LAPIN AMK
Lapland University of Applied Sciences

TETRIS-PELIN LUOMINEN FULL-STACK-TEKNOLOGIOILLA

Laatikainen Christopher

Opinnäytetyö

Tietojenkäsittelyn koulutus
Tradenomi (AMK)

2024

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tietojenkäsittelyn koulutus
Tradenomi (AMK)

Tekijä Christopher Laatikainen **Vuosi** 2024
Ohjaaja(t) Juha Orre
Toimeksiantaja
Työn nimi Tetris-pelin luominen full-stack-teknologioilla
Sivumäärä 48

Tämän opinnäytetyön aiheena oli tutustua kahteen web-kehityksessä käytettyyn teknologiaan, eli Reactiin ja ASP.NET Coreen. Näiden teknologioiden pohjalta toteutettiin Tetriksen tyyppinen peli. Opinnäytetyön tarkoituksena oli tutkia Reactin ja ASP.NET Coren käyttöä ja niiden eroja muihin teknologioihin sekä luoda Tetris-peli käyttäen näitä teknologioita. Tavoitteena oli oppia teknologioiden käyttöä ja luoda toimiva peli.

Pelin toteutusosassa käsiteltiin front- ja back-endin ohjelmointiympäristön ja työkalujen asennusta. Sen jälkeen luotiin uusi projekti. Lopuksi esiteltiin valmiin pelin kulkua sekä koodin rakennetta ja selityksiä sen toiminnasta. Työ ei pidä sisällään kaikkea pelin koodia, vaan tarkoituksena oli esitellä konsepteja, joita voi tarvita pelin teossa. Työn lopussa käytiin myös läpi ongelmatilanteita pelin kehityksessä, verrattiin työssä käytettyjä teknologioita muihin teknologioihin sekä pohdittiin työntekoa yleisesti.

Opinnäytetyön tuloksena syntyi Tetris-peli, jossa toimii perusmekaniikat. Pelissä on kehitettävää muun muassa tietoturvan ja mekaniikkojen parantamisen kanssa. Opinnäytetyötä on mahdollista hyödyntää Tetriksen teossa sekä yleisesti ohjelmointityössä, etenkin Reactia ja ASP.NET Corea käytettäessä.

Avainsanat verkko-ohjelmointi, peliohjelmointi, videopelit
Muita tietoja Työn löytää osoitteesta https://github.com/claatikainen/reactris.

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Degree Programme in Data Processing
Bachelor of Business Administration

Author	Christopher Laatikainen	**Year**	2024
Supervisor	Juha Orre		
Commissioned by			
Subject of thesis	Creating a Tetris game with Full Stack technologies		
Number of pages	48		

The topic of this thesis deals with the usage of two web technologies, React and ASP.NET Core. A Tetris game was created based on these technologies. The purpose of this thesis was to study the usage of React and ASP.NET Core, compare them to other technologies, and create a Tetris game using these technologies. The objective was to learn to use these technologies and create a functioning game.

The section on developing the game included installation and setup instructions of development environment and tools used both in the frontend and backend. A new project was created using these tools, and the operation of the finished game as well as the code used in the game and its explanation were presented. The thesis does not include all the code used in the game, instead the goal is to introduce concepts that may be needed in developing a Tetris game. The problems that arose during the development of the game are discussed at the end of the thesis. The technologies used in the thesis were compared with other technologies. Finally, a general reflection on the thesis was presented.

The result of this thesis is a Tetris game in which the basic game mechanics works. Security and some mechanics among other things can be developed further. The thesis can be used when developing a Tetris game and generally in software development, especially when using React and ASP.NET Core.

Key words web programming, game programming, video games

Special remarks The game is accessible from https://github.com/claatikainen/reactris.

SISÄLLYS

1 JOHDANTO 6
2 TETRIKSEN HISTORIA JA TOIMINTA 8
3 KÄYTETYT TEKNOLOGIAT 11
3.1 Reactin historia ja toiminta 11
3.2 ASP.NET Core ja sen erot ASP.NETiin 13
4 LUOMANI TETRIS-PELIN TOTEUTUS 15
4.1 Front-end 15
4.1.1 Ympäristön perusedellytykset 15
4.1.2 Uuden projektin luominen 15
4.1.3 Valmiin pelin kulku 16
4.1.4 Front-endin rakenne 20
4.2 Back-end 30
4.2.1 Ympäristön perusedellytykset 30
4.2.2 API-projektin luominen 30
4.2.3 Back-endin rakenne 36
5 ONGELMATILANTEET PELIN KEHITYKSESßÄ 42
6 TEKNOLOGIOIDEN VERTAILU 44
6.1 Front-endien vertailu 44
6.2 Back-endien vertailu 44
6.3 Päätelmät 46
7 POHDINTA 48
LÄHTEET 49

KÄYTETYT LYHENTEET JA TERMIT

React.js	Javascript-kirjasto UI:den rakentamiseen
Node.js	ympäristö Javascriptin suorittamiseen
Npm	paketinhallintajärjestelmä Javascriptille
ASP.NET Core	ohjelmistokehys web-sovelluksien kehitykseen
JSX	Javascript Syntax Extension, Javascriptin laajennus, joka mahdollistaa HTML:n kirjoittamisen Javascriptillä
JSON	Javascript Object Notation

1 JOHDANTO

Ohjelmoinnissa on monia alueita. Näiden alueiden välillä on eroja sekä niissä käytettävissä teknologioissa että työkysynnässä. (HyperionDev 2017.) Vuonna 2023 eniten kysyttyihin ohjelmointitöihin kuuluu front-end-, back-end- ja full-stack-ohjelmointi (Indeed 2023). Front-end-ohjelmointiin kuuluu interaktiivisten ja hyvännäköisten verkkosivujen kehittäminen. Back-end-ohjelmointi koostuu palvelinpuolisten asioiden toteuttamisesta, kuten tietokannasta ja reitityksestä. Full-stack-ohjelmointi pitää sisällään kummankin, front- ja back-endin. (Amir 2023.) Opinnäytetyön aihe on verkkoteknologioihin perustuva full-stack Tetris-peli. Käytän front-endinä React-kirjastoa ja back-endinä ASP.NET Corea. Opinnäytetyössä tutkin, miten Reactia ja ASP.NET Corea käytetään, ja niiden eroja muihin teknologioihin.

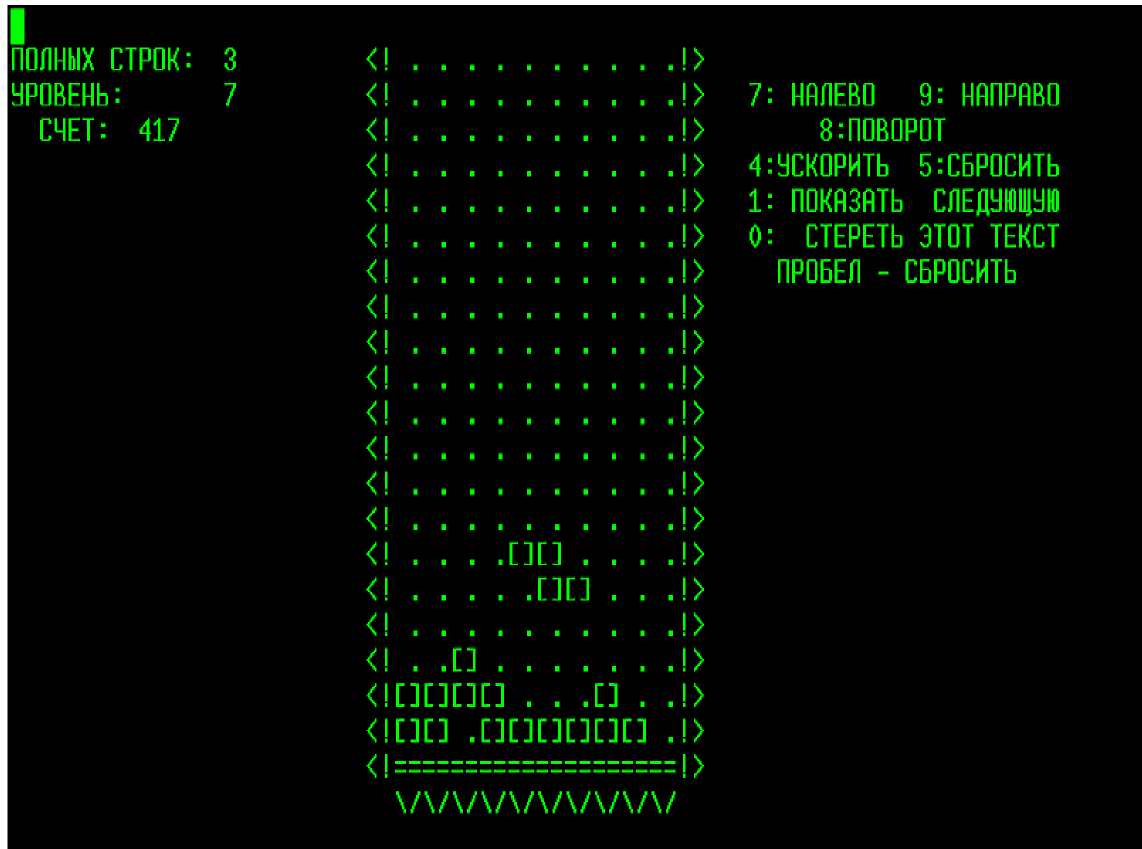
Opintojeni aikana olen käyttänyt monia ohjelmointikieliä sekä teknologioita. Olen työskennellyt muun muassa web- ja mobiiliohjelmoinnissa, front- ja back-endissä sekä pelien tuottamisessa. Näistä alueista pidin eniten web-ohjelmoinnista. Tämän takia valitsin opinnäytetyökseni web-teknologioihin perustuvan pelin. Koska full- stack on kysytty ohjelmointityö, päätin käyttää pelissäni myös back-endiä. Olen käyttänyt monia front-end-teknologioita, kuten Vue.js:ää ja Angularia, mutten ole käyttänyt Reactia. Koska React on kyselyn mukaan käytetyin web UI-teknologia (Stack Overflow 2022), päätin käyttää sitä. Back-endinä käytän ASP.NET Corea. Olen käyttänyt ASP.NET Corea yhdessä projektissa, ja pidin siitä. Lisäksi Clark (2022) kirjoittaa ASP.NET Coren olevan ideaali back-end Reactin kanssa käytössä. Näiden syiden takia valitsin ASP.NET Coren.

Tetriksestä on tehty monia front-end-teknologioilla toteutettuja projekteja ja kirjoitettu blogeja niistä. Tällaisten blogien tarkoitus on auttaa lukijaa oppimaan front-end-ohjelmointia. Näissä blogeissa esitetyt Tetris-pelit sisältävät klassisen Tetriksen piirteitä eikä moderneja mekaniikkoja. Tekemäni peli sisältää modernin Tetriksen eli niin sanotun Tetris Guidelinen mekaniikkoja, joita käsittelen työssäni. Työ sisältää front-endin lisäksi back-endin, jota näkee verrattain vähemmän kuin pelkästään front-end-projekteja. Opinnäytetyön perusteluna on kertoa laajemmin Tetris-pelin teosta sisältämällä Tetris Guidelinen mekaniikkoja ja yhdistämällä back-end-projektiin.

Opinnäytetyön tavoite on tehdä web-pohjainen full-stack Tetris-peli. Pelin käyttöliittymä luodaan React-kirjastolla ja back-endinä toimii ASP.NET Core. Raportissa käsittelen toteutusteknologioita, front- ja back-endin kehittämistä, ongelmatilanteita pelin kehityksessä sekä vertailen eri front- ja back-end-teknologioita.

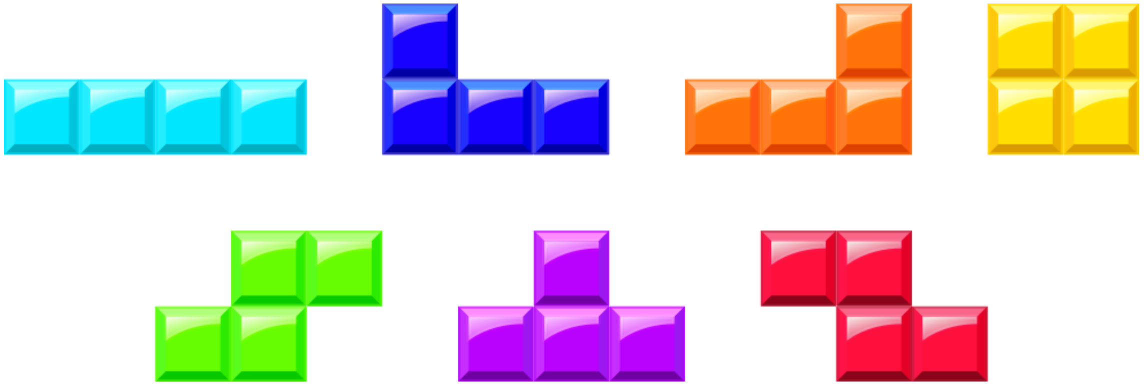
2 TETRIKSEN HISTORIA JA TOIMINTA

Tetris on videopeli, jonka on kehittänyt venäläinen pelisuunnittelija Alexey Pajitnov vuonna 1985. Pelin tarkoituksena on pyörittää tippuvia palikoita kenttien läpäisemiseksi. Tetris kehitettiin alunperin Electronika 60 -tietokoneelle (Britannica 2024). (Kuvio 1.)



