D O K U M E N T Á C I Ó

Vizsgaremek

SZOFTVERFEJLESZTŐ ÉS -TESZTELŐ TECHNIKUS SZAKMA

Tartalomjegyzék

Bevezetés

A projektcsapatunk két főből áll, Kiss Marcell és Kiss Olivér személyében. A projekt célja az volt számunkra, hogy egy munkaerő nyilvántartó adatbázist létrehozzunk, egy fiktív munkaerő kölcsönző cég részére, és ezt egy weboldalon keresztül, online felületen megvalósítsuk. Kiss Marcell személyesen, már dolgozott fejvadász cégekkel és azok vállalatirányítási rendszerükben, és ott nem volt egy hasonló adatbázis elérhető. Ez adta az alapvető ötletet, hogy megalkothassuk a programot. Ezúttal a szoftver teljes dokumentációjával szeretnénk leírni, hogy miként valósultak meg terveink lépésről lépésre.

Technikai megvalósítást segítő programok és programozási nyelvek

A projektünket a következő számítástechnikai programokban valósítottuk meg, amelyek nélkülözhetetlenek voltak a reszponzív weboldal megalkotásában. A képzés során megismert technológiákat vettük alapul.

Weboldal készítésére felhasznált fejlesztői környezetnek a Visual Studio Code nevezetű programot választottuk. Azért választottuk ezt a programot, mert ez a fejlesztői környezet sokat segít a kódok írása közben, sokszor vannak felajánlásai, amivel egyszerűbb a kódolás. Emellett számos kiegészítőt lehet telepíteni a programon belül, ami tovább segíti a munkákat. A Visual Studio Code-on belül az ES7+ React/Redux/React-Native snippets, Prettier-Code formatter, REST Client nevezetű kiegészítőket telepítettük fel, hogy minél több segítség legyen a programon belül. Ezek mellett használható egy beépített terminál, amin keresztül különböző parancsokkal csomagokat telepíthetünk, és indíthatunk el, ami szükséges a frontend és backend programok felépítéséhez.

A kiválasztott fejlesztői környezet mellé telepíteni kellett a Node.js programcsomagot, ami egyben egy programozási keretrendszer is. A Node.js egy nyílt forráskódú, többplatformos, backend JavaScript futtatókörnyezet, és JavaScript kódot hajt végre webböngészőn kívül. A Node.js lehetővé teszi a JavaScript használatát parancssori eszközök írásához és szerveroldali parancsfájlok futtatásához, a parancsfájlok szerveroldali futtatásához, hogy dinamikus weboldaltartalmat hozzanak létre, mielőtt az oldalt elküldené a felhasználó webböngészőjének.

Adatbázis készítését a phpMyAdmin nyílt forráskódú adminisztrációs eszközön végeztük, ahol a MySQL nyelv számára segít adatbázisokat építeni és kezelni. Az adatbázis kezelő programot a XAMMP Control Panel segítségével használtuk. A XAMPP egy nyílt forráskódú, többplatformos webszerver-megoldáscsomag, amely lehetővé teszi az átállást a helyi tesztkiszolgálóról egy élő kiszolgálóra.

A weboldal projekt elkészítésében a következő programozási technológiákat alkalmaztuk. Frontend programozáshoz főként a HTML, a CSS, a Bootstrap, a JSON és a JavaScript nyújtott segítséget, illetve a React.js keretrendszer, amellyel elkészítettük a fő alkalmazást. Backend programozáshoz a Node.js keretrendszer segítségével építettük fel a REST API-t, és alkalmaztuk az adatbázis kezelési módszereket a MySQL nyelv segítségével. Az adatbázisunk szerkezeti ábráját a Diagrams.net nyílt forráskódú, platformok közötti diagramrajzoló szoftverrel készítettük, ami szintén megtalálható a projektünk mellékletei között.

Tesztfuttatást is készítettünk, és végeztünk a projektünkhöz, amelyet egy külön csatolt dokumentumban fogjuk részletesebben ismertetni. A tesztet a Visual Studio 2019 nevű fejlesztői környezetben készítettük, főként a C# programozási nyelvet alkalmazva.

Ezek voltak a projektünket megvalósítását segítő programok és alkalmazások. Lépjünk tovább a dokumentációban, ahol az alkalmazás felépítése lesz részletezve az alapoktól kezdve.

A projekt alapjainak a felépítése

A következő lépésekkel épült fel a projekt törzse a frontend, a backend, és az adatbázis oldalt. Az első lépésünk az volt, hogy egy API-t építsünk fel MySQL adatbázis háttérrel Node.js keretrendszer alatt. A backend nevezetű mappát létrehozva a projektben, meg kellett nyitni a beépített terminált, ahol a szükséges csomagokat telepíteni kellett a Node,js-hez. A csomagok kezelését a Node Package Maneger segítségével végezzük. Ennek a telepítéséhez kellett megadni az npm init -y parancsot létrehozva a package.json file-t. A telepítés után a package.json-ban módosítani kellett, a start script-et a következőre:"start":"nodemon index.js". Ezek után a npm start parancs futtatásával már megy a script később, ezzel a paranccsal indul a szerver.

