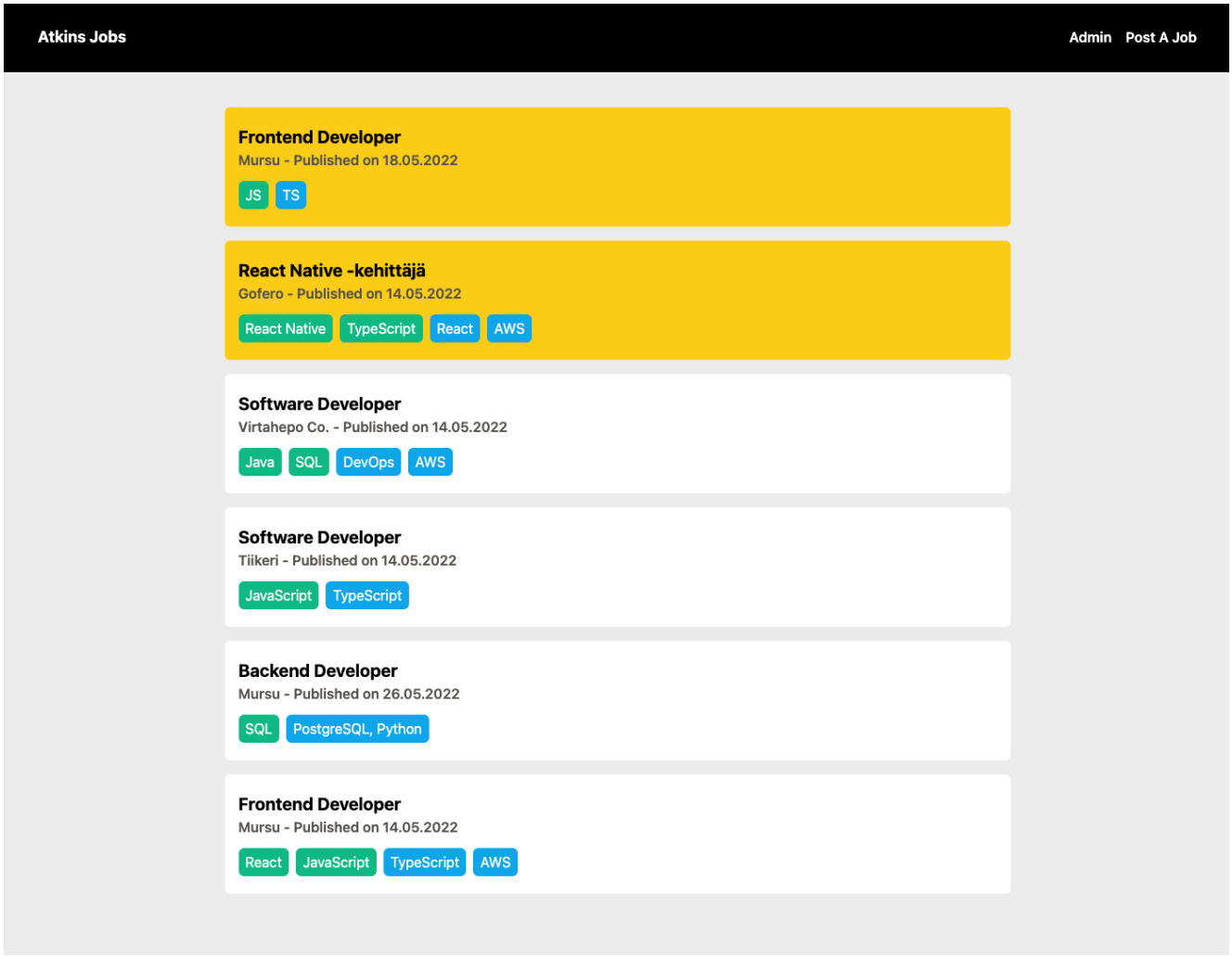
5.3.4 Tietokanta

Päätin käyttää palvelussa PostgreSQL-relaatiotietokantaa, koska olen työskennellyt sen kanssa aikaisemmin ja se on todettu kehittäjien keskuudessa yleisesti hyväksi ratkaisuksi. Isännöin

PostgreSQL-relaatiotietokannan Heroku-julkaisupalvelussa, sen helppouden, nopeuden ja ilmaisuuden takia.

