ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці ІІ

Кафедра математики та інформатики

Реєстраційний №_____

Кваліфікаційна робота

Розробка українсько-угорського (угорсько-українського) математичного онлайн-словника для навчання студентів-математиків.

Ковач Іштван Іштванович

Студент IV-го курсу

Освітня програма 014 «Середня освіта (Математика)»

Ступінь вищої освіти: бакалавр

Тема затверджена Вченою радою ЗУІ

Протокол **№ 10 від 27 жовтня 2021 року**

Науковий керівник: **Головач Йожеф Ігнацович**

доктор технічних наук, професор

Завідувач кафедрою математики та інформатики: **Кучінка Каталін Йожефівна**

к. ф.-м. н

Робота захищена на оцінку _____, «___» _____ 202_ року

Протокол № _____ / 202_

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

Закарпатський угорський інститут ім. Ференца Ракоці II

Кафедра математики та інформатики

Кваліфікаційна робота

Розробка українсько-угорського (угорсько-українського) математичного онлайн-словника для навчання студентів-математиків.

Ступінь вищої освіти: бакалавр

Виконав: студент IV-го курсу

Ковач Іштван Іштванович

Освітня програма 014 «Середня освіта (Математика)»

Науковий керівник: **Головач Йожеф Ігнацович**

Д.т.н., професор

Рецензент: **Міца О.В.**

завідувач кафедри ІУСТ УжНУ, Д.т.н., доцент

Берегове
2022

Зміст

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол № „3” від „27” квітня 2021 р.
Ф-КДМ-2

II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola

Matematika és Informatika Tanszék

UKRÁN-MAGYAR (MAGYAR-UKRÁN) MATEMATIKAI ONLINE SZÓTÁR FEJLESZTÉSE A MATEMATIKUS HALLGATÓK SZÁMÁRA

Szakdolgozat

Képzési szint: alapképzés

Készítette: Kovács István

IV. évfolyamos hallgató

Képzési program: 014 „Középiskolai oktatás (Matematika)”

Témavezető: Holovács József

műszaki tud. doktora, professzor

Recenzens: Mitsa O. V.

UNE IVRT tanszék tanszékvezetője,

műszaki tud. doktora, docens

Beregszász – 2022

Tartalomjegyzék

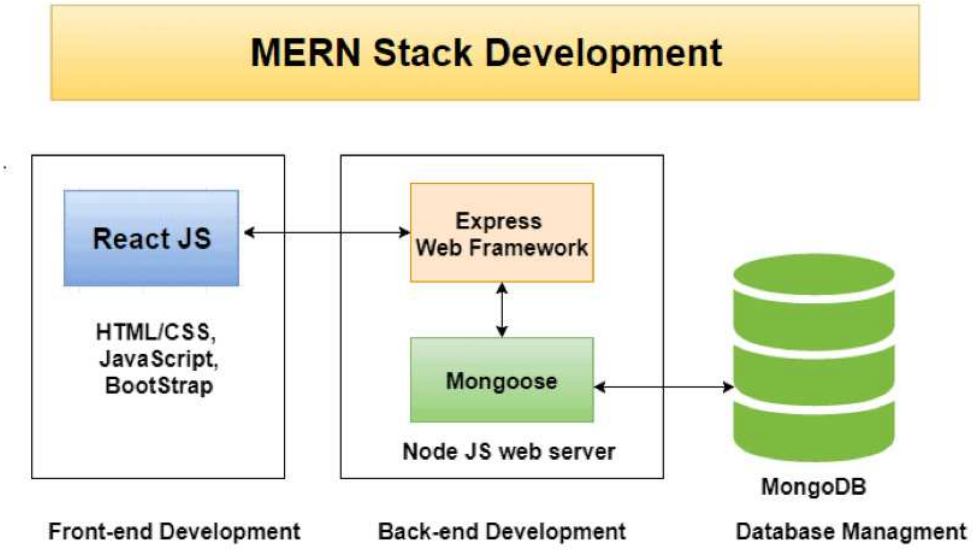
Bevezetés

Dolgozatom célja modern informatikai eszközökkel támogatni az ukrajnai magyar iskolákban a matematika oktatását, egy könnyen kezelhető, online elérhető, autentikáció és autorizációval rendelkező ukrán-magyar matematikai szótár fejlesztésével. Erre a célra a MERN veremet választottam.