A következő lépés az Express csomag telepítése volt a npm install express paranccsal, amely a szerverhez különböző kiszolgáló tevékenységet nyújt. A következő csomag a Cors telepítése volt, hogy cross-origin kérések is rendben menjenek. Ez a npm install cors paranccsal indult el. A MySQL nyelv adatbázis kezeléshez szükséges csomag telepítése a npm install mysql paranccsal történik. Végül a Nodemon telepítésére került sor, hogy ne kézzel kelljen újraindítgatni a szervert a npm install --save-dev nodemon parancs megadásával.

A fenti lépésekben a különböző már feltelepített csomagok segítségével elkészült a backend technikai háttere. Következő lépésünk az alkalmazáshoz létező backend, egy frontend nevű mappa létrehozása, ahol React.js keretrendszer telepítésével, megalkotjuk a frontend rész technikai hátterét.

Elsőnek itt is meg kellett nyitni a beépített terminált a React App létrehozásához a npx create-react-app . parancs megadásával, ekkor telepíti fel a kért csomagokat. Egy több mappából, és file-ból álló programcsomagot kapunk alapjaiban készen. Bootstrap telepítése szükséges a fejszleszői munkához, amelyez a npm install --save bootstrap paranccsal megadva végzünk el, majd az index.js nevű file-ba tegyük be a következőt: import 'bootstrap/dist/css/bootstrap.css'; ezzel importálja be a Bootsrap keretrendszert az alkalmazásba. Végül a react-router-dom telepítése volt szükgéges a npm i --save react-router-dom@5.1 paranccsal. Ezek után a npm start parancs futtatásával a beépített terminálba, elindul maga az alkalmazás az alapértelmezett webes böngészőben.a megadott localhoston.

A fent leírt lépésekkel alkottuk meg a projektünk alapjait. A továbbiak részletesen leírjuk, hogy milyen részekből áll az alkalmazásunk.

Alkalmazás részletes ismertetése

Három nagyobb részre bontva készítettük el az alkalmazásunkat. Elsőnek az adatbázisunk létrehozása, és felépítése volt a feladatunk, mert ebből fog dolgozni a programunk. Az adatbázis megalkotása, után a projekt backend részét építettük fel a szükséges elemekkel, és végül kezdtük el a frontend elemek létrehozását. A következő sorokban fogjuk ismertetni a három fő részét, a projektünknek.

A projek adatbázis részének az elkészítése

Az adatbázisunkat a XAMMP Control Panel-en keresztül végeztük az Apache és MySQL modulokat elindítva, és használva a phpMyAdmin oldalán. Elsőnek létrehoztuk az adatbázist munkaero néven, majd négy táblát alkottunk munkavallalok, dolgozok, becenevek és elerhetosegek névvel. A táblák létrehozása után feltöltöttük a szükséges adatokkal az adatbázisunkat, amely később fontos információkkal fog szolgáltatni az alkalmazásunkban. A kész adatbázis export verziója munkaero_export_dump.sql nevű file-ként található meg, de mellékeltük azt a file-t is amit beimportáltunk a kész adatbázis adatokat a phpMyAdmin oldalára munkaero.sql néven. Fontos, hogy a korábban említett két modul (Apache és MySQL) legyen mindig elindítva a XAMPP-on keresztül, mert enélkül nem fogja betölteni a későbbiekben kért adatokat, a backend és frontend projekthez.

A projekt backend részének az elkészítése

Az adatbázisunk elkészítése után a backend alkalmazás megírása volt a következő lépés. A backend projekt mappában több file is található. A node_modules mappa és a benne található adatok, a package.json és a package-lock.json file-ok, a korábban beírt parancsokra lettek telepítve. Amit mi készítettünk három fő adathalmaz az a index.js, a dbrepo.js, és a client.rest nevezetű file-ok. Ezek tartalmazzák a backend forráskódokat, amit részletesebben mutatunk be felépítésük sorrendjében.

Az index.js file tartalmazza a backend kiszolgáló legfontosabb elemeit. Ameddig nem a végpontok megírása kezdődik, addig tart a kiinduló konfigurációja a szervernek, amely az alábbi elemekből áll.

A ,const express=require('express'); const cors=require('cors'); const mysql=require('mysql'); const app=express(); kódokkal vannak behíva a már korábban telepített, és szükséges csomagok, hogy működhessen a program.

A const {becenevek,dolgozok,munkavallalok,ujbecenev,modositbecenev,torolbecenev, elerhetoseg} =require('./dbrepo.js'); kóddal vannak behívva a dbrepo.js file-ban megírt végpontokhoz szükséges promise-ok, amivel a szükséges adatbázis parancsokat alkalmazza a program.

Az app.use(cors()); app.use(express.json()); app.use(express.urlencoded({extended:true})); kódokkal van alkalmazva a zárójelekben megnevezett funkciók, és parancsok. A const conn=mysql.createConnection({host:"localhost",user:"root",password:"",database:"munkaero"}); kódsorral van létrehozva a kapcsolat a MySQL adatbázissal.

A rendszerüzenet az app.listen(8000,()=>{console.log("A szerver működik és fut");}); kóddal van megírva, ami a 8000-es porton futattja a szervert. A projekt szerver neve az app.get('/', (req,res)=>{res.send("Munkavállaló nyilvántartás");
}); kóddal van megírva. Eddig tartott a kódnak azon része, ami tartalmazza a backend kiszolgáló legfontosabb elemeit.