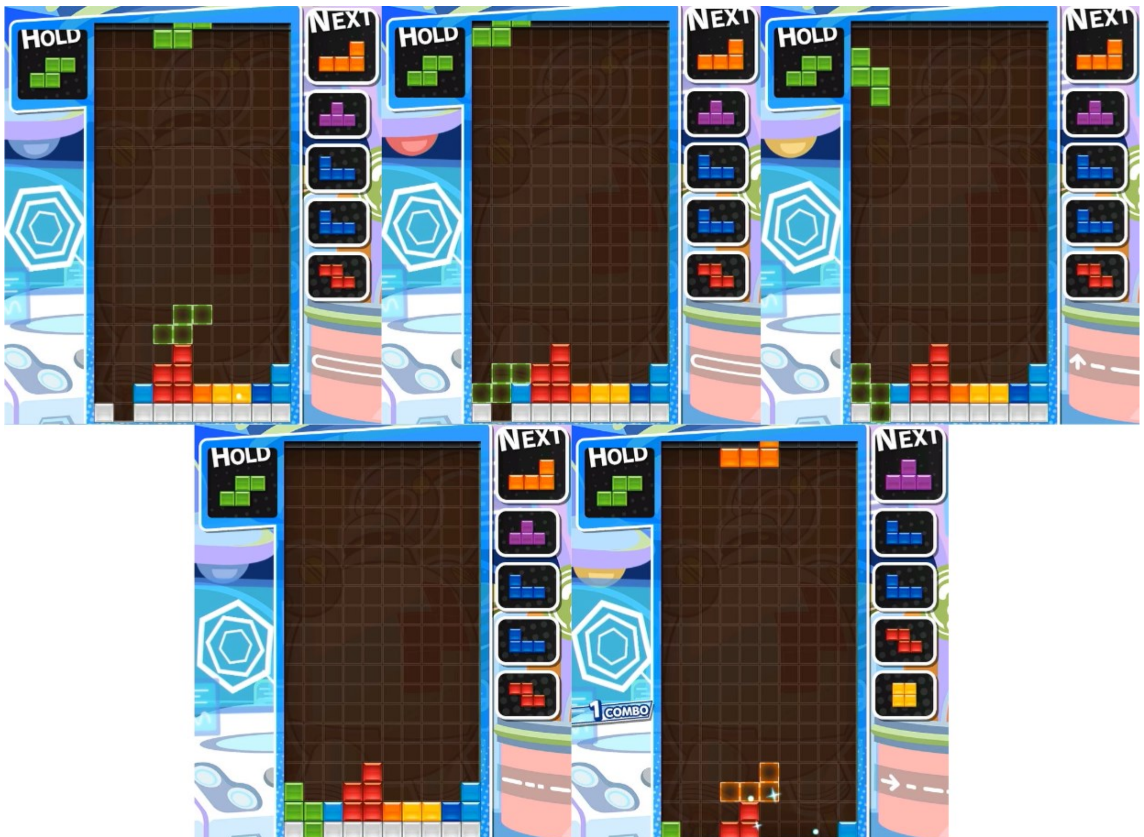
Kuvio 1. Alkuperäinen Tetris Electronika 60 -tietokoneella (Tetris.wiki 2015)

Pajitnovin inspiraationa Tetrikselle oli peli nimeltä ”pentominoes”, jossa yritetään asettaa kuvioita laatikon sisälle. Jokainen kuvio vie viiden neliön verran tilaa. Pajitnov kuvitteli näiden kuvioiden tippuvan ylhäältä lasiin ja pelaajan ohjaavan niitä. Pajitnov kehitti pelin vapaa-ajallaan ja kutsui sitä Tetrikseksi. Tetriksen nimi juontuu latinan sanasta tetra, joka tarkoittaa neljää, sekä tenniksestä, joka on Pajitnovin suosikkipeli. Neljä viittaa tetriksessä pelattaviin paloihin, joita sanotaan tetrominoiksi. (Weisberger 2016.) Tetrominoja on seitsemänlaista, joista jokainen vie neljä ruutua tilaa. (Kuvio 2.)



Kuvio 2. Tetriksessä käytetyt palat eli tetrominot (Tetris.wiki 2019)

Ruudun yläosasta tippuu tetrominoja, joita pelaajan kuuluu pyörittää ja siirtää ennen niiden laskeutumista. Jos pelaaja onnistuu asettamaan tetrominoja täytettyihin riveihin, sen rivin solut tyhjenevät. Peli loppuu, kun tetrominot yltävät ruudun yläosaan. (Kent 2001.) (Kuvio 3.)



Kuvio 3. Tetriksen eteneminen, pelaaja asettaa tetrominon ja tyhjetää kaksi riviä (IGN 2017)

Tetriksen lisensioinnista vastaa The Tetris Company. Lisenssin saamiseksi pelin tulee noudattaa tiettyjä sääntöjä, joita sanotaan Tetris Guidelineksi. Guideline määrittelee esimerkiksi, mikä nappi tekee mitäkin. Guideline standardisoi eri Tetris-pelit, joten pelaajat voivat aloittaa uuden Tetriksen pelaamisen tarvitsematta oppia mitään uutta. (Metts 2006)

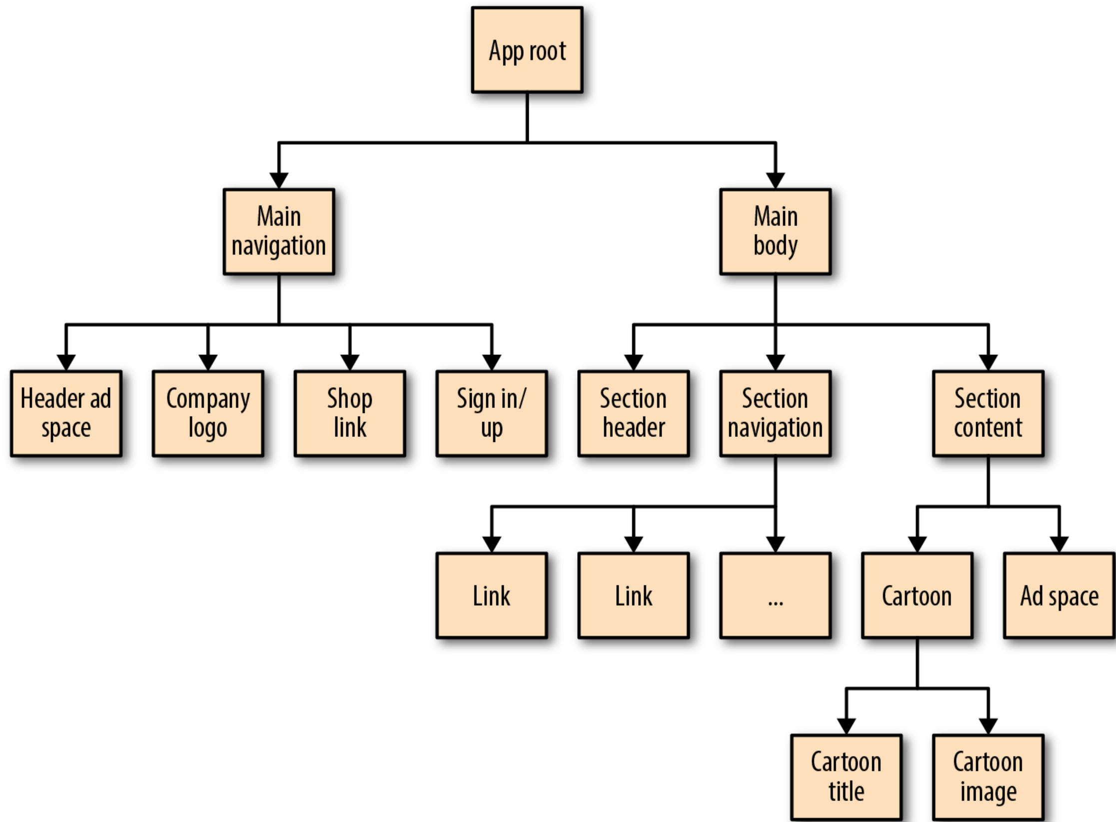
3 KÄYTETYT TEKNOLOGIAT

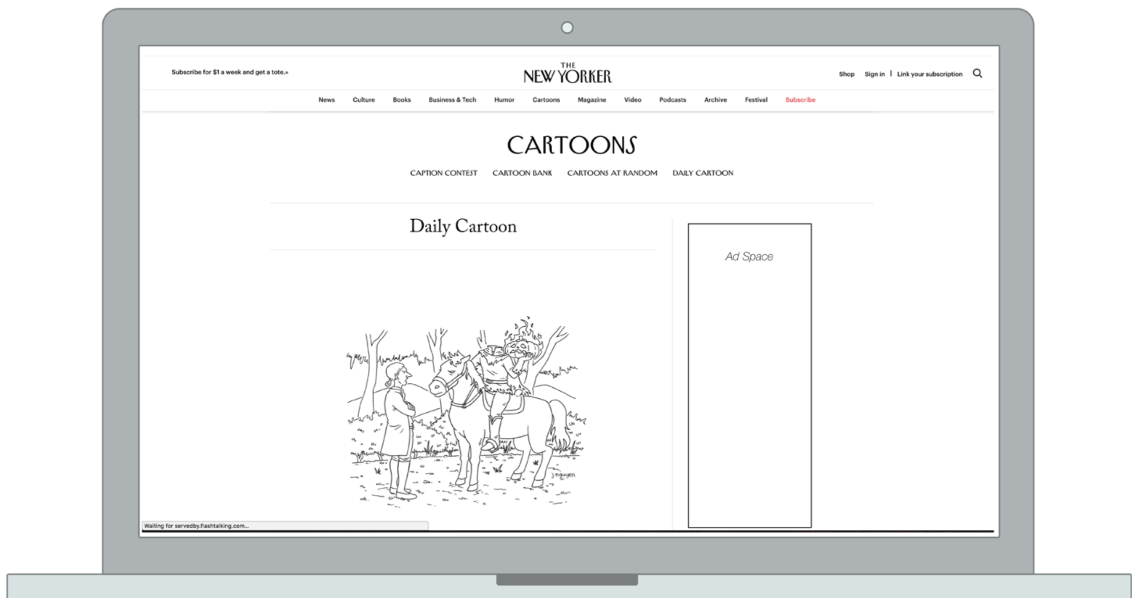
3.1 Reactin historia ja toiminta

React on Metan luoma Javascript-kirjasto, jolla rakennetaan UI:ssa käytettäviä komponentteja (Meta 2023). React on erittäin suosittu kirjasto, jota käyttää tuhannet yritykset, mukaan lukien Netflix ja Airbnb. Reactilla voi luoda käyttöliittymiä web-, mobiili- ja työpöytäsovelluksille. (Geekboots 2022.)

Ennen Reactia kompleksien web-käyttöliittymien kehitys oli vaikeaa. Muun muassa UI:n päivittämisen hallitseminen datan muuttuessa oli hankalaa. Facebook (nyk. Meta) korjasi tämän ongelman Reactilla. React osoittautui olevan todella nopea ja kehittäjäystävällinen. Reactin tehokkuus sekä yksinkertaisuus olivat osa hyötyjä, jotka tekivät Reactista niin suositun. (Mardan 2017, 5–6.)

React on käyttöliittymien kehittämiseen käytetty kirjasto, jolla luodaan komponentteja. Komponentti koostuu HTML:stä ja Javascriptistä, ja se pitää sisällään tarvittavan logiikan näyttääkseen osan laajemmasta UI:sta. Komponentteja voidaan käyttää yhdessä luodakseen monimutkaisia sovelluksen osia. Sovelluksen komponentit voidaan esittää komponenttipuuna. Esimerkiksi New Yorkerin sivulla on navigaatio-komponentti, joka pitää sisällään muun muassa kirjautumiskomponentin (kuvio 4).