5.3.5 Palvelun näkymät

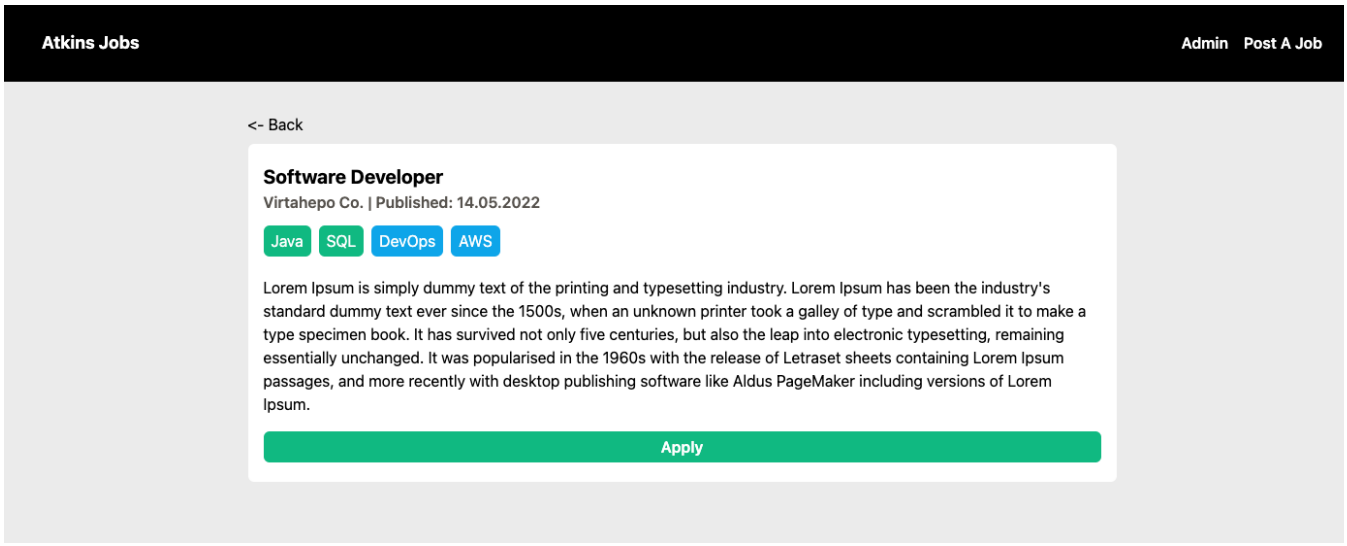
Tässä luvussa esitellään palvelun näkymiä ja tuloksia.