Az index.js file-ban található többi kód a megírt végpontokat tartalmazza. Az itt látható kódokhoz szorosan tartoznak a dbrepo.js file-ban megírt Promise kódok, amelyekkel együtt a végpontokhoz meghívva működik a backend szerver adatbázis kezelő része.

Az app.get('/becenevek',async(req,res)=>
{becenevek(conn).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amivel a /becenevek nevű Promise parancs az adatbázisban található becenevek adathalmazba található információkat, lekérdezéssel behívja a projektbe, a GET REST API paranccsal.

Az app.get('/dolgozok',async (req,res)=>
{dolgozok(conn).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amivel a /dolgozok nevű Promise parancs az adatbázisban található dolgozok adathalmazba található információkat, lekérdezéssel behívja a projektbe, a GET REST API paranccsal.

Az app.get('/munkavallalok',async (req,res)=>
{munkavallalok(conn).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amivel a /munkavallalok nevű Promise parancs az adatbázisban található munkavallalok adathalmazba található információkat, lekérdezéssel behívja a projektbe, a GET REST API paranccsal.

Az app.post('/ujbecenev',async (req,res)=>
{ujbecenev(conn,req.body.becenev).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amely segítségével új adatot tudunk bevinni, az /ujbecenev nevű Promise paranccsal a becenev adathalmazba a POST REST API paranccsal.

Az app.patch('/modositbecenev',async (req,res)=>
{modositbecenev(conn,req.body).then(adat=>res.json(adat)).catch(err=>res.send(err));
}); kóddal van leírva az a végpont, amely segítségével meglévő adatot tudunk módosítani, a /modositbecenev nevű Promise paranccsal a becenev adathalmazba a PATCH REST API paranccsal.

Az app.delete('/torolbecenev',async (req,res)=>
{torolbecenev(conn,req.body.id).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amely segítségével meglévő adatot tudunk törölni, a /torolbecenev nevű Promise paranccsal a becenev adathalmazba a DELETE REST API paranccsal.

Az app.get('/elerhetoseg',async (req,res)=>
{elerhetoseg(conn).then(adat=>res.json(adat)).catch(err=>res.send(err));}); kóddal van leírva az a végpont, amivel az /elerhetoseg nevű Promise parancs az adatbázisban található elerhetoseg adathalmazba található információkat, lekérdezéssel behívja a projektbe, a GET REST API paranccsal.

Miután leírtuk az index.js file-ban található kódokat funkciójukkal együtt, a dbrepo.js file-ban található forráskódokra térünk rá, mert ezek nélkül nem működnek a végpontok, az index.js file-ban. A Promise-ok létrehozása az index.js végpontjaihoz volt itt a fő feladat, mert csak ezek megalkotásával, működnek megfelelően az adatbázis kezelésben alkalmazott utasítások.

Teljes funkcionalitású API létrehozása volt ezzel a fő cél, hogy ne a végpontoknál szerepeljenek az adatbázis műveletek, ezért külön file-ba tettük őket. Az adatbázis műveletek Promise-ba ágyazva futnak, hogy aszinkron módon is tudjuk használni őket. Ennek segítségével egy jól átlátható kód készült a dbrepo.js és index.js file-okban. Ebből adódóan mindig oda kellett figyelni programozás közben, hogy mindenhol szerepeljenek a megfelelő adatok, mert azok hiányában nem működne rendeltetésszerűen a program.

A module.exports.becenevek=function(conn)
{return new Promise((reject,resolve)=>{conn.query("select * from becenevek",(err,rows)=>{if(err) {reject(err);} else {resolve(rows);}});});} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az összes becenevet kilistázza a frontend alkalmazás felé.

A module.exports.munkavallalok=function(conn)
{return new Promise((reject,resolve)=>{conn.query("select * from munkavallalok", (err,rows)=>{if(err){reject(err);} else {resolve(rows);}});})} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az összes munkavállalót kilistázza a frontend alkalmazás felé.

A module.exports.dolgozok=function(conn)
{return new Promise((reject,resolve)=>{conn.query("SELECT dolgozok.id,munkavallaloid, becenevid,eletkor,belepesdatum,becenev,munkakor,eredetinev FROM dolgozok "+"INNER JOIN becenevek ON becenevid=becenevek.id "+"INNER JOIN munkavallalok ON munkavallaloid=munkavallalok.id",(err,rows)=>{if(err){reject(err);} else {resolve(rows);}});})} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az összes dolgozót kilistázza a frontend alkalmazás felé. Mivel ez egy több táblából dolgozó lekérdezés, emiatt külön erre figyelve kellett megírni egy többszörösen összetett lekérdezés utasítást.

A module.exports.ujbecenev=function(conn,becenev)
{return new Promise((reject,resolve)=>{conn.query("insert into becenevek (becenev) values(?)", [becenev],error=>{if(error){reject(error);} else {resolve({status:201,message:"Becenév sikeresen beillesztve!"});}});});} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az egy új becenevet hozunk létre.

A module.exports.modositbecenev=function(conn,beceNevAdat)
{return new Promise ((reject,resolve)=>{conn.query("update becenevek set becenev=? where id=?", [beceNevAdat.becenev,beceNevAdat.id],(error)=>{if (error){reject(error);} else {resolve({status:201,message:"Becenév sikeresen módosítva!"});}});});} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az egy már meglévő becenevet módosíthatunk.