Kuvio 4. Komponenttipuu esittää New Yorkerin sivua (Baer 2018)

Konkreettisena esimerkkinä yksinkertaisen tehtävälistan voi luoda kuvion 5 mukaan. Todo on funktio, joka palauttaa HTML:n li-elementin, joka pitää sisällään todoItem-objektin text-ominaisuuden. TodoList palauttaa ul-elementin, joka pitää sisällään taulukon Todoita. (Baer 2018.)

```
const Todo = ({ todoItem }) => (
  <li>{ todoItem.text }</li>
);
const TodoList = ({ todoListData }) => {
  const todoListItems = todoListData
    .map(todo => <Todo todo={todo} />);

  return <ul>{todoListItems}</ul>;
};
```

Kuvio 5. Todo- ja TodoList-funktiot Reactissa (Baer 2018)

Edellä selitetyt koodit sisältävät JSX:ää. JSX eli Javascript Syntax Extension on Javascriptin laajennus. Se helpottaa HTML:n käyttämistä Javasciptin kanssa React-sovelluksissa. JSX:llä voi esimerkiksi asettaa muuttujan arvoksi h1-elementin (kuvio 6). Koska Javascript ei ymmärrä JSX:ää, JSX-koodi pitää kääntää Javascriptiksi kääntäjällä. *Create-react-app*-paketti käyttää Babel-kääntäjää JSX:n kääntämiseksi automaattisesti. (Chavan 2021.)

```
const jsx = <h1>This is JSX</h1>
```

Kuvio 6. JSX-koodia Reactissa (Chavan 2021)

3.2 ASP.NET Core ja sen erot ASP.NETiin

ASP.NET Core on ohjelmistokehys, jolla rakennetaan moderneja sovelluksia, jotka ovat kykeneviä käyttämään internetiä ja pilveä. ASP.NET Corella voidaan rakentaa muun muassa web-sovelluksia ja palveluita, IoT-sovelluksia ja back-endejä mobiilisovelluksille. (Microsoft 2022.) ASP.NET Core on uudelleensuunniteltu ASP.NET 4.x-ohjelmistokehys. ASP.NETiin verrattuna ASP.NET Corella on useita parannuksia, muun muassa se on järjestelmäriippumaton, tehokkaampi ja tarjoaa parempaa tietoturvaa. (iFour Technolab 2019.)

ASP.NET Core perustuu vanhempaan ASP.NET-ohjelmistokehykseen. ASP.NET Core sisältää monia parannuksia ASP.NETiin verrattuna. Ensinnäkin ASP.NET tukee vain Windows-järjestelmiä, kun taas ASP.NET Core on järjestelmäriippumaton. Se toimii Windowsin lisäksi Linuxilla sekä MacOS:llä. Näin

ASP.NET Coresta tuli entistä käytettävämpi tiimeille, jotka eivät käytä Microsoftin tuotteita. (Ilyukha 2020.)

ASP.NET Core on tehokkaampi kuin ASP.NET. Tehokkuus parantaa käyttäjäkokemusta, sillä datan prosessointi vie vähemmän aikaa, mikä vähentää sivujen latausaikoja. Toiseksi tehokkuus tekee ASP.NET Coresta halvemman, sillä se vaatii vähemmän resursseja. Kustannuksiin vaikuttaa myös se, että ASP.NETiä suoritetaan tyypillisesti Windows-palvelimella, joka vaatii lisenssiä käyttäjältä. ASP.NET ei vaadi lisensiointia. (Ilyukha 2020.)

ASP.NET vanhempana ohjelmistokehyksenä tukee enemmän plugineita ja kirjastoja kuin ASP.NET Core. Suurempi määrä kehittäjiä on kokeneempia ASP.NETin käytössä kuin ASP.NET Coren, joka tekee tiimin kokoon saamisen helpommaksi ASP.NET:llä. (Ilyukha 2020.)

Uuden projektin aloittamisessa suositellaan ASP.NET Corea ASP.NETin sijaan, koska edellisten hyötyjen lisäksi Microsoft luultavasti lakkauttaa ASP.NETin kehittämisen jossain vaiheessa ja tukee pelkästään ASP.NET Corea. Tällöin vältytään siirtymästä ASP.NETistä Coreen. (Ilyukha 2020.)

4 LUOMANI TETRIS-PELIN TOTEUTUS

4.1 Front-end

4.1.1 Ympäristön perusedellytykset

React-sovelluksen luomiseen tarvitaan ensiksi ohjelmointiympäristö. Ohjelmistoympäristö on ohjelma, jota käytetään sovelluksien luomiseen. Se sisältää paljon ohjelmoinnissa käytettyjä työkaluja. (Redhat 2019.) Valitsin ohjelmistoympäristökseni Neovimin.

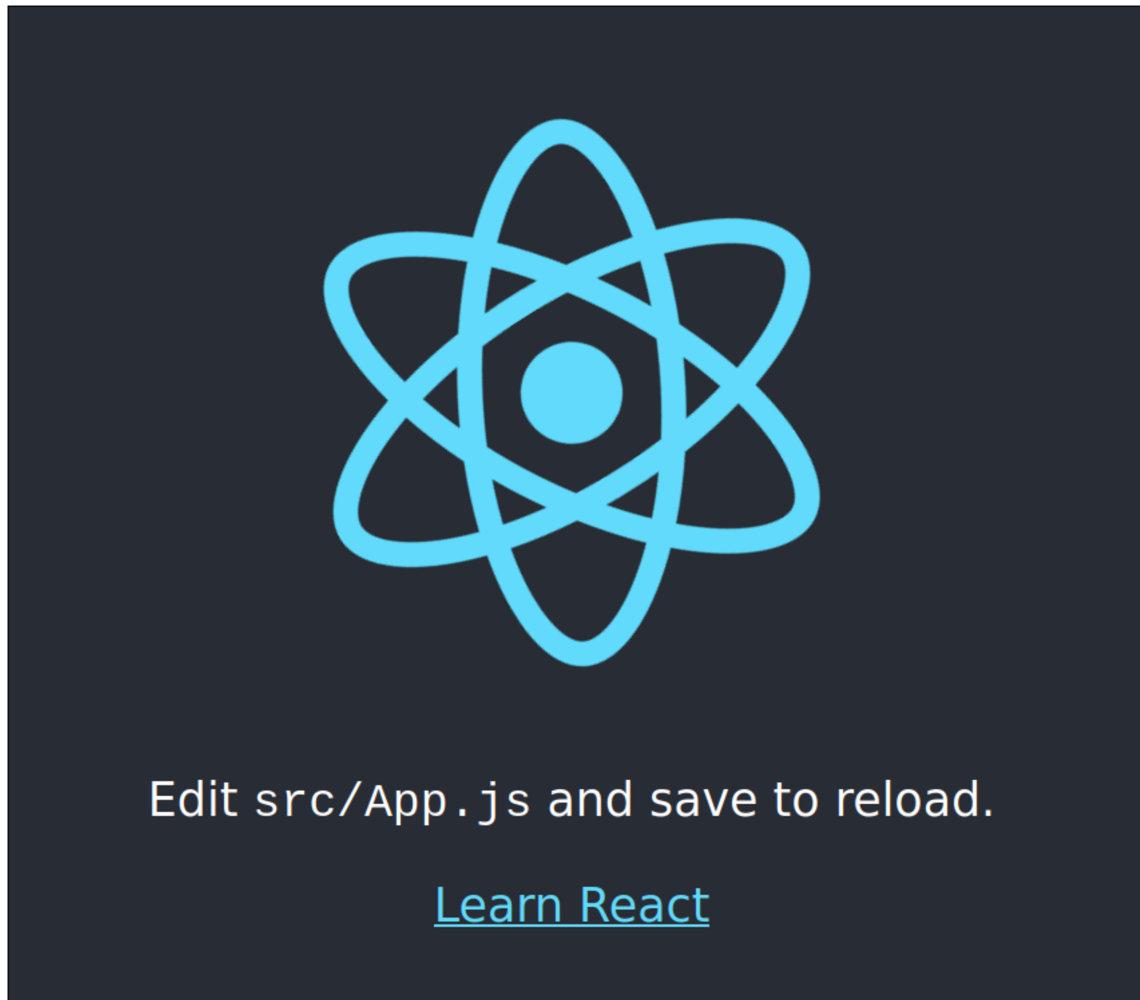
Ohjelmistoympäristön lisäksi tarvitaan Node.js ja npm. Node.js on ympäristö, jossa suoritetaan Javascript-koodia. Verkkoselaimissa on sisäänrakennettu Javascriptin suoritusympäristö, mutta selaimen ulkopuolella Javascriptin suoritukseen tarvitaan suoritusympäristö. (Semah 2022.) Npm on Node.js:n paketinhallintajärjestelmä. Npm koostuu CLI-työkalusta ja pakettivarastosta. Npm mahdollistaa Node.js-pakettien lataamisen ja levittämisen. (Nguyen 2020.)

Tämän luvun kuviot liittyvät pelini kehitykseen. Kuviot sisältävät pelini toimintaa, sen koodia, satunnaisia komentoja ja teknologioiden käyttöliittymiä.

4.1.2 Uuden projektin luominen

Uusi React-projekti luodaan create-react-app-paketilla. Create-react-app suoritetaan komennolla ”*npx create-react-app* projektin_nimi”. Npx asentaa create-react-app-paketin, Reactin ja tarvittavat riippuvuudet sekä luo uuden projektin.

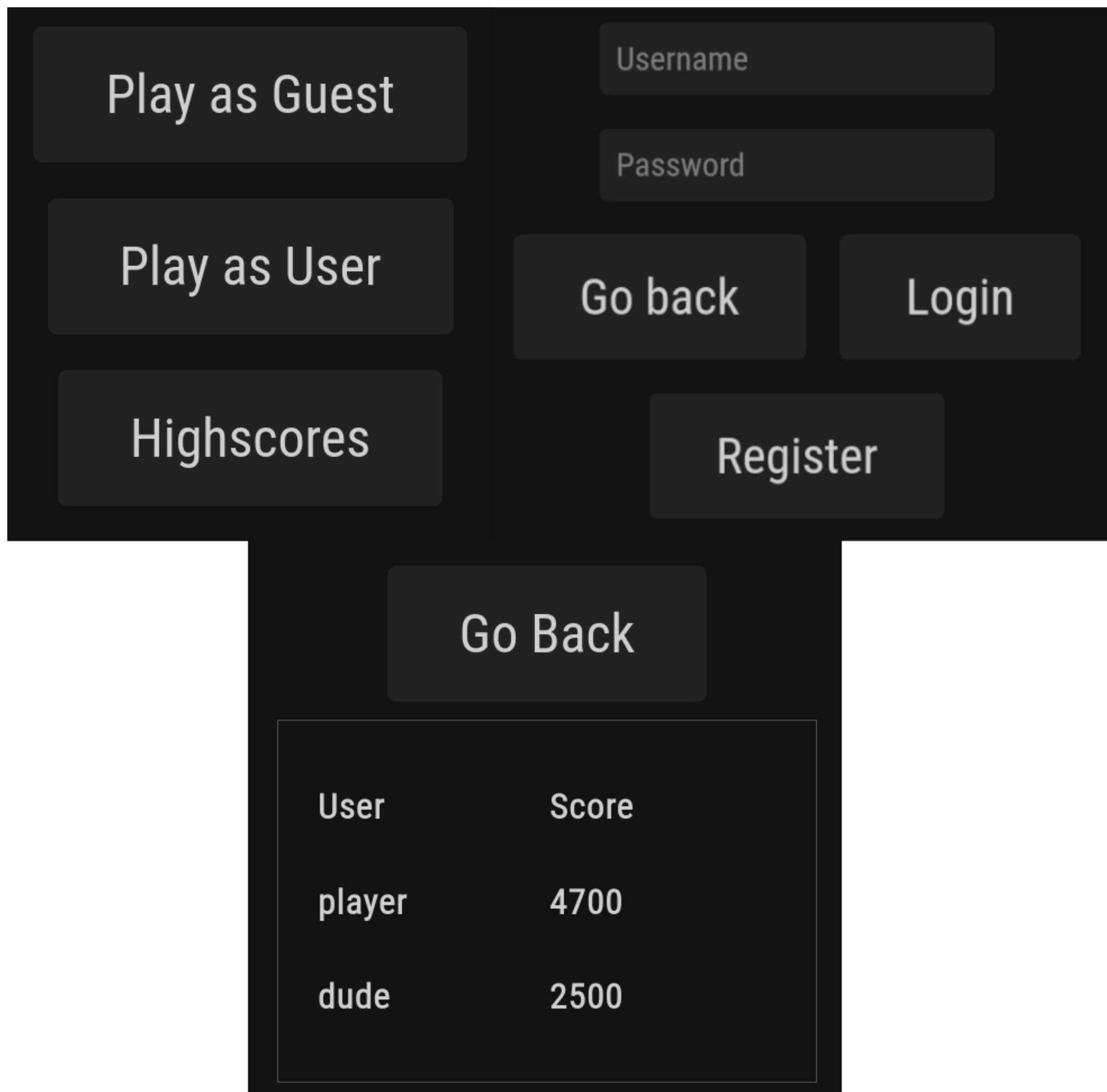
Projektin luotua npx kehottaa suorittamaan komento ”*npm start”* uuden projektin hakemistossa. ”*Npm start*” aloittaa paikallisen web-palvelimen, joka päivittyy automaattisesti, kun tekee muutoksia projektiin. Create-react-appin luoma sovellus aukeaa oletuksena osoitteessa http://localhost:3000. Kun linkin aukaisee selaimessa, create-react-appin luoma oletussivu aukeaa (kuvio 7).