Ez egy olyan webfejlesztési keretrendszer, amely a MongoDB, az Express.js, a React.js és a Node.js halmazából áll. Az egész MERN veremnek három rétege van: adatbázis-kezelés, back-end és front-end fejlesztés. MongoDB biztosítja az adatbázis-kezelést, az Express és a Node felel a az alkalmazási rétegért (back-end), a bemutató réteg (front-end) React segítségével valósul meg. A MERN verem népszerűsége töretlen, mivel rendkívül erős, nagymértékben skálázható technológiák gyűjteményéből áll.

1. MERN Stack

A MERN Stack a MongoDB, az Express, a React.js és a Node.js rövidítése, ezek mind nyílt forráskódú szoftverek amelyek egyoldalas alkalmazások tervezésére vannak optimalizálva (Single Page Application). Ez a MERN Stack együtt tartalmazza az adatbázist, a back-end keretrendszert, a front-end könyvtárat és a szerver-oldali runtime környezetett. A Node.js lehetővé teszi, hogy a MERN stack nagyon hatékony legyen ún. hosting web szolgáltatóként, mivel rengeteg ügyfelet képes kiszolgálni egyidejűleg. A MERN stack a JavaScript nyelv használatával lett programozva, emiatt nagy előnye a fejlesztőknek, hogy csak egyetlen programozási nyelvet kell megfelelőképpen használni. Eltekintve ettől az előnytől, hogy nem kell nyelvet váltani, amikor az alkalmazásunk különböző részein dolgozunk, még a ún. kód megosztást is lehetővé teszi az alkalmazás egész területén. [1]



1.1. ábra A MERN Stack szerkezete

[1] https://www.mongodb.com/mern-stack

1.1. MongoDB

A MongoDB egy dokumentum-orientált NoSQL adatbázis, amelyet nagy mennyiségű adattárolásra használnak. Ahelyett, hogy táblákat és sorokat használna, mint a hagyományos relációs adatbázisok, a MongoDB gyűjteményeket és dokumentumokat használ. A dokumentumok kulcs-érték párokból állnak, amelyek a MongoDB alapvető adategységei. A gyűjtemények a MongoDB számára lehetővé teszik a gyűjteményben lévő dokumentumok rendezését és nyomon követését, és megakadályozzák a névfedést is. A MongoDB minden egyes dokumentumának egyedi objektumazonosítója van, amelyet automatikusan generál, amelyen keresztül a dokumentum elérhető [2]. A MongoDB JavaScript nyelven, a lekérdezés nyelve pedig a JSON-on alapul (JavaScript Object Notation).

Mi az a JSON?

A JavaScript Object Notation-t, közismertebb nevén JSON-t, a JavaScript-nyelv részeként határozta meg a 2000-es évek elején Douglas Crockford, a JavaScript alkotója, bár a formátumot csak 2013-ban határozták meg hivatalosan. A JavaScript-objektumok egyszerű asszociatív tárolók, amelyekben egy karakterlánc-kulcs egy értékre van leképezve (amely lehet szám, karakterlánc, függvény vagy akár egy másik objektum). Ez az egyszerű nyelvi tulajdonság lehetővé tette a JavaScript objektumok rendkívül egyszerű szöveges megjelenítését: A JSON használata lekérdezési nyelvként jelentősen megkönnyíti a lekérdezés használatát. A művelet típusának megadásával egy JSON objektumban dokumentum létrehozható, törölhető, visszakereshető, frissíthető vagy kereshető, és lehetővé teszi a fejlesztő számára, hogy további programozási lekérdezés-eket készítsen. A MongoDB egy olyan adatbázis, amely a 2000-es évek közepe táján került napvilágra [3].