A module.exports.torolbecenev=function(conn,id)
{return new Promise ((reject,resolve)=>{conn.query("delete from becenevek where id=?",[id], (error)=>{if(error){reject(error);} else{resolve({status:201,message:"Becenév sikeresen törölve lett!"});}});});} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az egy már meglévő becenevet törölhetünk, ha ez engedélyezett, mert újonnan bevitt becenevet bármikor törölhetünk, de ahol már meglévő becenevet szeretnénk törölni, a többszörösen összekapcsolt táblák miatt nem fogja engedni a renszer.

A module.exports.elerhetoseg=function(conn)
{return new Promise((reject,resolve)=>{conn.query("select * from elerhetoseg",(err,rows)=>{if(err) {reject(err);} else {resolve(rows);}});});} kóddal van megírva az a Promise, amely a tartalmazza azt az adatbázis kezelés módszert, amellyel az összes elérhetőséget kilistázza a frontend alkalmazás felé.

Miután leírtuk a dbrepo.js file-ban található kódokat funkciójukkal együtt, csak a client.rest file-ban található kódok magyarázata maradt hátra a backend projektben. A Visual Studio Code-ba korábban említett, és telepített REST Client nevezetű kiegészítő program funkciója most fog teret nyerni a client.rest file-ban. Ez szükséges, ahhoz, hogy az adatbázis kezelő parancsok és lekérdezések funkciói, a frontend alkalmazástól függetlenül is tesztelhetőek legyenek, a létrehozott végpontok alapján. A következő kódolást kellett végezni, ebben a file-ban.

A GET http://localhost:8000/ megadásával adjuk meg a szerver elérési címét. Fontos, hogy ezután ha új REST Client-et akarunk megadni, akkor a sorokat el kell választani ###-el és szükséges a sortörés is, illetve az elérési cím megadása után is, ami a projektben látszódik is. De a lenti dokumentációban is ez alapján írjuk le.

###
GET http://localhost:8000/becenevek utasítással hozzuk létre, hogy backend oldalt lefusson a GET paranccsal az összes becenév listázása az adatbázisunkból a korábban megírt végpont alapján.

###
GET http://localhost:8000/munkavallalok utasítással hozzuk létre, hogy backend oldalt lefusson a GET paranccsal az összes munkavállaló listázása az adatbázisunkból a korábban megírt végpont alapján.

###

GET http://localhost:8000/dolgozok utasítással hozzuk létre, hogy backend oldalt lefusson a GET paranccsal az összes dolgozó listázása az adatbázisunkból a korábban megírt végpont alapján.

###

POST

http://localhost:8000/ujbecenev

Content-Type: application/json

{"becenev":"Jankó"} utasítással hozzuk létre, hogy backend oldalt lefusson a POST paranccsal a megadott új becenév bevitele az adatbázisunkba a korábban megírt végpont alapján.

###

PATCH http://localhost:8000/modositbecenev

Content-Type: application/json

{"id":1,"becenev":"Laca} utasítással hozzuk létre, hogy backend oldalt lefusson a PATCH paranccsal a megadott becenév módosításának a bevitele az adatbázisunkba a korábban megírt végpont alapján.

###

DELETE

http://localhost:8000/torolbecenev

Content-Type: application/json

{"id":1}utasítással hozzuk létre, hogy backend oldalt lefusson a DELETE paranccsal a megadott becenév törlése az adatbázisunkba a korábban megírt végpont alapján.

###

GET http://localhost:8000/elerhetoseg utasítással hozzuk létre, hogy backend oldalt lefusson a GET paranccsal az összes elérhetőség listázása az adatbázisunkból a korábban megírt végpont alapján.

Az adatbázis terv leírása, és a backend programunkról készült részletes dokumentáció után, a következő fejezetben a frontend programunk leírásáról lesz szó, amivel később szemléltetni is tudjuk a grafikai megvalósítását a tervezett alkalmazásunknak.

A projekt frontend részének az elkészítése

A korábban létrehozott frontend mappába kerültek React App alkalmazás elemei hasonlóan, mint a backend esetén, vannak csomagok telepítve, amik a fejlesztést könnyítik. Ezek a node_modules, a public, a src mappák, és a .gitnore, package-lock.json, package.json és README.md file-ok. A munkák érdemleges lépései a src nevű mappában voltak, a projektünk folyamán. A React.js keretrendszer segítségével nem hagyományos lapozó file-os weboldalakat lehet készíteni, hanem minden funkció a JavaScript nyelv segítségével rögtön betöltődik anélkül, hogy újabb file-t kelljen megnyitnia. A src mappában a components mappa és azokban létrehozott file-ok kivételével minden más file-t már tartalmazott a React App. A fő munkaterek amiket haszáltunk a programozás közben a components mappa összes tartalma, az App.js, az index.js, az index.css, és az App.css file-ok. A frontend felépítésének sorrendjében fogunk haladni, a dokumentációban.

Az első lépéseink közt volt, hogy az index.js file-ba beimportáljuk a Bootstrap keretrendszert, és a nem szükséges import parancsokat eltávolítottuk. Ezután meg is kezdtük a munkát a frontend törzsével. Az App.js file tartalmazza az alkalmazásunk gerincét, amely a kövezkezőként épült fel.