Kuvio 7. React-sovelluksen oletusnäkymä

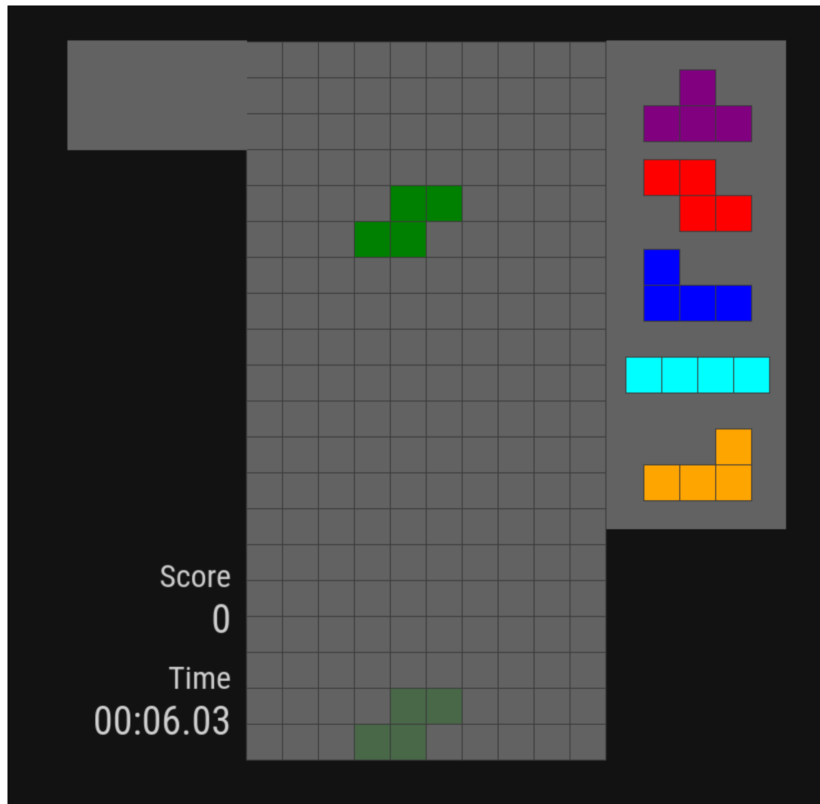
4.1.3 Valmiin pelin kulku

Pelin etusivulla on kolmea nappia. Kun käyttäjä painaa Play as Guest -nappia, peli alkaa suoraan kirjautumatta. Kun painaa Play as User -nappia, sivu pyytää käyttäjää kirjautumaan sisään, jonka jälkeen peli alkaa. Vaihtoehtoisesti käyttäjä voi rekisteröityä sivulle. Highscores-napista pääsee sivulle, jossa on listattuna jokaisen pelaajan huippupisteet. (Kuvio 8.)



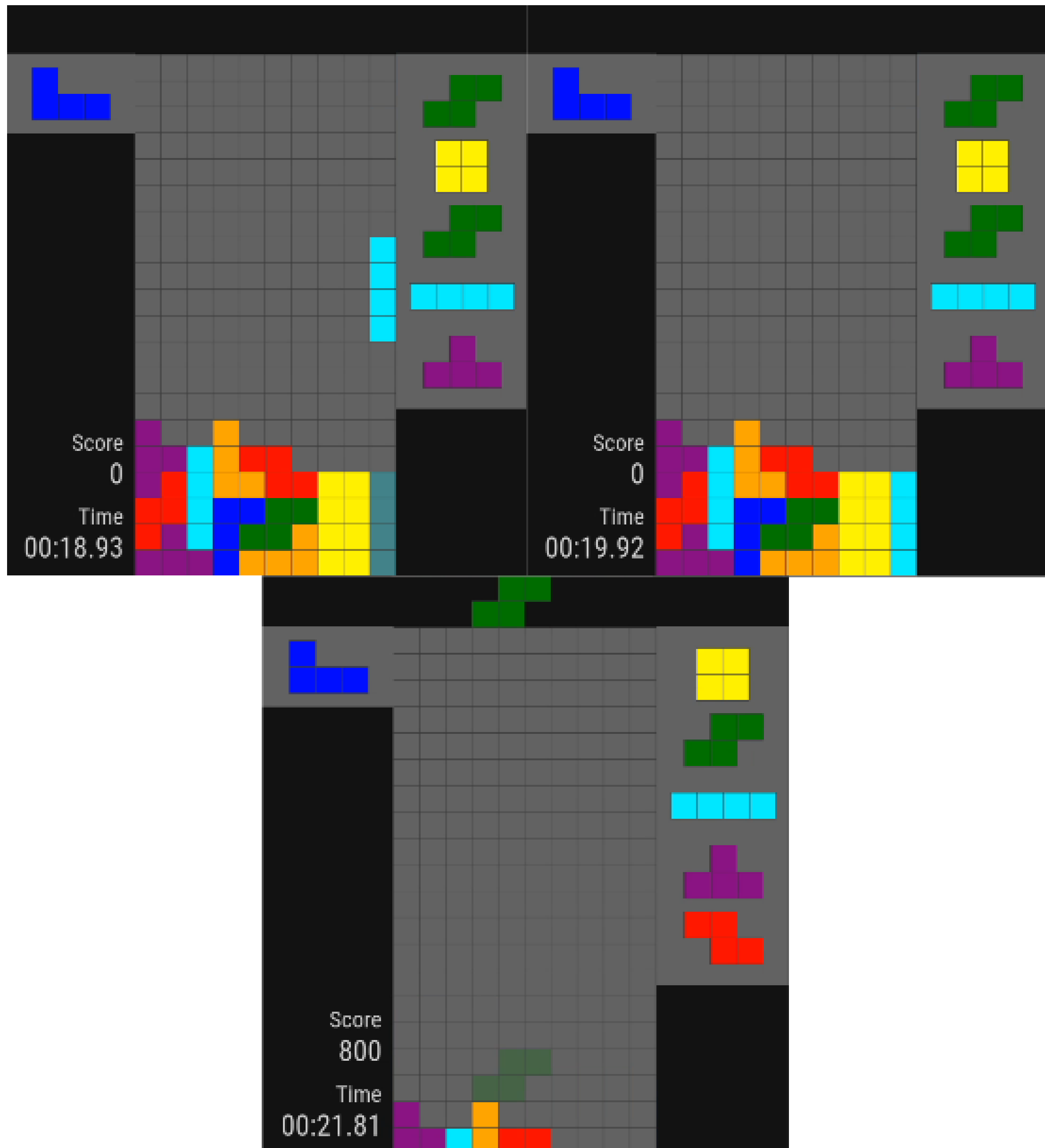
Kuvio 8. Etu- ja kirjautumissivu sekä huippupisteet-lista

Kun peli alkaa, pelikenttä ilmestyy. Kuviossa 9 pelaaja ohjaa S-tetrominoa, ja kentän pohjalla on tetrominon ”haamu” (engl. ghost piece), joka esittää, mihin tetromino tippuu. Kentän oikealla puolella on tetrominojen esikatselu, joka näyttää seuraavat tetrominot. Kun pelaaja pelaa S-tetrominon, seuraavana jonossa on T-tetromino. Vasemmalla puolella on pelaajan pisteet, kulunut aika sekä niin sanottu ”hold piece”, jonka pelaaja voi vaihtaa pelattavan tetrominon kanssa.



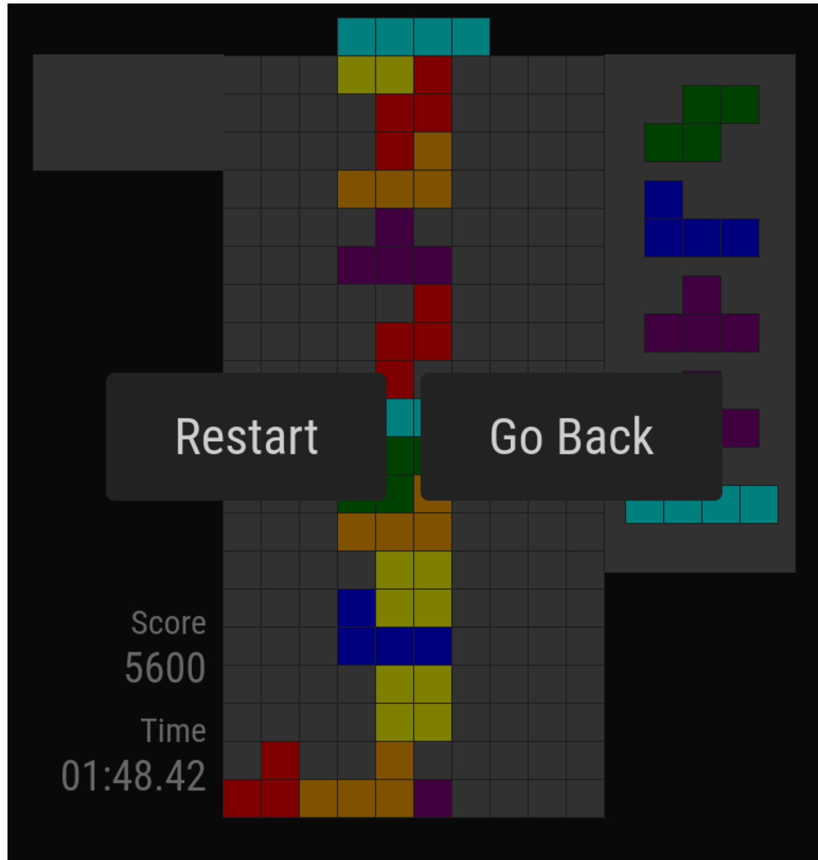
Kuvio 9. Pelikenttä

Kun pelaaja täyttää rivejä, hän saa pisteitä ja täytetyt rivit tyhjenevät. Pelaaja on jättänyt yhden sarakkeen välin, johon hän aikoo tiputtaa I-tetrominon. Tetromino lukkiutuu, jonka jälkeen hän saa 200 pistettä per tyhjennetty rivi, eli yhteensä 800 pistettä. Lisäksi rivit tyhjennetään. (Kuvio 10.)



Kuvio 10. Rivien tyhjentäminen I-tetrominolla

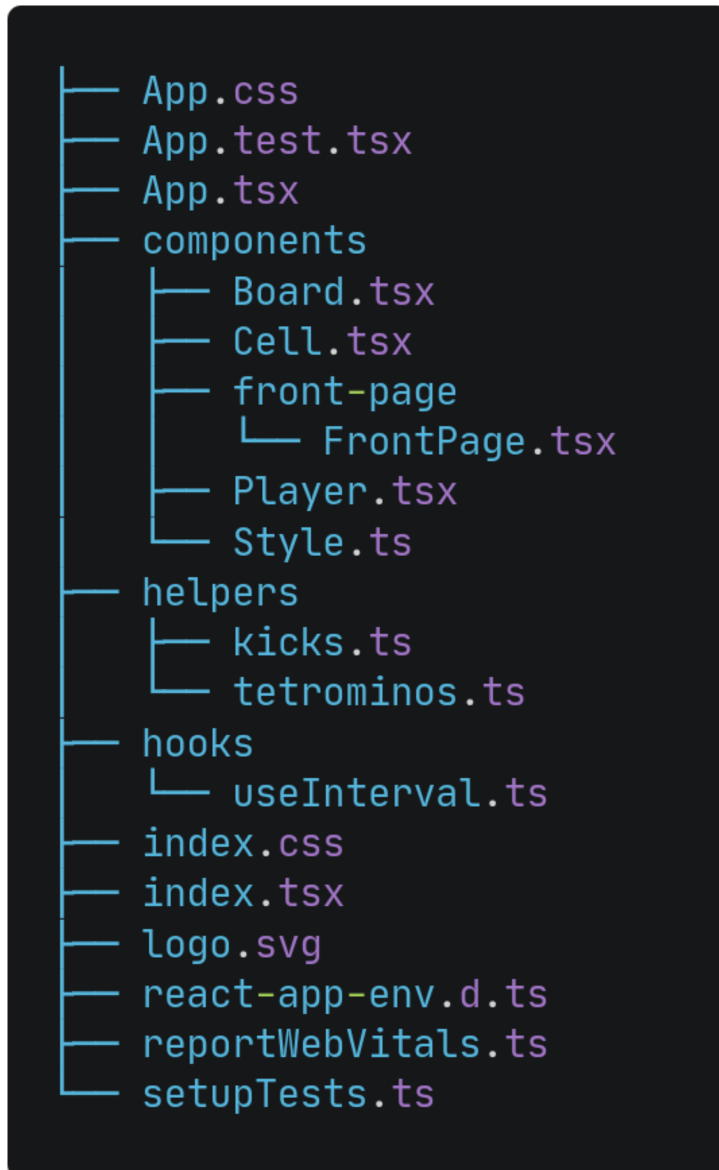
Jos pelaaja pinoaa tetrominoja pelikentän yläreunan yli, peli päättyy (kuvio 11). Samalla pelaajan pisteet tallennetaan tietokantaan. Pelin loputtua pelaaja voi joko aloittaa alusta tai mennä takaisin etusivulle. Jos pelaajan pisteet on hänen uudet huippupisteensä, se näytetään Highscores-sivulla.