Kuva 14. Palvelun etusivu ja työpaikkalistaus

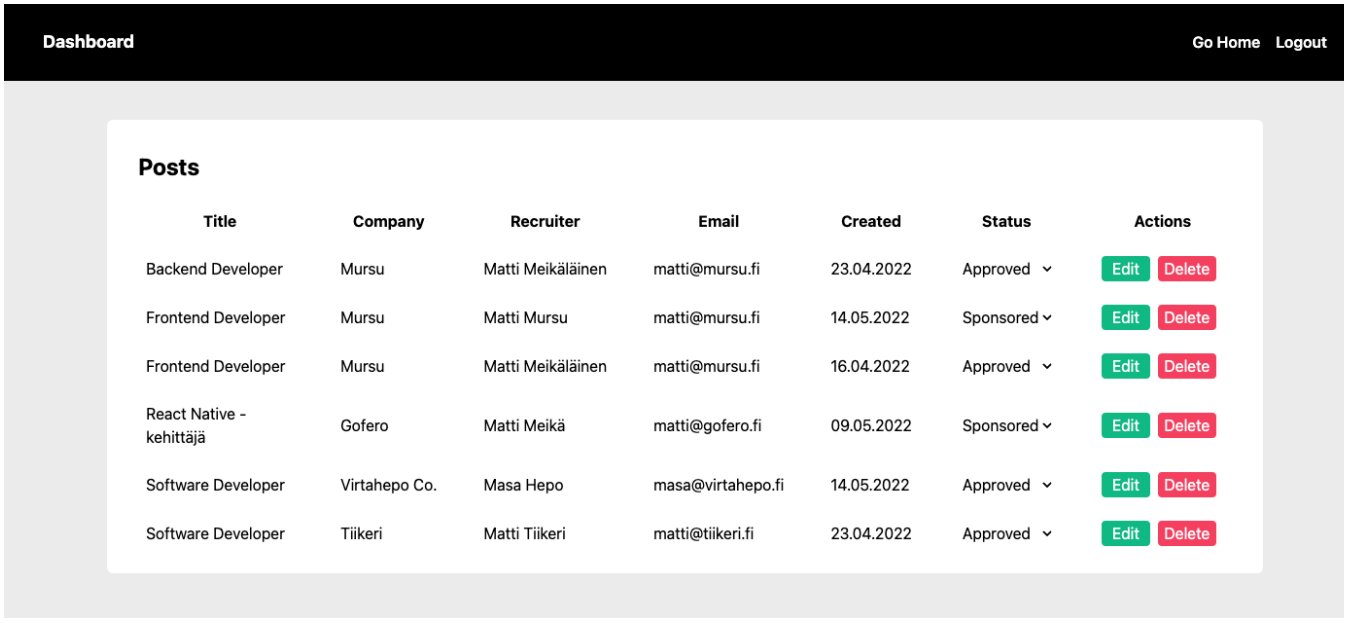
Kuvassa 14 voidaan nähdä, miltä palvelun etusivu näyttää. Palvelun etusivulla näkyy kaikki hallintapaneelista hyväksytyt työpaikkailmoitukset. Keltaisella olevat työpaikkailmoitukset ovat niin sanottuja yhteistyöilmoituksia. Ilmoituslaatikot ovat PostCard-nimisiä komponentteja. Niissä on työpaikan nimi, yrityksen nimi, julkaisupäivämäärä ja vaaditut ja suositellut taidot työpaikkaa varten.

Navigaatiossa voidaan nähdä linkit ”Atkins Jobs”, ”Admin” ja ”Post A Job”. ”Admin”-linkki tulee poistumaan tuotantoversiossa navigaatiosta, se on ollut siinä vain helpottaakseen kehittämistä.



Kuva 15. Yksittäisen työpaikkailmoituksen sivu

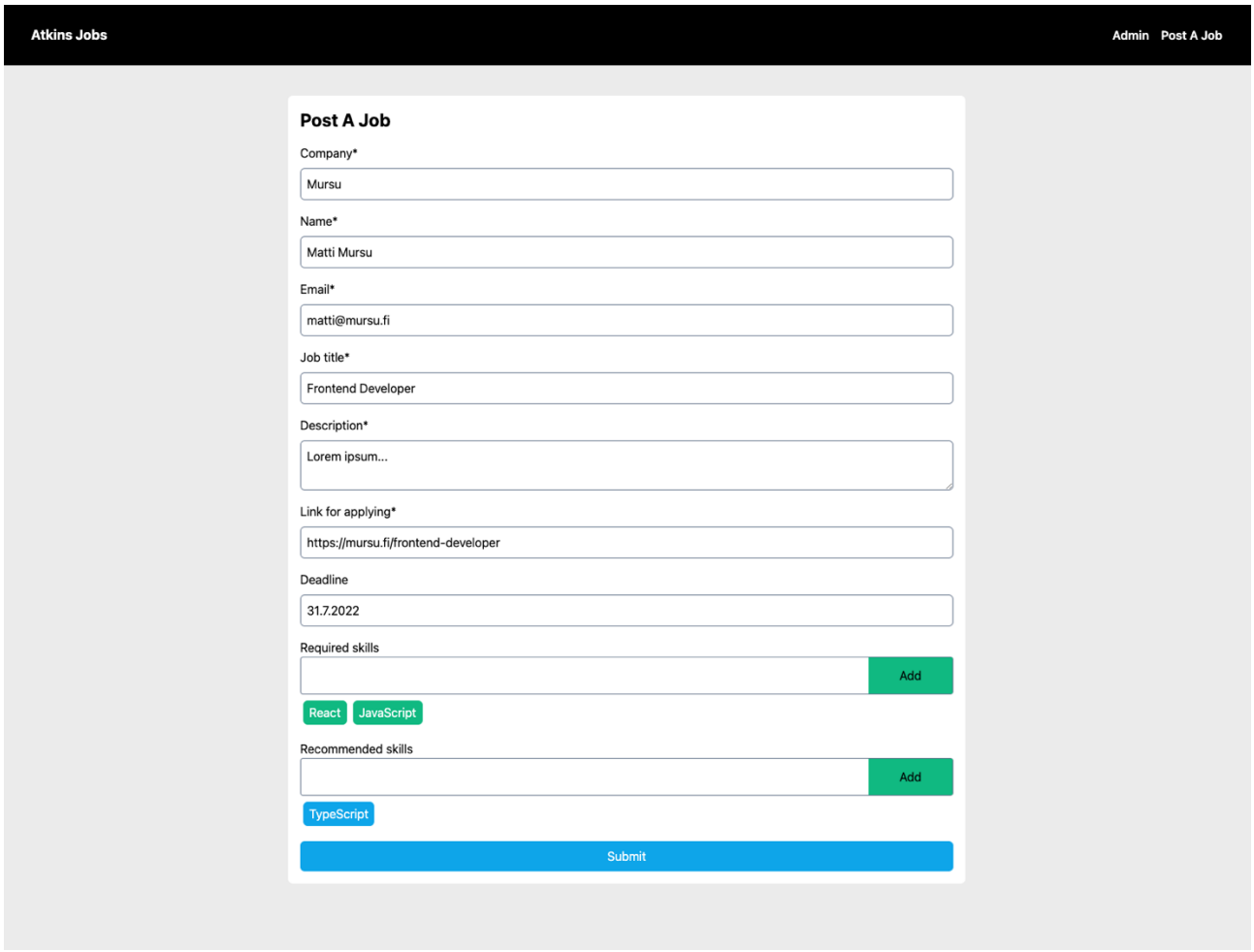
Kuvassa 15 on yksittäisen työpaikkailmoituksen sivu, jossa on muuten samat tiedot kuin etusivun ilmoituksissa, mutta siellä lisäksi työpaikan kuvaus ja linkki työpaikan hakemiseksi.