A megírt JavaScript file-ok a components mappából a következőképpen hívtuk be. Az import Header from './components/Header'; import Becenevek from './components/Becenevek'; import Munkavallalok from './components/Munkavallalok'; import Dolgozok from './components/Dolgozok'; import Ujbecenev from './components/Ujbecenev'; import Footer from './components/Footer'; import Elerhetoseg from './components/Elerhetoseg'; parancsokkal történtek meg a behúzások. A megírt components-ek után a React Router DOM behíváva volt a következő lépés. Ez az import {BrowserRouter,NavLink,Switch,Route, Redirect} from 'react-router-dom'; prananccsal történt meg. Ezt követően a CSS fájlok behívása történt meg az import './App.css'; import './index.css'; parancsokkal.

Frontend alkalmazásunk arculatáért felelős HTML elemeket a function App() {return(....);} export default App;-on belül írtuk meg. Most a legfőbb Tag-eket fogjuk ismertetni, hogy mi mit is jelent. A div className="container"-en belül helyezkednek el az összes szerkezeti elemei a weboldalnak, amik a components mappából importálva kapnak funkciókat, hogy interaktív legyen a megjelenő oldal. A div className="divison1"-ben vannak megírva a Header tag részei, vagyis a fejléc konténer részei, amelyek már információkat szolgának a weboldalon.

A BrowserRouter nyitó és záró tag-ek közt (<BrowserRouter>...</BrowserRouter>) helyezkedik el az a menüsáv és adathalmaz rész, amivel az adatbázis adatai fognak a kért funkciókkal behívásra kerülni a backend alkamazás segítségével. Az importált React Router DOM segítségével a következő tag-ekkel, és beépített léptető és navigáló funkciójukkal lehetséges a menüsávban kiválaszani a behívni kívánt adatokat. A BrowserRouter, a NavLink, a Switch, a Route, és a Redirect nyitó és záró tag-ek elemei és beépített funkciói azok, amelyek lehetővé teszik, az interaktív navigációs menü használatot anélkül, hogy lapozó file-okat kelljen alkalmazni, mint a hagyományos weboldalak esetében. A nav className="navbar-ban vannak azok a Bootsrap menü elemek, amelyek segítségével responsive-an jelennek meg a menü adatai. A div className='footer'-ben vannak megírva a Footer tag részei, vagyis a lábléc konténer részei, amelyek még töb információkat szolgának a weboldalról.

Most értünk el a components nevű mappa JavaScript elemeinek ismeretéséhez. Ezekben a file-okban vannak megírva azon funkciók, amik segítségével az App.js file-ban található arculati elemek működőképesek lesznek, és itt kapcsolodók össze a frontend programunk a backend résszel, ami a külső adatbázból, fogja haszálni a rendelkezésre álló adatokat. Az App.js-be importált components elemek azok, amik a React Router DOM {BrowserRouter,NavLink,Switch,Route, Redirect} elemeivel lépnek összhangba. Sorban lesznek bemutatva a JavaScript file-ok.

A Becenev.js file-ban két funkció is meg lett írva. Az egyik a módosítás, a másik a törlés funkció. Ebben a JavaScript file-ban található a legosszabb funkcóival kódunk a projekt során. Elsőnek az import {useState} from 'react'; paranccsal kellett behívni az useState-t. A function Becenev({elem,setRefresh}) {..}-en belül a useState segítségével létrehoztuk a setBecenev és setFormOn elemeket, const [formON,setFormOn]=useState(false); és const [becenev,setBecenev]=useState(elem.becenev); kódokkal.

A sendAdat funckió létrehozásával megírtuk a módosítás funkció kódját, hogy az await fetch-el a http://localhost:8000/modositbecenev címre küldje a beírt módosított adatot. Ezen belül PATCH REST API utasítással, és a JSON.stingify(adat) és Content-type":"Application/json megjelenési formával adtuk meg a kód algoritmusát. Az await res.json();alert(valasz.message);-el fog írni a program, a bevitt értékek esetén.
A const sendAdat=async(adat)=>{const res=await fetch('http://localhost:8000/modositbecenev', {method:"PATCH", headers:{"Content-type":"application/json"}, body:JSON.stringify(adat)}); const valasz=await res.json();alert(valasz.message);} a teljes kódja a funkció működésének.

A const becenevTorles=async(id)=>{...}-ban leírtak létrehozásával megírtuk a törlés funkció kódját, hogy az await fetch-el a http://localhost:8000/torolbecenev címre küldje a kért parancsot. Ezen belül DELETE REST API utasítással, és a JSON.stingify(id) és Content-type":"Application/json megjelenési formával adtuk meg a kód algoritmusát. Az await res.json()-el és az abban található if-else elágazással a program fog írni, egy üzenetet, ha nem törölhető egy becenév, az adatbáziskapcsolatok miatt. A setRefresh(prev=>!prev);-el van megírva az a lépés, hogy a végrehajtott műveletek esetén, legyen ez a módosítás, vagy törlés mindig vissza álljon a webes felület az eredeti lapra, hogy ne kelljen feleslegesen kattintani.
A const becenevTorles=async(id)=>{const res=await fetch('http://localhost:8000/torolbecenev', {method:"DELETE", headers:{"Content-type":"application/json"}, body:JSON.stringify(id)}); const valasz=await res.json();if(valasz.sqlMessage){alert("A kiválasztott név nem törölhető, mert "+valasz.sqlMessage);} else {alert(valasz.message);}setRefresh(prev=>!prev);} const onSubmit=(e)=>{e.preventDefault(); sendAdat({"id":elem.id, "becenev":becenev})setRefresh (prev=>!prev);} a teljes kódja a funkció működésének.