Mi az a BSON?

A BSON egyszerűen a „bináris JSON" rövidítése, és pontosan erre találták ki. A BSON bináris struktúrája típus- és hosszinformációkat kódol, ami lehetővé teszi azok sokkal gyorsabb elemzését. A kezdeti megfogalmazása óta a BSON-t

[2] https://www.mongodb.com

[3] https://www.mongodb.com/json-and-bson

kibővítették néhány opcionális, nem JSON-natív adattípussal, például dátumokkal és bináris adatokkal, amelyek nélkül a MongoDB nem kapott volna értékes támogatást. Azok a nyelvek, amelyek bármilyen összetett matematikát támogatnak, általában különböző méretű egész számokkal (ints vs longs) vagy különböző szintű decimális pontossággal rendelkeznek (float, double, decimal128, stb.). Nemcsak az hasznos, hogy a MongoDB-ben tárolt adatokban ábrázolni tudjuk ezeket a különbségeket, hanem lehetővé teszi az összehasonlítások és számítások elvégzését közvetlenül az adatokon, oly módon, hogy leegyszerűsítsük az alkalmazáskódok felhasználását.

Az adatokat natív módon BSON (Binary JSON) formájában tároljuk.Ennek az az oka, hogy a MongoDB támogatja a különféle adattípusok beillesztését különböző programozási nyelvekre, mivel a nyelv illesztőprogramja támogatott. Ezután az illesztőprogram átalakítja az alkalmazásokban használt különböző adattípuso-kat (amelyek különböző programozási nyelveken írhatók) BSON típusokká. Ez a kialakítás gyorsabb vizsgálatot tesz lehetővé lekérdezéskor, és növeli az adattárolás hatékonyságát [3].

[3]https://www.mongodb.com/json-and-bson

1.2. Express

Az Express.js vagy egyszerűen az Express a Node.js háttér-webalkalmazás-keretrendszere, amely ingyenes és nyílt forráskódú szoftver amelyet eredetileg TJ Holowaychuk épített. Egyszerűsíti a szerveroldali kód felépítését. Bár ezt gyakorlatilag egyedül a Node.js-sel is meg lehet tenni, de az Express abban segít nekünk, hogy kevesebb és tisztább kóddal ugyanazt az eredményt érjük el, miközben további funkciókhoz férünk hozzá. Az Express lényegében egy köztes szoftverkészletként írható le, amely a webalkalmazástól elvárt összes válasz kezelésére épül, például válaszkódok, sütik beállítása vagy egyedi fejlécek küldése.[4]

A keretrendszer kiemelkedő jellemzője az útválasztás, annak meghatározása, hogy az alkalmazás végpontjaiban (URL-ben) bizonyos mintáknak megfelelő HTTP-kéréseket hogyan lehet feldolgozni. Lényegében elemzi a kérés URL-jét, a fejléceket, a paramétereket, és a megadott módon válaszol. [5] Az Express template motorok teszik lehetővé a statikus sablonfájlok használat-át az alkalmazásban. A sablonokban szereplő változókat a tényleges értékek helyettesítik és alakítják át a HTML fájlba, mielőtt visszaküldenék azt a kliensnek. Ez a HTML oldal egyszerűbb dízájnolását kínálja. Az Express a népszerű fejlesztési stack-ek, például a MEAN, a MERN vagy a MEVN stack back-end-jének összetevője, a MongoDB adatbázis-szoftverrel és a JavaScript front-end keretrendszerrel vagy könyvtárral együtt. [6]