Kuva 16. Hallintapaneeli (admin)

Kuvassa 16 on palvelun hallintapaneeli, jossa Atkins ry:n työntekijät voivat muokata palveluun jätettyjä työpaikkailmoituksia. Hallintapaneelissa näkyy työpaikan tiedot ja sen statusta voi muuttaa, työpaikkaa voi muokata tai sen voi poistaa.

Hallintapaneeli on salasanasuojattu muiden käyttäjien pääsyn epäämiseksi. Se käyttää evästeitä tämän mahdollistamiseen.



Kuva 17. Työpaikkailmoituksen jättölomake

Kuvassa 17 on työpaikkailmoituksen jättölomakkeen näkymä. Kuten kuvasta voi nähdä rekrytoija antaa ilmoitukselle yrityksen nimen, yhteystietonsa mahdollista Atkins ry:n työntekijän yhteydenottoa varten, työpaikan tittelin, työpaikkakuvauksen, linkin työpaikan hakusivulle, mahdollisen deadlinen ja sekä vaaditut ja suositellut taidot. Jos lomakkeen lähettäminen onnistuu ”Submit”-nappulaa painamalla, saapuu se palvelun hallintapaneeliin Atkins ry:n työntekijän käsiteltäväksi.

5.4 Testaus

Projektilla olisi hyvä olisi erilaisia testejä, kuten esimerkiksi yksikkö-, integraatio- ja end-to-end-testejä. Tämän projektin tapauksessa tämäntyyppiset testit jäävät nyt puuttumaan projektin aikataulupaineiden alla. Tarkoitus olisi lisätä erilaisia testejä projektin julkaisun jälkeen jatkokehitysvaiheessa.

Tässä projektissa testaus on toteutettu manuaalitestauksella. Olen testannut käyttöliittymää manuaalisesti klikkailemalla lähinnä erilaisia polkuja eri näkymiin. Lisäksi olen testannut työpaikkailmoituksen jättölomaketta syöttämällä erilaisia työpaikkoja manuaalisesti käsin lomakkeeseen.

Lisäksi olen myös testannut REST API:n päätepisteitä lähettämällä dataa eri päätepisteihin Postman-ohjelmalla.