A return (....);-en belül létrehozott formai elemekkel kapnak arculatot a korábban megírt funkciók. A HTML Tag-ek között van megírva az összes formai elem. A {!formON ? <h4>{elem.becenev}
</br><span onClick={()=>setFormOn(prev=>!prev)} className="badge rounded-pill bg-primary">Adat módosítása<span onClick={()=> becenevTorles({"id":elem.id})} className="badge rounded-pill bg-danger">Adat törlése
</br></h4> kód sorokkal vannak megírva a modosítás és törlés gombok megjelenési formái. A <span onClick={()=>setFormOn(prev=>!prev)} className="badge rounded-pill bg-primary">Vissza a becenevek listájához
</br> sorokkal van megírva a visszalépés gomb, hogy kattintásra az eredeti lista oldalára vissza lépjen a weblapunk.

A Becenevek.js file-ba található megírva az a funkció, ami az adatbáziból a beceneveket listázza a backend segítségével. Első lépésben beimpotáltuk az useStaste és useEffect react funkciókat, illetv a Becenev.js file-t az import {useState,useEffect} from 'react'; import Becenev from './Becenev'; kódokkal.

A szükséges elemek behívása után a következő kóddal kellett megírni, hogy a frontend oldalt megjelenjenek a kért adatok a backend segítségével:

A function Becenevek() {const [becenevek,setBecenevek]=useState([]); const [refresh,setRefresh]=useState(false); useEffect(()=>{fetch('http://localhost:8000/becenevek') .then(adat=>adat.json()).then(adat=>setBecenevek(adat)).catch(err=>console.log(err));},[refresh]); return (<h3>Becenevek listája:</h3>
</br>{becenevek.map((elem,index)=> (
<h5> <Becenev key={index} elem={elem} setRefresh={setRefresh}/></h5>))})} export default Becenevek; kód szolgálja ki az adatbekérést funkciót.

A létrehozott kóddal function-ban használt useState-el hoztuk létre a setBecenevek elemet, ami alapjául szolgált az adatbehívásban. Az useEffect-et használva a fetch,then,catch elemekkel adtuk meg, hogy az http://localhost:8000/becenevek elérhetőségen kérje le az adatokat, JSON formátumban legyenek listázva a setBecenevek-be, és hiba esetén legyen egy hiba (error) üzenet. A return({..}); résznél írtuk meg becenevek.map((elem,index))=>() funkcióval, hogy a .map segítségével a megadott elem-ek legyenek megjelenítve az adatbázisból a kért HTML Tag-ek között. Illetve van egy setRefresh funkció, ami azt a célt szolgálja, hogyha a becenek esetén módosítás, vagy törlés történik, akkor az aktuális adatokkal látszodjon a lista.

A Dolgozok.js file-ba található megírva az a funkció, ami az adatbáziból a dolgozókat listázza a backend segítségével. A megírt adatbázis lekérések alapján. Első lépésben beimpotáltuk a useStaste és useEffect react funkciókat az import {useState,useEffect} from 'react'; kóddal. A szükséges elemek behívása után az lent leírt kóddal kellett megalkotni, hogy a frontend oldalt megjelenjenek a kért adatok a backend segítségével:
A function Dolgozok() {const [dolgozok,setDolgozok]=useState([]); useEffect(()=>{fetch('http://localhost:8000/dolgozok').then(adatok=>adatok.json()).then(adatok=>setDolgozok(adatok)).catch(err=>console.log(err));},[]); return (<h3>Dolgozók részletes adati:</h3>
</br>{dolgozok.map((elem)=>(<h5>
Teljes név: {elem.eredetinev}, becenév: {elem.becenev}, életkor: {elem.eletkor}, belépés dátuma: {elem.belepesdatum}, munkakör: {elem.munkakor}.
</h5>))});}export default Dolgozok; kód szolgálja ki az adatbekérést funkciót.

A létrehozott kóddal function-ban használt useState-el hoztuk létre a setDolgozok elemet, ami alapjául szolgált az adatbehívásban. Az useEffect-et használva a fetch,then,catch elemekkel adtuk meg, hogy az http://localhost:8000/dolgozok elérhetőségen kérje le az adatokat, JSON formátumban legyenek listázva a setBecenevek-be, és hiba esetén legyen egy hiba (error) üzenet.

A return({..}); résznél írtuk meg becenevek.map((elem)=>() funkcióval, hogy a .map segítségével a megadott elem-ek legyenek megjelenítve az adatbázisból a kért HTML Tag-ek között.