Kuvio 11. Pelaaja pinoaa tetrominoja pelikentän yli ja peli loppuu

4.1.4 Front-endin rakenne

React-sovelluksen juurihakemistossa on useita tiedostoja ja kansioita, mutta suurin osa sovelluksen toiminnasta tapahtuu src-kansiossa, joten keskityn tässä luvussa vain sen esittelemiseen. Src-kansion hakemistorakenne sisältää components-, helpers- ja hooks-hakemistot, joissa on suurin osa ohjelman koodista. Muut tiedostot ovat create-react-app-paketin luomia eikä erityisen tärkeitä, joten jätän ne välistä. Tärkeimmät tiedostot ovat siis index.tsx ja App.tsx sekä tiedostot components-, helpers- ja hooks-hakemistoissa. (Kuvio 12.)



Kuvio 12. Src-kansion hakemistorakenne

Sovelluksen suoritus alkaa index.tsx-tiedostosta. Tiedoston alussa tuodaan tiedostoja käyttöön import-lausekkeella. Kun tiedostot on tuotu esiin, voidaan niiden ominaisuuksia käyttää nykyisessä tiedostossa. Import-lausekkeiden jälkeen käytetään ReactDOM-paketin createRoot-funktiota Reactin root-objektin luomiseen. Root-objektilla on JSX:n renderöimiseen tarkoitettu render-funktio, jota käytetään App-komponentin renderöimiseen. (Kuvio 13.)

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import "@fontsource/roboto-condensed";

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(<App />);
```

Kuvio 13. Index.tsx-tiedosto, josta sovelluksen suoritus alkaa

App-komponentti on määritetty omassa tiedostossaan. App palauttaa FrontPage-komponentin. Komponentin tiedoston lopussa viedään App-komponentti export-lauseella. Komponentin vieminen mahdollistaa sen tuomisen kuviossa 13. Appin palauttama FrontPage-komponentti suorittaa renderPage-funktion. RenderPage palauttaa komponentin, joka lopuksi renderöidään root-elementissä (kuvio 14).

```
function renderPage() {
  switch (page) {
    case Pages.FrontPage:
      return renderFront();

    case Pages.Login:
      return renderLogin();

    case Pages.Game:
      return <Board setPage={setPage} />;

    case Pages.Highscores:
      const jsx = (
        <>
        ...
```

Kuvio 14. RenderPage-funktio määrittää, mikä komponentti renderöidään

RenderPage sisältää switch-lausekkeen, jolla käydään läpi page-muuttujan mahdollisia arvoja. Page-muuttujan tyypiksi on määrätty Pages-enum. Enum on Typescriptin tietotyyppi, jolle on määrätty arvojoukko. Pages pitää sisällään kaikki sivut, joilla käyttäjä voi käydä, eli FrontPage, Login, Game ja Highscores. Kun käyttäjä avaa sovelluksen, pagen arvo on Pages.FrontPage. Tässä tapauksessa renderPage palauttaa renderFront-funktion arvon. RenderFront renderöi sitten etusivun. Etusivun nappia painamalla voi vaihtaa page-muuttujan arvon, jolloin renderPage vaihtaa renderöidyn komponentin (kuvio 15).

```
function renderFront() {
  return (
    <>
      <StyledButton onClick={() => handlePlayGuest()}>
        Play as Guest
      </StyledButton>
      <StyledButton onClick={() => setPage(Pages.Login)}>
        Play as User
      </StyledButton>
      <StyledButton onClick={() => setPage(Pages.Highscores)}>
        Highscores
      </StyledButton>
    </>
  );
}
```

Kuvio 15. RenderFront-funktio renderöi etusivun

RenderFront-funktiossa on käytetty StyledButton-komponentteja. StyledButton on tyylitetty komponentti, eli se pitää sisällään CSS:ää. StyledButton on luotu styled-components-kirjastolla. Styled-components helpottaa komponenttien tyylittämistä esimerkiksi lisäämällä kriittisiä tyylejä automaattisesti. (Kuvio 16.)

```
export const StyledButton = styled.button`
  background-color: ${Colorscheme.c900};
  color: ${Colorscheme.c200};
  padding: 2rem 4rem;
  font-size: 3rem;
  border: none;
  border-radius: 0.5rem;
  margin: 1rem;
  &:hover {
    background-color: ${Colorscheme.c800};
    cursor: pointer;
  }
`;
```

Kuvio 16. StyledButton-komponentti pitää sisällään CSS-tyyliasuja

FrontPage-komponentin alussa luodaan usernameRef- ja passwordRef-objekti Reactin useRef-koukulla. UseRef-koukkua voidaan käyttää DOM-elementin referenssin tallentamiseen. Nämä objektit asetetaan DOM-elementin ref-ominaisuuden arvoksi. (Kuvio 17.)

```
const usernameRef = useRef<HTMLInputElement>(null);
const passwordRef = useRef<HTMLInputElement>(null);
...
<StyledInput placeholder="Username" ref={usernameRef} />
<StyledInput placeholder="Password" ref={passwordRef} />
```

Kuvio 17. DOM-elementin referenssin tallentaminen ja asettaminen DOM-elementtiin

Referenssejä käytetään käyttäjän sisäänkirjautumisessa handleLogin-funktiossa. Koska referenssit on määrätty input-elementteihin, niiltä voi saada input-elementtien arvot current.value-ominaisuudella (kuvio 18). Kirjautumista varten luodaan reqOpts-objekti, johon tallennetaan API-kutsua varten tarvittavat tiedot. API-päätepiste käyttää POST-metodia, joten se tallennetaan objektiin. Kutsussa pitää myös antaa lähetetyn datan tyyppi Content-Type-otsikolla. Content-Type on application/json ja objektin body-ominaisuudeksi asetetaan kirjautumistiedot JSON-muodossa. Fetch-metodilla tehdään kutsu /User/login-päätepisteeseen reqOpts-objektin kanssa. Jos kirjautuminen onnistuu, peli alkaa.

```
function handleLogin() {
  if (!usernameRef.current?.value || !passwordRef.current?.value) {
    alert("Need username and password");
    return;
  }

  const username = usernameRef.current.value;
  const password = passwordRef.current.value;
  const reqOpts = {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ username: username, hashedPassword: password }),
  };

  fetch("http://localhost:5267/User/login", reqOpts).then(res => {
    if (res.ok) {
      setCredentials();
      setLogged(true);
      setPage(Pages.Game);
    } else alert("Couldn't log in");
  });
}
```

Kuvio 18. HandleLogin-funktio kirjaa pelaajan sisään ja aloittaa pelin

Peli alkaa, kun page-muuttujan arvo on Pages.Game. Sovellus renderöi silloin Board-komponentin. Pelikenttä koostuu pääosin board-objektista ja pelaajasta. Board-objekti edustaa pelialuetta, joten se pitää sisällään jokaisen pelikentän solun. Board on kaksiulotteinen taulukko, joka alustetaan Array.from- ja Array.fill-metodeilla sekä Array-konstruktorilla. (Kuvio 19.)

```
const [board, setBoard] = useState(
  Array.from(Array(boardSize.height), () =>
    Array.from(
      Array<CellData>(boardSize.width).fill({
        filled: false,
        color: Colorscheme.c600,
        transparent: false,
      })
    )
  )
);
```

Kuvio 19. Board-muuttujan alustaminen

Board-objektin solut muutetaan JSX:ksi Array.map-metodilla. Map-metodi suorittaa annetun funktion taulukon jokaiselle elementille ja palauttaa uuden taulukon muunnetuilla elementeillä. Board-objektille suoritetaan map-metodi, jonka jokaiselle elementille, eli tässä tapauksessa pelikentän riville (row), suoritetaan taas map-metodi. Näin pääsee käsiksi pelikentän solujen (cell) dataan, joita käytetään pelikentän JSX:n luomisessa. (Kuvio 20.)

```
<StyledBoard
  width={boardSize.width}
  ref={divRef}
  tabIndex={0}
  style={{ marginTop: `-${cellWidth * 2}px` }}
  onKeyUp={e => stopRepeat(e.code)}
  onKeyDown={e => move(e)}>
  {board.map((row, y) => {
    return row.map((cell, key) => {
      return (
        <Cell
          cellColor={cell.color}
          width={cellWidth}
          transparent={cell.transparent}
          key={key}
          style={
            cell.color !== Colorscheme.c600 &&
            cell.color !== Colorscheme.cbase
              ? {
                  border: `1px solid ${Colorscheme.c800}`,
                  width: cellWidth + 2 + "px",
                  marginTop: "-1px",
                  marginLeft: "-1px",
                  boxSizing: "border-box",
                  zIndex: 10,
                }
              : {
                  border:
                    y >= 2
                      ? `1px solid ${Colorscheme.c800}`
                      : `1px solid ${Colorscheme.cbase}`,
                  marginTop: "-1px",
                  marginLeft: "-1px",
                }
          }
        />
      );
    });
  })}
</StyledBoard>
```

Kuvio 20. Pelikentän JSX, joka renderöidään pelaajalle

Board-objektin solut muuntuvat, kun pelaaja tekee liikkeitä. Pelaajan tekemät liikkeet vaikuttavat player-objektiin, joka kuvaa pelaajan pelaamaa tetrominoa.

Player sisältää pos-objektin, joka pitää sisällään pelaajan x- ja y-koordinaatit. Playerissä on myös tetromino-objekti, jossa määritellään tetrominon muoto, tetrominon täyttävät solut määrittelevä ruudukko, tetrominon väri sekä tetrominon suunta. Playerissä on lopuksi merged-ominaisuus, joka määrää, onko pelaaja pelannut tetrominon. (Kuvio 21.)

```
const [player, setPlayer] = useState({
  pos: { x: 0, y: 0 },
  tetromino: {
    shape: key,
    grid: tetromino.grid,
    color: tetromino.color,
    orientation: 0,
  },
  merged: false,
});
```

Kuvio 21. Player-objektin alustaminen

Player tulee esiin pelikentälle, kun se kopioidaan boardiin. Playerin tetromino-objektin grid-taulukolla suoritetaan forEach-silmukka. Silmukka käy läpi taulukon jokaisen rivin, joilla taas suoritetaan toinen forEach-silmukka. Kaksi silmukkaa mahdollistaa funktion suorittamisen jokaiselle solulle tetrominon taulukossa. Aluksi tarkistetaan, jos solun arvo on epätosi. Epätosi tarkoittaa, että solu ei kuulu tetrominoon. Tällöin jätetään funktio väliin ja siirrytään seuraavan soluun. Seuraavaksi asetetaan pelitila lopetettavaksi, jos pelaaja on kasannut tetrominoja pelikentän yli. Lopuksi pelikentän soluun, jossa pelaajan käsiteltävä solu on, kopioidaan käsiteltävän solun arvot. (Kuvio 22.)