5.5 Julkaisu

Julkaisin projektin demoversion Vercel-julkaisupalveluun (vercel.com), koska Vercel on yritys, joka kehittää Next.js-sovelluskehystä ja Next.js-sovelluksen julkaisu siellä on erittäin helppoa. Käytännössä Verceliin pitää tehdä tunnukset tai sinne voi kirjautua esimerkiksi omilla GitHub-tunnuksilla. Kun tunnukset on luotu, voidaan projektin julkaisu aloittaa suoraan GitHub-tietovarastosta. Kun GitHub-tunnus on yhdistetty Verceliin onnistuu projektin julkaisu nappia painamalla. Vercel kuuntelee projektin GitHub-tietovaraston muutoksia ja aina kun tietovaraston main-haaraan pusketaan uusia muutoksia, menevät muutokset julkaisuputken läpi tuotantoon. Nyt projektin demoversio on toistaiseksi toiminut osoitteessa https://atkins-jobboard.vercel.app/.

Demoversio käyttää Vercelin kapeinta pakettia nimeltä Hobby, joka on ilmainen. Tämän paketin ominaisuudet ei kuitenkaan välttämättä ole tarpeeksi laajat projektin tuotantoversiota varten. Maksullinen Pro-versio maksaisi 20 dollaria kuussa, minkä lisäksi tulisi vielä maksettavaksi erikseen isännöitävä tietokanta (Vercel s.a.). Tämä olisi kuitenkin kohtuullisen kallista yhdistykselle yhden palvelun takia, minkä takia tarkastelin muita halvempia vaihtoehtoja.

Vielä tämän opinnäytetyön kirjoittamishetkellä ei sopivaa palvelua ole vielä valittu, mutta se tullaan valitsemaan pian. Yksi potentiaalinen vaihtoehto olisi DigitalOcean-pilvipalvelutarjoajan Apps-alusta, joka maksaisi 12 dollari kuukaudessa tietokantakustannusten lisäksi.

5.6 Ylläpito

Projektin ylläpito siirtyy nyt sen valmistuessa Atkins ry:lle. Tein Atkins ry:lle GitHubiin Atkins ry -nimisen organisaation, johon kloonasin toteuttamani projektin Git-tietovaraston. Annoin Atkins ry:n puheenjohtajalle ylläpito-oikeudet tähän organisaatioon, joten Atkins ry pystyy nyt näkemään ja kloonaamaan projektin itselleen ja tekemään itse siihen haluamiaan muutoksia.

Sovin lisäksi Atkins ry:n kanssa, että voin pientä korvausta vastaan tehdä palveluun ylimääräisiä korjauksia ja lisäominaisuuksia ja antaa tukea mahdollisissa ongelmissa ainakin vielä tänä vuonna.

6 Pohdinta

Projekti onnistui kokonaisuutta katsellen hyvin. Projektin tärkeimmät vaatimukset saatiin tehtyä ja saavutettua. Atkins ry:llä on nyt käyttökelpoinen työpaikkapalvelu, jota on hyvä lähteä jatkokehittämään. Projektia painoi kuitenkin kovat aikataulupaineet, koska jouduin tekemään opinnäytetyöni hyvin lyhyessä ajassa valmistuakseni.

Yksittäisiä onnistumisia olivat pienimmän toimivan tuotteen (MVP) valmistuminen. Palvelussa voidaan selata työpaikkailmoituksia, rekrytoijat voivat lomakkeen avulla ilmoittaa työpaikkailmoituksensa, Atkins ry:n työntekijät voivat muokata työpaikkailmoituksen tilaa, sponsoroidut työpaikkailmoitukset ovat selvästi näkyvillä ja työpaikkailmoituksilla on oma tarkemmat tiedot sisältävät sivunsa.

Projekti kärsi pienistä puutteista ajan loppumisen vuoksi. Tästä syystä esimerkiksi hakuominaisuus, rikas tekstieditori ja pienet käyttöliittymän tyylisäädöt jäivät tästä opinnäytetyön aikaisesta projektista pois. Ilman näitäkin sovellus on käytettävä ja valmiina tuotantoon. Ehdottomasti tärkein asia mikä olisi tehnyt projektin tuloksista paremmat, on projektin aloittaminen aikaisemmin. Pelkästään kuukausikin lisää olisi auttanut todella paljon.

Projektin aikana totesin, että Next.js-sovelluskehys on ehkä paras käyttämäni React-kirjasto tai -sovelluskehys. Next.js-sovelluskehyksessä on todella hyvä kehittäjäkokemus, sen tiedostojärjestelmäpohjaisen reitittäjän ja monien muiden pienten ominaisuuksien ansiosta. Ehdottomasti paras ominaisuus on kuitenkin, että kehittäjä saa valita renderöintitavan helposti ilman mitään ylimääräisiä kirjastoja.