Az Elerhetoseg.js file-ba található a funkció, ami a munkavállalók elérhetőségét listázza a backend segítségével. A megírt adatbázis lekérések alapján. Első lépésben beimpotáltuk az useStaste és useEffect react funkciókat az import {useState,useEffect} from 'react'; kóddal. A szükséges elemek behívása után az lent leírt kóddal kellett megalkotni, hogy a frontend oldalt a menüsávban megjelenjenek a kért adatok a backend segítségével:
A function Elerhetoseg(){const[elerhetoseg,setElerhetoseg]=useState([]); useEffect(()=> {fetch('http://localhost:8000/elerhetoseg').then(adatok=>adatok.json()).then(adatok=>setElerhetoseg(adatok)).catch(err=>console.log(err));},[]); return(<h3>Dolgozók elérhetősége:</h3>
</br>{elerhetoseg.map((elem)=>(
<h5>Dolgozó teljes neve: {elem.eredetinev}, Dolgozó telefonszáma: {elem.telefonszam},</h5>))});} export default Elerhetoseg; kód szolgálja ki az adatbekérést funkciót.

A létrehozott kóddal function-ban használt useState-el hoztuk létre a setElerhetoseg elemet, ami alapjául szolgált az adatbehívásban. Az useEffect-et használva a fetch,then,catch elemekkel adtuk meg, hogy az http://localhost:8000/elerhetoseg elérhetőségen kérje le az adatokat, JSON formátumban legyenek listázva a setBecenevek-be, és hiba esetén legyen egy hiba (error) üzenet. A return({..}); résznél írtuk meg becenevek.map((elem)=>() funkcióval, hogy a .map segítségével a megadott elem-ek legyenek megjelenítve az adatbázisból a kért HTML Tag-ek között.

A Footer.js file-ba a láblécet hoztuk létre a function Footer({szoveg1,szoveg2,szoveg3}) {return (...)} export default Footer; kóddal vannak létrehozva a HTML tagek a konténeren belül, amelyek beimportálva az App.js file-ba az általunk létrehozott konténerbe beírt adatokat tartalmazza.

Továbbá a lábléc tartalmaz egy külső hivatkozást a Adatbázis szerverének elérhetősége Tag-ek közt, ami az adatbázis szerver oldalára vezet minket.

A Header.js file-ba a fejlécet hoztuk létre a function Header({szoveg1,email,szoveg2}) {return (...)} export default Header; kóddal vannak létrehozva a HTML tagek a konténeren belül, amelyek beimportálva az App.js file-ba az általunk létrehozott konténerbe beírt adatokat tartalmazza.

Az Munkavallalok.js file-ba található az a funkció, ami a munkavállalókat listázza a backend segítségével. A megírt adatbázis lekérések alapján. Első lépésben beimpotáltuk az useStaste és useEffect react funkciókat az import {useState,useEffect} from 'react'; kóddal.

A szükséges elemek behívása után az lent leírt kóddal kellett megalkotni, hogy a frontend oldalt a menüsávban megjelenjenek a kért adatok a backend segítségével:
A function Munkavallalok(){const[munkavallalok,setMunkavallalok]=useState([]); useEffect(()=> {fetch('http://localhost:8000/munkavallalok').then(adatok=>adatok.json()).then(adatok=>setMunkavallalok(adatok)).catch(err=>console.log(err));},[]); return(<h3>Munkavállalók adatai:</h3>
</br>{munkavallalok.map((elem)=>(
<h5>Munkakör: {elem.munkakor}, Teljes név: {elem.eredetinev}</h5>))});} export default Munkavallalok; kód szolgálja ki az adatbekérést funkciót.

A létrehozott kóddal function-ban használt useState-el hoztuk létre a setMunkavallalok elemet, ami alapjául szolgált az adatbehívásban. Az useEffect-et használva a fetch,then,catch elemekkel adtuk meg, hogy az http://localhost:8000/munkavallalok elérhetőségen kérje le az adatokat, JSON formátumban legyenek listázva a setBecenevek-be, és hiba esetén legyen egy hiba (error) üzenet.

A return({..}); résznél írtuk meg becenevek.map((elem)=>() funkcióval, hogy a .map segítségével a megadott elem-ek legyenek megjelenítve az adatbázisból a kért HTML Tag-ek között.

Az Ujbecenev.js components file-ban vannak megírva azok a kódok, amik lehetővé teszik, egy teljesen új adat felvitelélt a webes felületen keresztül az adatbázisunkba. A kód megírása elején az import {useState} from 'react'; behívása volt szükséges. Létre kellett hoznunk a címét, a beviteli mezőt, és a gombot funkcióival együtt.

A function Ujbecenev-ben lett egyszerre megírva a grafikus és funckionális része a file-nak, ami a kövezkető:
A function Ujbecenev() {const [becenev,setBecenev]=useState(''); const sendAdat=async(adat)=>{const res=await fetch('http://localhost:8000/ujbecenev',{method:"POST", headers:{"Content-type":"Application/json"}, body:JSON.stringify(adat)});const valasz=await res.json(); alert(valasz.message);} const onSubmit=(e)=>{e.preventDefault();sendAdat({becenev}); setBecenev('');} return (<form onSubmit={onSubmit}>
</br><label for="becenev" className="form-label"><h3>Új becenév</h3> </label>
</br><input type="text" class="form-control" id="becenev" placeholder="új becenévnek a bevitele" onChange={(e)=>setBecenev(e.target.value)} value={becenev}/>
</br> <button type="submit">Új adat belüldése</button> </form>);} export default Ujbecenev; kód szolgálja ki az új adat beviteli funkciót.