```
player.tetromino.grid.forEach((row, y) =>
  row.forEach((cell, x) => {
    if (!cell) return;

    if (player.pos.y + y < 2 && player.merged) setGameState(GameState.Ended);
    board[player.pos.y + y][player.pos.x + x] = {
      color: player.tetromino.color,
      filled: player.merged,
      transparent: false,
    };
  })
);
```

Kuvio 22. Playerin kopioiminen boardiin

4.2 Back-end

4.2.1 Ympäristön perusedellytykset

.NET-sovelluksien kehittämiseen tarvitaan .NET SDK. .NET SDK sisältää ASP.NET Coren, johon back-end perustuu. .NET SDK:n asennukseen löytyy ohjeet alustasta riippuen sivulta https://dotnet.microsoft.com/en-us/download/dotnet. .NET SDK:n asennuksen jälkeen voi käyttää dotnet-ohjelmaa tarvittavien pakettien asentamiseen.

.NET-sovelluksissa, joiden pitää hakea ja manipuloida dataa, käytetään usein Entity Framework Core -ohjelmistokehystä. Entity Framework Core on object-database mapper, joka helpottaa tietokannan kanssa työskentelyä. Tässä projektissa on tietokantana käytössä PostgreSQL. Projektiin lisätään EF Core sekä PostgreSQL-tuki EF Corelle. Tarvittavat paketit näille on *Npgsql.EntityFrameworkCore.PostgreSQL*, *Microsoft.EntityFrameworkCore* ja *Microsoft.EntityFrameworkCore.Design*, ja ne asennetaan komennolla ”*dotnet add package* paketin_nimi”. Lisäksi itse PostgreSQL tulee asentaa.

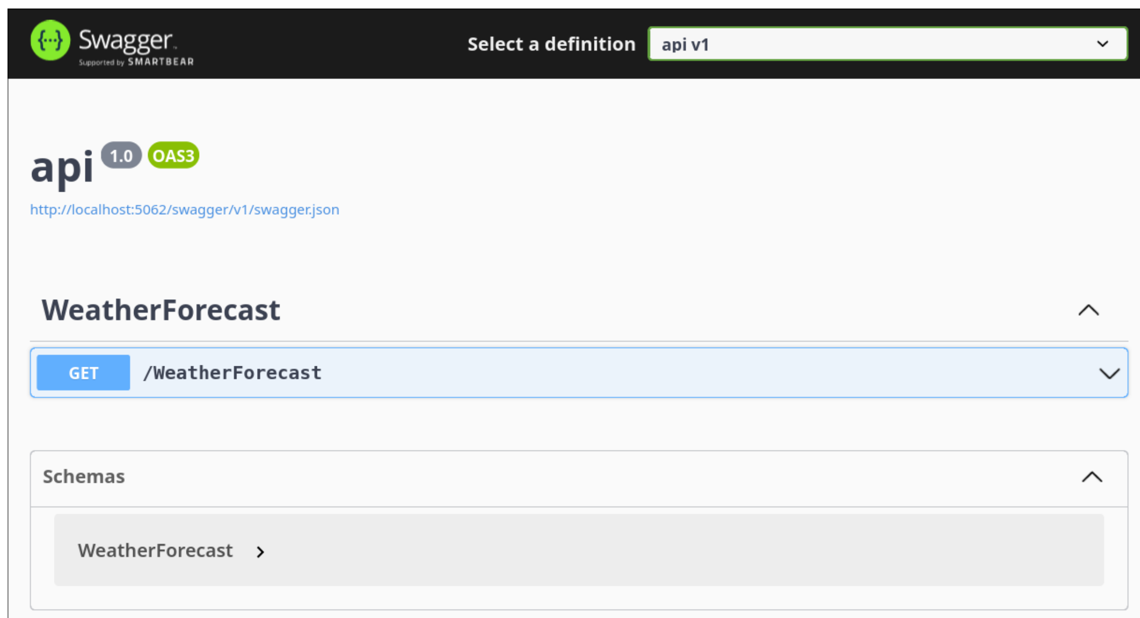
4.2.2 API-projektin luominen

Pelin back-endin luominen on helpompi aloittaa käyttämällä mallia, joka sisältää yksinkertaisen projektin. Tästä on etenkin hyötyä, jos ASP.NET Core on uusi teknologia käyttäjälle, koska malli sisältää monia minimaalisia toiminnallisuuksia,

joita ASP.NET Core tarjoaa. Näitä on helpompi opetella käyttämään minimaalisessa projektissa. Toiseksi projektia voi käyttää lopullisen back-endin työn aloittamisessa, koska se sisältää yksinkertaisimmat asiat, joita tarvitaan työssä. Tämä luku selittää mallilla luodun projektin kehittämistä.

.NET SDK:n asennuksen jälkeen voi käyttää dotnet-ohjelmaa. Minimaalisen api-projektiin voi käyttää webapi-mallia komennolla ”*dotnet new webapi -o* projektin_nimi”. Projekti sisältää yksinkertaisen API:n, jolla voi hakea säätietodataa. Ohjelma suoritetaan komennolla ”*dotnet run”.*

Kun ohjelma on käynnissä, API:ta voi käydä tutkimassa Swaggerissa osoitteessa http://localhost:5062/swagger/index.html. Swagger on API:en kehitykseen ja dokumentointiin tarkoitettu työkalu. Swagger näyttää kaikki API-päätepisteet ja mahdollistaa kutsujen suorittamista niihin. Sovellus sisältää oletuksena vain yhden päätepisteen, joka palauttaa säätietodataa. (Kuvio 23.)



Kuvio 23. Dotnetin webapi-mallin luoma API Swaggerissa

Projektissa käytetty säätiedon datamalli on WeatherForecast.cs-tiedostossa. Malli sisältää päivämäärän, lämpötilan sekä celcius- että fahrenheit-asteina ja tekstikentän tiivistelmälle. Malliin voidaan lisätä tässä vaiheessa Id-kenttä, joka toimii tietokannan pääavaimena (engl. primary key). (Kuvio 24.)

```
public class WeatherForecast
{
    public int Id { get; set; }
    public DateOnly Date { get; set; }
    public int TemperatureC { get; set; }
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
    public string? Summary { get; set; }
}
```

Kuvio 24. WeatherForecast-datamalli, johon on lisätty Id-kenttä

ASP.NET Coren controllerien tarkoitus on käsitellä tulevat HTTP-pyynnöt ja palauttaa vastauksia. Tässä sovelluksessa pyyntöjä käsittelee WeatherForecastController. Tässä on määritetty kuvion 23 päätepiste. Päätepiste valitsee numeron yhden ja viiden väliltä ja palauttaa sen verran WeatherForecast-objekteja GET-metodilla. (Kuvio 25.) Päätepistettä voi testata esimerkiksi cURL-ohjelmalla komennolla ”*curl –X ’GET’ ’http://localhost:5062/WeatherForecast’”.* Päätepiste palauttaa vastauksen JSON-objektina, joka voi olla esimerkiksi ”*[{"id":0,"date":"2024-04-12","temperatureC":23...]*”.

```
using Microsoft.AspNetCore.Mvc;

namespace api.Controllers;

[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy",
"Hot", "Sweltering", "Scorching"
    };

    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
        _logger = logger;
    }

    [HttpGet(Name = "GetWeatherForecast")]
    public IEnumerable<WeatherForecast> Get()
    {
        return Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = Summaries[Random.Shared.Next(Summaries.Length)]
        })
        .ToArray();
    }
}
```

Kuvio 25. WeatherForecastin controller, joka palauttaa säätietodataa

API ei käytä vielä tietokantaa, joten sille pitää lisätä tarvittavat asiat ensin. Projektiin pitää lisätä konteksti tietokantaa varten DbContextilla. DbContext on olennainen osa EF Corea. Se vastaa yhteyden luomisesta sovelluksen ja tietokannan välillä, ja näin mahdollistaa mm. kyselyiden suorittamisen tietokannassa. WeatherContext-luokka peritään DbContext-luokasta. Tietokannan Weathers-taulu lisätään WeatherContextiin DbSet-tyyppisellä ominaisuudella, jolloin tauluun pääse käsiksi WeatherContextin kautta. Konteksti pitää myös rekisteröidä, jotta se tulee käyttöön. Rekisteröinti tapahtuu Program.cs-tiedostossa AddDbContext-metodilla. (Kuvio 26).

```
// WeatherForecast.cs

public class WeatherContext : DbContext
{
    public WeatherContext(DbContextOptions<WeatherContext> options) :
base(options) { }

    public DbSet<WeatherForecast> Weathers => Set<WeatherForecast>();
}

// Program.cs

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<WeatherContext>(options =>
{
    options
        .UseNpgsql(

 builder.Configuration.GetConnectionString("DefaultConnection"));
});
```

Kuvio 26. DbContextin lisääminen ja sen rekisteröiminen

WebApplicationBuilderin Services-kokoelmaan (engl. collection) lisätään edellä luotu WeatherContext. UseNpgsql-metodi konfiguroi PostgreSQL-tietokantaan kontekstilla. UseNpgsql vaatii merkkijonon (engl. connection string), jossa on tarvittavat tiedot tietokantaan yhdistämiseen. Connection string tallennetaan appsettings.json-tiedostoon (kuvio 27), josta se ladataan UseNpgsql:lle.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=reactris;UserId=user;
Password=pass;Include Error Detail=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Kuvio 27. Connection stringin lisääminen appsettings.json-tiedostoon

DbContext pitää lisätä vielä WeatherForecast-luokan controlleriin, jotta sitä voi käyttää. Controlleriin luodaan context-kenttä, ja asetetaan sen arvo konstruktorissa (kuvio 28). EF Core injektoi WeatherContextin konstruktoriin, koska aiemmin rekisteröitiin WeatherContextin palvelu.

```
private readonly ILogger<WeatherForecastController> _logger;
private readonly WeatherContext context;

public WeatherForecastController(ILogger<WeatherForecastController>
logger, WeatherContext context)
{
    _logger = logger;
    this.context = context;
}
```

Kuvio 28. DbContextin lisääminen controlleriin

Nyt DbContextia voi käyttää, joten luodaan uusi API-päätepiste, joka tallentaa tietoa tietokantaan. Päätepisteellä on oma reittinsä (engl. route), SaveWeatherForecast, jossa siihen tehdään pyyntö. Reittien asettamisessa käytetään Route-attribuuttia. Attribuutit mahdollistavat metadatan lisäämisen koodiin. Attribuutit määritellään neliösulkujen sisällä. Uudella päätepisteellä on toinen attribuutti, HttpPost, joka määrittää päätepisteen HTTP-metodin. Päätepiste kääntää objektin pyynnön rungosta (engl. body) WeatherForecast-tyyppiseksi. Objekti lisätään tietokantaan DbContextin Add-metodilla, ja muutokset tallennetaan SaveChanges-metodilla. (Kuvio 29.)

```
[Route("SaveWeatherForecast")]
[HttpPost]
public IActionResult SaveWeatherForecast([FromBody] WeatherForecast
forecast)
{
    this.context.Add(forecast);
    this.context.SaveChanges();
    return Ok();
}
```

Kuvio 29. API-päätepiste, joka tallentaa säätiedot tietokantaan

Lopuksi EF Corella pitää suorittaa migraatio ja päivittää tietokanta. Migraatio mahdollistaa tietokantamallin inkrementaalisen päivityksen. Tietokantamalli voidaan päivittää ja palauttaa tiettyyn migraatioon säilyttäen tietokannassa olevan datan. Koska tietokantaa ei ole vielä olemassa, migraatio ja päivitys tulee suorittaa. Migraatio tapahtuu EF Coren *migrations*-työkalulla komennolla ”*dotnet ef migrations add* migraation_nimi”. Tietokanta päivitetään ”*dotnet ef database update*”-komennolla. EF Core luo migraation lopuksi Migrations-hakemiston projektiin. Täällä on tiedostot, jotka sisältävät migraatioihin liittyvät tiedot. (Kuvio 30.)

```
Migrations
├── 20231109031440_InitialMigration.cs
├── 20231109031440_InitialMigration.Designer.cs
└── WeatherContextModelSnapshot.cs
```

Kuvio 30. Migraation luodut tiedostot

4.2.3 Back-endin rakenne

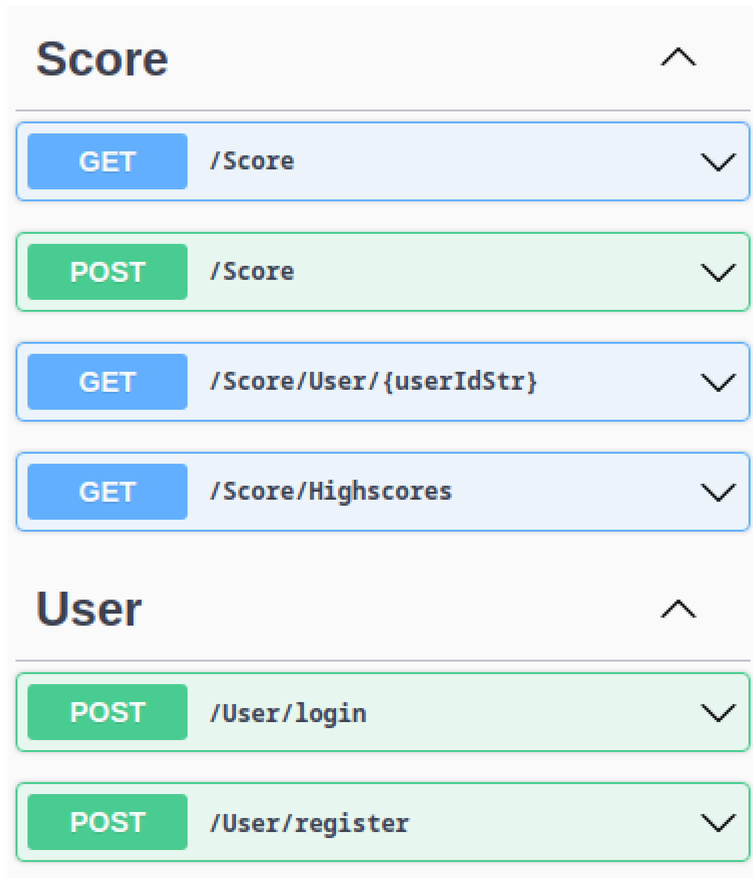
Back-endissä käytettävä data koostuu käyttäjien kirjautumistiedoista (User) sekä pisteistä (Score) (kuvio 31). Score-malli sisältää kentät UserId ja User. Tämä muodostaa suhteen (engl. relationship) User- ja Score-mallien välille, jolloin Score-malliin tallennetun käyttäjän tietoja voidaan tarkastella ja käyttää Score-mallin objektilla (esim. score.User.Username).