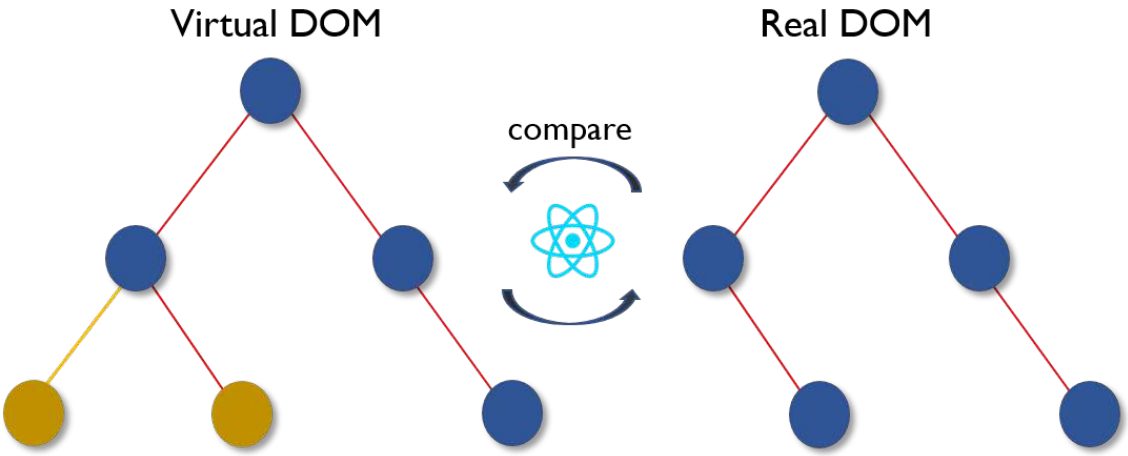
[4]https://www.oreilly.com/library/view/web-development-with/9781491902288/ch01.html
[5]https://expressjs.com/
[6]https://expressjs.com/en/guide/using-template-engines.html

1.3. React.js

A React.js egy kliensoldali szoftver, ez egy nyílt forráskódú JavaScript komponens alapú könyvtár, amelyet a Facebook fejlesztett ki.[7] Eltérően a többi JavaScript könyvtártól, mint a jQuery, a React.js csakis a nézet (view) rétegre fókuszál.

A React.js-ben a nézetek deklaratívak. A deklaratív nézet teszi lehetővé a kód kiszámíthatóságát és könnyebb karbantartását, mivel a fejlesztőnek csak minden nézethez meg kell terveznie az összetevőt (komponenst), a React.js pedig csak azt az összetevőt frissíti hatékonyan, amelyben a változás bekövetkezett. Ez a virtuális DOM (Document Object Model), amely egy memóriában lévő adatstruktúra, amely a nézet deklarálásakor az adatokat ábrázolja. A React.js összehasonlítja a virtuális DOM-ot a tényleges DOM-mal, és ha van különbség, akkor a tényleges DOM ennek megfelelően frissül. [8]