Tailwind CSS -kehys oli mielenkiintoinen tapa kehittää käyttöliittymän tyylejä. Sovelluskehyksessä oli suhteellisen suuri oppimiskäyrä, koska kaikkia tyyliavainsanoja ei tiennyt alussa, ja ne täytyi etsiä dokumentaatiosta, minkä takia kehittäminen oli aluksi hidasta. Pikkuhiljaa kuitenkin, kun alkoi muistamaan avainsanoja ulkoa ja oppimaan dokumentaation käyttöä, kehittäminen nopeutui huomattavasti.

Projektin jatkokehityksessä tullaan todennäköisesti kehittämään ensimmäisenä rikas tekstieditori siistimpien työpaikkakuvausten vuoksi, työpaikkojen hakuominaisuus ja pieniä tyylimuutoksia. Muita mahdollisia ominaisuuksia olisi käyttäjätunnukset rekrytoijille, jolloin he pääsisivät muokkaamaan omia ilmoituksiaan ja uusien sivujen luominen hallintapaneelista (esimerkiksi infosivu).

Lähteet

FreeCodeCamp 2019. The Next.js Handbook. Luettavissa: https://www.freecodecamp.org/news/the-next-js-handbook. Luettu: 4.5.2022.

GeeksforGeeks 2021. Introduction to Tailwind CSS. Luettavissa: https://www.geeksforgeeks.org/introduction-to-tailwind-css/. Luettu 21.5.2022.

Google Search Central s.a. Understand the JavaScript SEO basics. Luettavissa: https://developers.google.com/search/docs/advanced/javascript/javascript-seo-basics. Luettu: 10.5.2022.

Lewis, S. 2019. waterfall model. SearchSoftwareQuality. Luettavissa: https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model. Luettu: 19.5.2022.

MDN Web Docs 2022. What is JavaScript? Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Luettu: 3.5.2022.

Martin, M. 2022. SDLC (Software Development Life Cycle): What is, Phases & Models. Guru99. Luettavissa: https://www.guru99.com/what-is-sdlc-or-waterfall-model.html. Luettu 23.5.2022.

Next.js s.a. a. What is Next.js? Luettavissa: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs. Luettu: 4.5.2022.

Next.js s.a. b. Data fetching: getStaticProps. Luettavissa: https://nextjs.org/docs/basic-features/data-fetching/get-static-props. Luettu: 8.5.2022.

Next.js s.a. c. Data fetching: getServerSideProps. Luettavissa: https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props. Luettu: 8.5.2022.

Next.js s.a. d. Data fetching: Client side. Luettavissa: https://nextjs.org/docs/basic-features/data-fetching/client-side. Luettu: 8.5.2022.

Node.js 2011. What is npm? Luettavissa: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. Luettu: 10.5.2022.

Node.js s.a. About Node.js. Luettavissa: https://nodejs.org/en/about/. Luettu: 10.5.2022.

PostgreSQL s.a. About. Luettavissa: https://www.postgresql.org/about/. Luettu: 14.5.2022.

Prisma s.a. What is Prisma? Luettavissa: https://www.prisma.io/docs/concepts/overview/what-is-prisma. Luettu: 21.5.2022.

React s.a. Tutorial: Intro to React. Luettavissa: https://reactjs.org/tutorial/tutorial.html. Luettu: 8.5.2022.

Seobility Wiki s.a. Rendering. Luettavissa: https://www.seobility.net/en/wiki/Rendering. Luettu: 9.5.2022.

Tutorials Point s.a. SDLC – Waterfall model. Luettavissa: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. Luettu: 22.5.2022.

TypeScript Tutorial s.a. What is TypeScript? Luettavissa: https://www.typescripttutorial.net/typescript-tutorial/what-is-typescript/. Luettu: 10.5.2022.

Vercel s.a. Pricing. Luettavissa: https://vercel.com/pricing. Luettu 23.5.2022.

Waseem, A. 2022. Waterfall methodology: history, principles, stages & more. Management Library. Luettavissa: https://managementhelp.org/waterfall-methodology. Luettu: 22.5.2022.