```
public class User
{
    public int Id { get; set; }
    public Guid UserId { get; set; }
    public string? Username { get; set; }
    public string? Password { get; set; }
}

public class Score
{
    public int Id { get; set; }
    public Guid ScoreId { get; set; }
    public int UserId { get; set; }
    public User? User { get; set; }
    public int PlayerScore { get; set; }
}
```

Kuvio 31. Back-endissä käytetyt datamallit

Back-endissä on käytössä kuusi API-päätepistettä. /Score-päätepistettä käytetään pisteiden hakemiseen GET-metodilla ja pisteiden tallentamiseen POST-metodilla. Käyttäjän pisteet saadaan /Score/User/{userIdStr}-päätepisteellä lisäämällä käyttäjänimi päätepisteen loppuun. Huippupisteet saadaan /Score/Highscores-päätepisteellä. Käyttäjän kirjautumiseen käytetään /User/login-päätepistettä ja rekisteröintiin /User/register-päätepistettä. (Kuvio 32.)



Kuvio 32. Back-endin API-päätepisteet Swaggerissä

Koska jotkin API-päätepisteet kuuluvat toimia vain kirjautuneille käyttäjille, pitää back-endiin lisätä valtuutus (engl. authorization). Projektissa käytetään Basic Authenticationia käyttäjän todentamisessa (engl. authentication). Basic Authenticationissa käyttäjän kirjautumistiedot lähetetään pyynnön mukana base64-muodossa. Tämä on erittäin yksinkertainen tapa todentaa käyttäjä, minkä vuoksi päätin käyttää sitä.

Aluksi luodaan UserService.cs-tiedosto Services-hakemistoon. Tiedostoon luodaan Authenticate-metodi, jonka tarkoituksena on tarkistaa, onko käyttäjä olemassa. Metodille annetaan käyttäjänimi ja salasana parametreinä. Metodi palauttaa käyttäjän, jos käyttäjä löytyy, muutoin se palauttaa null-arvon. UserService-luokka luodaan IUserService-interfacen pohjalta. UserService myös rekisteröidään AddScoped-metodilla Program.cs-tiedoston Main-metodissa. (Kuvio 33.)

```
// UserService.cs

namespace WebApi.Services
{
    public interface IUserService
    {
        Task<User> Authenticate(string username, string password);
        Task<IEnumerable<User>> GetAll();
    }

    public class UserService : IUserService
    {
        private readonly UserContext context;

        public UserService(UserContext context) { this.context = context; }

        public async Task<User> Authenticate(string username, string password)
        {
            var user = await Task.Run(
                () => this.context.Users.SingleOrDefault(
                    x => x.Username == username && x.HashedPassword == password));

            // return null if user not found
            if (user == null)
                return null;

            return user;
        }
    }
}

// Program.cs

builder.Services.AddScoped<IUserService, UserService>();
```

Kuvio 33. UserService ja sen rekisteröiminen

Authenticate-metodi pelkästään tarkistaa käyttäjän olemassaoloa. Todentamiselle tarvitaan myös BasicAuthenticationHandler-luokka, joka luodaan BasicAtuhenticationHandler.cs-tiedostoon. HandleAuthenticateAsync-metodi suorittaa käyttäjän todentamisen. Aluksi metodissa tarkistetaan, onko Authorization-otsikko HTTP-pyynnössä mukana, ja palautetaan virheviesti, jos se puuttuu. Seuraavaksi Authorization-otsikon arvo muunnetaan base64-muodosta käyttäjänimeksi ja salasanaksi. Aiemmin luotua UserService-luokan Authenticate-metodia käytetään käyttäjän todentamisessa. Claimit sisältävät tietoa käyttäjästä, joita käytetään onnistuneen todennuksen palauttamisessa. BasicAuthenticationHandler tulee vielä myös rekisteröidä Program.cs-tiedostossa. (Kuvio 34.)

```
// BasicAuthenticationHandler.cs

public class BasicAuthenticationHandler
    : AuthenticationHandler<AuthenticationSchemeOptions>
{
    private readonly IUserService _userService;

    public BasicAuthenticationHandler(
        IOptionsMonitor<AuthenticationSchemeOptions> options,
        ILoggerFactory logger, UrlEncoder encoder, ISystemClock clock,
        IUserService userService)
        : base(options, logger, encoder, clock)
    {
        _userService = userService;
    }

    protected override async Task<AuthenticateResult> HandleAuthenticateAsync()
    {
        if (!Request.Headers.ContainsKey("Authorization"))
            return AuthenticateResult.Fail("Missing Authorization Header");

        User user = null;
        try
        {
            var authHeader =
                AuthenticationHeaderValue.Parse(Request.Headers["Authorization"]);
            var credentialBytes = Convert.FromBase64String(authHeader.Parameter);
            var credentials =
                Encoding.UTF8.GetString(credentialBytes).Split(new[] { ':' }, 2);
            var username = credentials[0];
            var password = credentials[1];
            user = await _userService.Authenticate(username, password);
        }
        catch
        {
            return AuthenticateResult.Fail("Invalid Authorization Header");
        }

        if (user == null)
            return AuthenticateResult.Fail("Invalid Username or Password");

        var claims = new[] {
      new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
      new Claim(ClaimTypes.Name, user.Username),
    };

        var identity = new ClaimsIdentity(claims, Scheme.Name);
        var principal = new ClaimsPrincipal(identity);
        var ticket = new AuthenticationTicket(principal, Scheme.Name);

        return AuthenticateResult.Success(ticket);
    }
}

// Program.cs
builder.Services.AddAuthentication("BasicAuthentication")
    .AddScheme<AuthenticationSchemeOptions, BasicAuthenticationHandler>(
        "BasicAuthentication", null);
```

Kuvio 34. Käyttäjän todentaminen BasicAuthenticationHandler-luokalla sekä rekisteröiminen

Lopuksi valtuutuksen käyttöön ottamiseksi lisätään Authorize-attribuutti jokaiselle controllerille, jotka tarvitsevat sitä. Controllerin jokainen päätepiste vaatii käyttäjän valtuutusta, kun authorize-attribuutti on käytössä. Yksittäisille päätepisteille voi lisätä AllowAnonymous-attribuutin, jos halutaan, että käyttäjän valtuutusta ei tarvitse. (Kuvio 35.)

```
[Authorize]
[Route("[controller]")]
public class ScoreController : ControllerBase
{
    private readonly UserContext context;

    public ScoreController(UserContext context) { this.context = context; }

    [AllowAnonymous]
    [HttpGet]
    public List<Score> Get()
    {
        var scores = this.context.Scores.Include(s => s.User).ToList();
        return scores;
    }
    ...
```

Kuvio 35. Authorize- ja AllowAnonymous-attribuuttien lisääminen

5 ONGELMATILANTEET PELIN KEHITYKSESßÄ

Pelin teossa ilmeni tarve kopioida moniulotteinen taulukko. Taulukon kopioimisessa kuitenkin ilmeni ongelma, sillä JavaScript luo yleisimmillä menetelmillä taulukosta vain pinnallisen kopion (engl. shallow copy). Tämän vuoksi taulukon sisäiset taulukot osoittavat samoihin objekteihin, eikä kopioitu taulukko sisällä kopioituja arvoja. Ongelma tässä on, että kopioidun taulukon arvot muuttuvat, kun muutetaan alkuperäisen taulukon arvoja. Ratkaisu tähän on luoda syvä kopio (engl. deep copy), jolloin taulukon sisäiset taulukot ovat oikeita kopioita. Käytin syvän kopion tekemiseen aluksi map-funktiota, jonka avulla suoritin slice-funktion jokaiseen sisäiseen taulukkoon. Slice palauttaa kopion taulukosta. (Kuvio 36.)

```
let tetromino = tetrominos[player.tetromino.shape].grid.map(inner ⇒
    inner.slice()
);
```

Kuvio 36. Moniulotteisen taulukon kopioiminen

Toinen ongelmatilanne oli viiveen lisääminen sivuttaisliikkeen alkuun. Kun pelaaja painaa vasemman tai oikean liikenapin pohjaan, ensimmäisen liikkeen jälkeen tulee olla lyhyt tauko, jonka jälkeen liike alkaa toistumaan. Tämän toiminnon toteuttaminen oli hieman haastavaa. Toteutuksessa käytin delay-muuttujaa, joka määrää, kuinka usein liike tapahtuu, kun nappia pidetään pohjassa. Muuttujan arvo on oletuksena 133, joka viittaa millisekunteihin. Liikkeen toistamiseen käytetään useInterval-koukkua, joka toistaa funktion tietyn väliajoin. UseInterval ottaa funktion lisäksi delay-muuttujan argumentiksi. Näin muuttamalla delaytä voi muuttaa, kuinka usein funktio suoritetaan. Ensimmäisen liikkeen jälkeen delayksi asetetaan 10. (Kuvio 37.)

```
const [heldKey, setHeldKey] = useState<React.KeyboardEvent | null>(null);
const [heldKeys, setHeldKeys] = useState(new Set());

const [delay, setDelay] = useState(133);

const [keysSize, setKeysSize] = useState(heldKeys.size);
useInterval(
  () => {
    function doMove(key: React.KeyboardEvent) {
      move(key);
      setDelay(10);
    }
    keysSize && heldKeys.has(heldKey?.code) && heldKey
      ? doMove(heldKey)
      : setDelay(133);
  },
  keysSize ? delay : null
);
```

Kuvio 37. UseInterval-koukun käyttö delay-muuttujalla

Lisäksi Boardin onKeyUp-tapahtumaan lisättiin stopRepeat-funktio. Kun pelaaja päästää irti napista, delayksi asetetaan uudestaan 133 (kuvio 38).