Az useState segítségével létrehoztuk a setBecenev elemet, ahol egy plusz sendAdat funckió létrehozásával megírtuk, hogy az await fetch-el a http://localhost:8000/ujbecenev címre küldje az új bevitt adatot. Ezen belül POST REST API utasítással, és a JSON.stringify és Content-type":"Application/json megjelenési formával adtuk meg a kód algoritmusát. Az await res.json();alert(valasz.message);-el fog írni a program, a bevitt értékek esetén. Az onSubmit=(e)=>{e.preventDefault();sendAdat({becenev});setBecenev('');-el adtuk meg a return-on belül létrehozott formai elem funkcióját. A return-on belül HTML Tag-ek között van megírva az összes formai elem, a cím label elemmel, a beviteli mező input elemmel, és a gomb button elemmel. A form-on belül az onSubmit-al, és az inputon belül az onChange-el együtt működik az új elem bevitele funkció.

Az index.html file-ban található pár tényezője a honlapunknak, amit külön kellett kezelni az App-js- től. A honlap megjelenési ikonja itt van megadva a <link rel="icon" href="%PUBLIC_URL
%/favicon.ico" /> Tag között. A <title>Munkaerő nyilvántartó</title> Tag között található a honlap felirati címe. A body-n belül van a konténer létrehozva, amely a body-ra vonatkozó összes adatot tárolja, többek közt itt lehet változtatni a fő háttér színeken is. A head részben található adatok közt a programcsomag telepítésekor automatikusan beállítódtak a honlaphoz szüksége reszponzív elemek.

Webes felületen látható elemek és funkciók működése

A véglegesre megírt fő három rész után (adatbázis, backend, frontend) a webes program megjelenési formájáról szeretnénk részletesebben írni. Amint a XAMPP-on belül el van indítva az Apache és MySQL modul, a backend és frontend el van indítva a belső terminálban a npm start paranccsal, akkor az alkalmazás az alapértelmezett böngészőn, a localhost:3000 URL alatt meg fog nyílni. Itt láthatóak lesznek, azok az információk, amit az App.js-ben írtunk meg. A Bootstrap és behívott CSS fájlok (App.css, index.css) segítségével, vannak formázva a látható konténerek és a HTML elemek. A sárga színnel kiemelt rész maga a navigációs menüsáv az előre megírt JavaScript components mappában található funkciókkal. Itt lehet látni, és kipróbálni, hogy miként működik a külső adatbázis a becenév, a dolgozók, a munkavállalók és az elérhetőségek adatainak behívása a kért szűrések alapján. Illetve a becenevek részén a menüben törölhetők, módosíthatók, és fel is lehet új becenév elemet vinni az adatbázisba. Fontos, hogy a Munkavállalók nyilvántartása gombra kattintva a navigáció mindig a fő oldalra küldi vissza a honlapot, ami a frontend megnyitása utána látszódik rögtön a NavLink React funkció segítségének köszönhetően.

Szoftver tesztelése és dokumentációja

A projektünk egyik tesztelő programját C# nyelven írtuk meg a Visual Studio 2019 nevű fejlesztői környezet alatt. A másik teszt a backend részen leírt client.rest file-ban találhatóak. Bővebb információk a szoftver teszt nevű mappában lehet találni a teszt futtatásokról. A megírt tesztek mellé tartozik egy külön dokumentáció, ami leírja a tesztelő programok céljait, és eredményeit. Fontos megjegyezni, hogy a teszt futtatása csak, abban az esetben működik teljesen, hogyha a XAMPP-on belül el van indítva az Apache és MySQL modul, a backend és frontend el van indítva a belső terminálban a npm start paranccsal. Ennek hiányában nem fog rendesen működni a tesztelés.

Összefoglalás

A szoftver dokumentációnk végén szeretnénk elmondani jövőbeli terveinket a programunk fejlesztésével kapcsolatosan. Szeretnénk a jövőben szerzett új programozási tudásunkat, ebben a projektben kihasználni fokozatosan, hogy szemmel látható legyen az, hogy egy bővíthető alkalmazást hoztunk létre. Továbbá fel fogjuk keresni a potenciális fejvadász cégeket, hogy bemutassuk nekik ötletünket, mert ez a formája az adatok tárolásnak, és kimutatása igen hasznos lehet a jövőbeli ügyfeleik számára, akik rövid idő alatt minél több információt szeretnének megtudni, azokról a bérmunkásokról, akiket ideiglenesen alkalmazni szeretnének a vállalatuk leterheltsége miatt.

Irodalomjegyzék, hivatkozásjegyzék

Felhasznált programok, és fejlesztői környezetek elérhetősége:

Visual Studio Code:
https://code.visualstudio.com/ Node.JS:
https://nodejs.org/en/
Visual Studio 2019: https://visualstudio.microsoft.com/downloads/
XAMPP: https://www.apachefriends.org/index.html

Felhasznált programozási nyelveknél ötletet adó oldalak:

C#: https://www.w3resource.com/index.php
WEB: https://www.w3schools.com/
Bootstrap: https://getbootstrap.com/
JavaScript: https://reactjs.org/
SQL: https://academy.oracle.com/en/oa-web-overview.html
MySQL: https://www.w3schools.com/