A komponensek a React.js építőkövei, ezek kisebb és újrafelhasználható



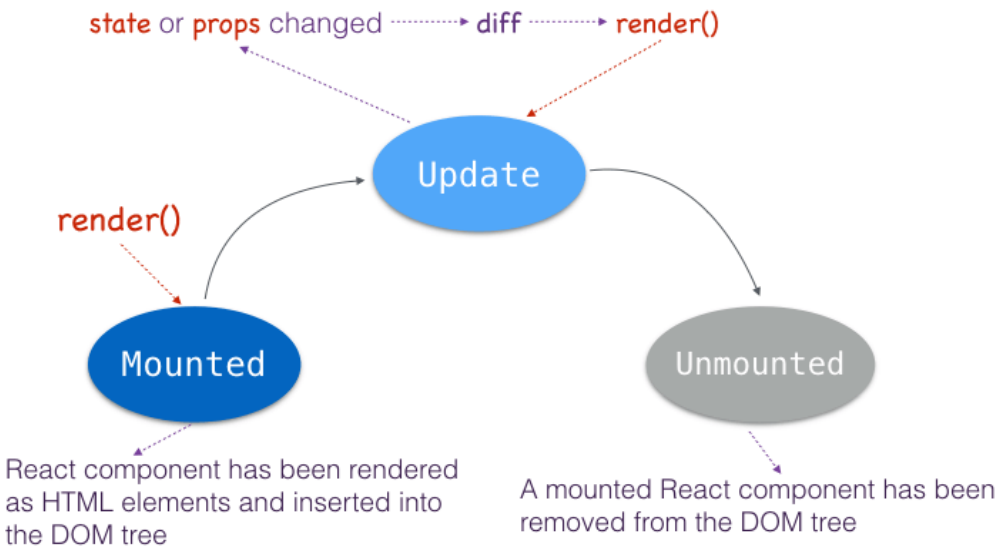
1.2. ábra A virtuális DOM és a tényleges DOM

kódrészletek, amelyek képesek fenntartani a saját állapotukat (state) és megjelenítésüket. Gyakorlatilag ezek olyan JavaScript osztályok vagy függvények, amelyek képesek bemenetek és tulajdonságok (props) fogadására, és egy React.js elem visszaadására. Ezen React.js elemek mindegyike leírja, hogyan kell kinéznie az UI-nak (User Interface, Felhasználói Felület). A komponenseket összerakhatjuk másik komponensekkel, hogy ún. szülőkomponenseket alkossunk, amely már megjelenítheti a

[7] https://hu.reactjs.org/
[8] https://github.com/facebook/react/

teljes nézetét vagy egy oldalát az alkalmazásunknak.

Az adatok a tulajdonságok (props) segitségével cserélhetők a komponensek között, és a komponensek állapotában (state) tárolhatók. Minden komponensnek megvan a saját maga életciklusa, egy eseménysorozat, amelyen a komponens átmegy a beépítésétől egészen a leszereléséig. A komponensben lévő adatok szükség szerint kezelhetőek az életciklus alatt. [9]



1.3. ábra Életciklus folyamatábra

[9] https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle

1.4. Node.js

A Node.js egy JavaScript runtime környezet. Maga a JavaScript csak egy programozási nyelv, a JavaScript kód kiszolgálóoldali végrehajtásához runtime környezetre van szükség. A Node.js runtime környezet a JavaScript további funkcióiból áll, beleértve a fájlrendszerhez: a fájlok olvasása és írása a fájlrendszerben JavaScript használatával és a hálózathoz való hozzáférést. JavaScript használatával http kéréseket lehet küldeni és válaszokat fogadni. Ez azt jelenti, hogy a Node.js olyan környezetben írt JavaScript kódot képes végrehajtani, ahol hozzáféréssel rendelkezik azokhoz a funkciókhoz, amelyek nem részei a JavaScript nyelvnek.

A Node.js modulok teszik lehetővé a JavaScript fájlok exportálását és importálását egy másik fájlba, mindaddig, amíg a modul specifikációit betartják. A Node.js modulok lehetővé teszik a fejlesztő számára, hogy a kódokat kisebb és újrafelhasználható darabokra bontsa, és jobb struktúrát és szervezést kínáljon a kód számára. Ezeket a modulokat könyvtáraknak tekintjük, amelyek egy másik JavaScript fájlba írt függvényeket használnak. A Node.js tartalmaz néhány beépített modult, amelyek hozzáférnek a fájlrendszerhez és a hálózati rendszerhez. [10] Harmadik féltől származó könyvtárak is letölthetők és telepíthetők az NPM (Node Package Manager) használatával, amely a Node alapértelmezett csomagkezelője. [11]

[10] https://nodejs.dev/learn

[11] https://www.javascripttutorial.net/nodejs-tutorial/what-is-npm/

2. CRUD alkalmazás

2.1. Backend

Első lépésben létrehozunk egy mappát, és terminálon keresztül elinditjuk az

npm init

parancsot, ezzel a lépéssel egy üres npm projektet hozunk létre.