```
const stopRepeat = (keycode: string) => {
  heldKeys.delete(keycode);
  setHeldKeys(heldKeys);
  setKeysSize(heldKeys.size);
  setDelay(133);
};
...
<StyledBoard
  onKeyUp={e => stopRepeat(e.code)}
...
```

Kuvio 38. Pelaajan liikkeen toiston lopettaminen stopRepeat-funktiolla

6 TEKNOLOGIOIDEN VERTAILU

Tässä luvussa vertaillaan Reactia ja ASP.NET Corea muihin käyttämiini teknologioihin. Vertailussa otan huomioon pääasiassa teknologioiden tehokkuuden, helppokäyttöisyyden ja toiminnallisuuden.

6.1 Front-endien vertailu

Front-endeistä Reactin lisäksi minulle on tuttu Vue.js. Vue.js on Javascript-framework, joka luotiin Evan Youn toimesta Googlella vuonna 2013. Vue.js:n ydin koostuu vain data bindingista ja komponenteista. Jos käyttäjä tarvitsee lisää ominaisuuksia, esim. reititystä, voidaan ottaa käyttöön niihin tarkoitettuja kirjastoja. (Cromwell 2016.)

Tehokkuudessa Vue.js voittaa Reactin. Esheten testauksessa Vue.js sai paremmat pisteet DOM:n manipulaatiossa, käynnistyksessä sekä muistin käytössä. Vue.js on 36 % nopeampi Reactia DOM:n manipulaatiossa, 19 % nopeampi käynnistyksessä sekä 21 % parempi muistin käytössä. (Eshete 2022.)

Koska Vue.js luotiin minimaaliseksi, se on yksinkertainen ja siksi helppokäyttöisempi kuin React. Vue.js:n integroiminen olemassa olevaan sovellukseen on helppoa sen joustavuuden vuoksi. Jos isompiin sovelluksiin tarvitaan ominaisuuksia, joita Vue.js ei tarjoa, niitä varten voidaan ottaa käyttöön virallisia kirjastoja, jotka on suunniteltu toimimaan keskenään. Tämä mahdollistaa Vue.js:n käytön joustavasti, koska käyttäjä voi valita, mitä ominaisuuksia käytetään sovelluksessa. Vue.js:n dokumentaatio on myös edellä Reactin dokumentaatiota. (Cromwell 2016; Nowak 2023.)

6.2 Back-endien vertailu

Back-endinä olen käyttänyt ASP.NET Coren lisäksi Express.js:ää, joten vertailen niitä keskenään. Express.js on framework, jota suoritetaan Node.js:llä. Ohjelmointikielenä se käyttää nimensä mukaisesti Javascriptiä. (Mastering Backend 2023.)

Tehokkuudessa ASP.NET Core voittaa Express.js:n; suurin osa ASP.NET Coren toteutuksista on moninkertaisesti tehokkaampi kuin Express.js:n toteutukset (kuvio 39). Jos web-sovelluksen back-endin tulee olla tehokas, ASP.NET Core on luultavasti parempi vaihtoehto. Jos kyseessä on kevyt sovellus, joka ei käytä paljoa resursseja back-endiltä, Express.js on myös hyvä framework.

Rnk	Framework	Best performance (higher is better)	
1	aspcore-ado-pg	363,344	100.0% (62.1)
2	aspcore-aot-ado-pg	350,712	96.5% (59.9)
3	aspcore-vb-mw-ado-pg	257,120	70.8% (43.9)
4	asp.net core [platform, my]	194,159	53.4% (33.2)
5	aspcore-vb-mw-ado-my	179,195	49.3% (30.6)
6	asp.net core [platform, mono, pg]	63,175	17.4% (10.8)
7	aspcore-mono-mw-pg	41,787	11.5% (7.1)
8	express-postgres	33,868	9.3% (5.8)
9	express-mysql	30,299	8.3% (5.2)
10	express-mongodb	29,354	8.1% (5.0)
11	aspcore-mono-mvc-pg	26,191	7.2% (4.5)

Kuvio 39. ASP.NET Coren ja Express.js:n suorituskykytestit (TechEmpower 2023)

Helppokäyttöisyydessä Express.js on huomattavasti parempi. Express.js-back-endin luominen tapahtuu *express*-paketin lisäämisellä projektiin npm:llä. Projektiin lisätään server.js-tiedosto, joka sisältää Express.js:n koodin (kuvio 40). Palvelinta suoritetaan komennolla ”*node server.js*”. Palvelin on yksinkertainen, eikä se vaadi paljoa konfigurointia sen käyttöönottamiseen. Opinnäytetyön luvussa 4.2 voi verrata yksinkertaista Express.js-palvelinta ASP.NET Core-projektiin. ASP.NET Core vaatii huomattavasti enemmän ymmärrystä sen toiminnasta verrattuna Express.js:ään. Helppokäyttöisyydessä tulee ottaa myös huomioon käytetyt ohjelmointikielet. Express.js käyttää Javascriptiä, mikä tekee siitä helpon back-endin yhdistämisen front-endiin, mikäli front-endin teknologiat käyttävät Javascriptiä. ASP.NET Core sen sijaan tukee C#:a, F#:a ja Visual Basicia, minkä vuoksi front- ja back-end kirjoitettaisiin eri kielillä.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Hello World!');
});

app.listen(3000);
```

Kuvio 40. Minimaalinen Express.js-palvelin (Mikoliuk 2023)

ASP.NET Core sisältää kattavasti olennaisia toiminnallisuuksia, jotka vaativat ylimääräisiä teknologioita Express.js:llä. Esimerkiksi Express.js:n turvallisuustoiminnot ovat yksinkertaisia, kun taas ASP.NET Coreen on sisäänrakennettu laajat turvallisuustoiminnot. Tämä tekee ASP.NET Coresta paremman laajoille ja komplekseille sovelluksille. (Mikoliuk 2023.)

6.3 Päätelmät

Teknologioita on monia ja varsinkin web-kehityksen maailmassa uusia teknologioita ilmestyy jatkuvasti. Vertailussa verrattiin Reactia Vue.js:ään sekä ASP.NET Corea Node.js:ään. Vertailussa otettiin huomioon tehokkuus, helppokäyttöisyys ja toiminnallisuus.

Vue.js on minimaalisempi kuin React. Vue.js mahdollistaa vain tarvittavien toiminnallisuuksien käytön, sillä Vue.js:n ydin koostuu minimaalisista toiminnallisuuksista. Muut ominaisuudet saadaan käyttöön virallisilla kirjastoilla. Vue.js on myös tehokkaampi kuin React.

Express.js on minimaalisempi kuin ASP.NET Core, joten se on helppokäyttöisempi. Tehokkuudessa ASP.NET Core voittaa Express.js:n. Express.js:ää käytetään enemmän pienemmissä sovelluksissa sen minimaalisuuden takia.

ASP.NET Corea taas käytetään paljon keskikokoisissa ja suurissa sovelluksissa sen laajojen ominaisuuksien vuoksi.

Teknologian valitsemisessa työhön ei välttämättä kannata valita teknologiaa, joka on tehokkain tai jolla on eniten ominaisuuksia. Oikea teknologia tiettyyn työhön riippuu työstä ja siitä, mitä teknologialla aiotaan tehdä. Yksinkertaiset sovellukset eivät välttämättä tarvitse monia ominaisuuksia tai tehokkuutta. Jos taas sovelluksen tarvitsee prosessoida paljon dataa, tehokkuus voi olla ensisijaisen tärkeä. Teknologia siis kannattaa valita ottamalla huomioon asiat, jotka sen tulee saavuttaa.

7 POHDINTA

Opinnäytetyön tavoitteena oli tutustua Reactiin ja ASP.NET Coreen, sekä tehdä näiden pohjalta Tetris-peli. Työssä esitettiin käytetyt teknologiat. Teknologioilla lähdettiin alkuun ympäristöjen luomisella ja projektien aloittamisella sekä lopullisen projektien esittämisellä.

Tunsin onnistuneeni pelin luomisessa. Minulla oli vähän kokemusta ASP.NET Coresta ennen opinnäytetyötäni, mutta ei yhtään Reactista. Sain silti aikaiseksi melko hyvin toimivan pelin. Työn onnistumiseen vaikutti aikaisempi osaamiseni, etenkin Javascript- ja Vue.js-osaamiseni, sekä ymmärrys tietokannoista ja HTTP-pyynnöistä.

Pelissä on kehitettävää turvallisuuden kannalta, sillä käytin yksinkertaisia menetelmiä turvallisuuden takaamiseksi. Lisäksi käyttäjä ei voi muokata pelin asetuksia, joten kehitettävää on sekä front- että back-endissä. Jotkin pelin mekaniikat ovat yksinkertaisia, esimerkiksi pisteiden lasku, jota voi myös kehittää.

Opinnäytetyötä tehdessä opin paljon sekä Reactista että ASP.NET Coresta. Reactin osalta opin komponenteista, data bindingista, JSX:stä ja Reactin koukuista. Opin myös HTTP-kyselyiden lähettämisestä back-endille Javascriptin Fetch API:lla. Tämän lisäksi opin Reactin ja ASP.NET Coren eroja Vue.js:ään ja Express.js:ään.

Opinnäytetyö käsittelee Reactin ja ASP.NET Coren perusasioita, muttei edistyneitä tekniikoita. Tämän vuoksi opinnäytetyöstä ei luultavasti ole paljoa hyötyä Reactin ja ASP.NET Coren osaajille. Sen sijaan näistä kiinnostuneet aloittelijat voivat hyötyä opinnäytetyöstä. Myös Tetris-pelin kehittäjät riippumatta käytetyistä teknologioista voivat hyötyä työstä, koska monia työssä käytettyjä tekniikoita voidaan käyttää muillakin teknologioilla.

LÄHTEET

Amir, M. 2023. Full Stack Developer Roadmap 2023: Master the Stack. The Geeks Bot 26.3.2023. Viitattu 20.8.2023 https://web.archive.org/web/20230502091617/https://www.thegeeksbot.com/2023/03/full-stack-developer-complete-roadmap.html.

Baer, E. 2018. What React Is and Why It Matters. O'Reilly Media. Viitattu 26.3.2023 https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html.

Britannica 2024. Tetris. Viitattu 4.4.2024 https://www.britannica.com/topic/Tetris#ref1062842.

Chavan, Y. 2021. JSX in React – Explained with Examples. FreeCodeCamp 1.2.2021. Viitattu 18.9.2023 https://www.freecodecamp.org/news/jsx-in-react-introduction/.

Clark, J. 2022. The Top 10 Backends For React. Back4App. Viitattu 20.8.2023 https://blog.back4app.com/best-backends-for-react/.

Cromwell, V. 2016. Evan Vue. Between the Wires 3.11.2016. Viitattu 30.12.2023 https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/.

Eshete, D. 2022. Comparative Web Performance evaluation of Vue & React using JSWFB. Medium 15.3.2022. Viitattu 28.12.2023 https://medium.com/@danialeshete/comparative-web-performance-evaluation-benchmarking-of-vue-react-using-jswfb-a76982097225.

Geekboots 2022. ReactJS and its usability. Viitattu 13.11.2023 https://www.geekboots.com/story/reactjs-and-its-usability

HyperionDev 2017. 10 Different Types of Software Development. Viitattu 9.4.2024 https://blog.hyperiondev.com/index.php/2017/09/26/types-of-software-development/.

iFour Technolab 2019. Differences Between ASP.NET and ASP.NET Core - ASP.NET vs ASP.NET Core. Viitattu 15.11.2023 https://www.ifourtechnolab.com/blog/differences-between-asp-net-and-asp-net-core-asp-net-vs-asp-net-core.

IGN 2017. 5 Minutes of Puyo Puyo Tetris Running on Switch. Viitattu 30.11.2023 https://www.ign.com/videos/5-minutes-of-puyo-puyo-tetris-running-on-switch.

Ilyukha, V. 2020. ASP.NET vs ASP.NET Core. Jelvix. Viitattu 25.2.2024 https://jelvix.com/blog/asp-net-vs-asp-net-core.

Indeed 2023. 15 Most In-Demand Programming Jobs for 2023 (With Salaries). Viitattu 20.8.2023 https://www.indeed.com/career-advice/finding-a-job/most-in-demand-programming-jobs.

Kent, S. 2001. The Ultimate History of Video Games: from Pong to Pokemon and beyond...the story behind the craze that touched our lives and changed the world. New York: Three Rivers Press.

Mastering Backend 2023. ExpressJS Tutorial: The Ultimate Guide (2023). Viitattu 17.12.2023 https://masteringbackend.com/posts/expressjs-5-tutorial-the-ultimate-guide.

Mardan, A. 2017. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL. New York: Manning Publications Co. LLC.

Meta 2023. React. Viitattu 23.8.2023 https://react.dev/.

Metts, J. 2006. Tetris from the Top: An Interview with Henk Rogers. NintendoWorldReport 6.4.2006. Viitattu 30.11.2023 https://www.nintendoworldreport.com/interview/11267/tetris-from-the-top-an-interview-with-henk-rogers.

Microsoft 2022. Overview of ASP.NET Core. Viitattu 9.9.2023 https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0.

Mikoliuk, E. 2023. Exploring Express Vs ASP.NET: A Detailed Comparison For Web Developers. MarketSplash. Viitattu 17.12.2023 https://marketsplash.com/tutorials/express/express-vs-asp-net/.

Nguyen, S. 2020. What is npm? A Node Package Manager Tutorial for Beginners. FreeCodeCamp 16.7.2020. Viitattu 11.9.2023 https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/.

Nowak, M. 2023. Vue vs. React in 2023 - Comparison of Two Most Popular JS Frameworks. Monterail. Viitattu 2.1.2024 https://www.monterail.com/blog/vue-vs-react.

Redhat 2019. What is an IDE? Viitattu 11.9.2023 https://www.redhat.com/en/topics/middleware/what-is-ide.

Semah, B. 2022. What Exactly is Node.js? Explained for Beginners. FreeCodeCamp 5.12.2022. Viitattu 11.9.2023 https://www.freecodecamp.org/news/what-is-node-js/.

Stack Overflow 2023. Stack Overflow Developer Survey 2023. Viitattu 20.8.2023 https://survey.stackoverflow.co/2023/.

TechEmpower 2023. Web Framework Benchmarks. Viitattu 17.12.2023 https://www.techempower.com/benchmarks/#section=data-r22&hw=ph&test=fortune&f=zik0z3-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-hra0hr-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-zik0zj-1ekf&p=zik0zj-zijocf-zijocf-zik0zj-zik0zj-sf.

Tetris.wiki 2015. Original Tetris. Viitattu 21.8.2023 https://tetris.wiki/File:Original_Tetris.png.

Tetris.wiki 2019. Tetrominoes. Viitattu 30.11.2023 https://tetris.wiki/File:Tetrominoes.png.

Weisberger, M. 2016. The Bizarre History of ’Tetris’. Livescience 13.10.2016. Viitattu 9.4.2024 https://www.livescience.com/56481-strange-history-of-tetris.html.