Telepítenünk kell a szükséges modulokat: express, mongoose, body-parser, cors. Lefutattjuk a következő parancsot:

npm install express mongoose body-parser cors –save

[12] **Az Express web szerver felépítése**

A gyökérkönyvtárunkban létrehozunk egy új *server.js* fájlt:

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

const db = require("./app/models");
db.mongoose
  .connect(db.url, {
```

[12] https://bezkoder.com/node-express-mongodb-crud-rest-api/

```
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Kapcsolat létrehozva az adatbázissal!");
  })
  .catch(err => {
    console.log("Nem sikerült kapcsolódni az adatbázishoz!", err);
    process.exit();
  });

app.get("/", (req, res) => {
  res.json({ message: "Welcome to my application." });
});

require("./app/routes/szotar.routes")(app);

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(‘Server is running on port ${PORT}.‘);
});
```

Importáltuk az express, body-parser és cors modulokat.

Az Express a rest api-k építésére szolgál.

A body-parser segít elemezni a kéréseket, és létrehozza a req.body objektumot.

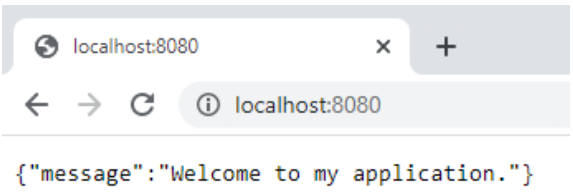
A cors Express köztes szoftvert (middleware) biztosít a CORS számára különféle opciókkal.

Létrehoztunk egy Express alkalmazást és hozzáadtuk a body-parser és cors köztes szoftvereket az *app.use ()* módszerrrel.

Az alkalmazás a 8080-as porton hallgassa meg a beérkező kéréseket.

Meghatároztunk egy GET útvonalat, amely a teszteléshez kell.

Futassuk le az alkalmazásunkat a következő paranccsal: *node server.js*
Nyissuk meg a böngészőnket a http://localhost:8080 URL-n.



2.1. ábra Képernyőmentés az alkalmazásról

[12] MongoDB adatbázis és Mongoose beállítása

Az *app* könyvtárban létrehozunk egy *config* mappát, benne egy *db.config.js* fájllal:

```
module.exports = {
  url: "mongodb://localhost:27017/evfolyammunka"
};
```

A következő lépésben meghatározzuk a Mongoose modelt (*szavak.model.js*), de előtte az *app/models* könyvtárban létrehozunk egy *index.js* fájlt a következő kóddal:

```
const dbConfig = require("../config/db.config.js");

const mongoose = require("mongoose");
mongoose.Promise = global.Promise;

const db = {};
db.mongoose = mongoose;
db.url = dbConfig.url;
db.szotar = require("./szavak.model.js")(mongoose);

module.exports = db;
```

A *models* könyvtárban létrehozzuk a *szavak.model.js* fájlt,de mivel mi az alkalmazásunkat frondend-en is szeretnénk használni, emiatt felülírjuk a *toJSON* metódust, amely az alapértelmezett objektumot hozzárendeli egy egyedi objektumhoz. Tehát igy fog kinézni a Mongoose modellünk:

```
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      title: String,
      description: String,
```

[12] https://bezkoder.com/node-express-mongodb-crud-rest-api/

```
      published: Boolean
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const Szavak = mongoose.model("szotar", schema);
  return Szavak;
};
```

Ezután már nem kell CRUD függvényeket írni, mivel a Moongose modell tartalmazza azokat.

[12] **Létrehozzuk a Vezérlőt**

Az *app/controllers* könyvtáron belül létrehozzuk a *szavak.controller.js* fájlt benne a CRUD függvényekkel(*create, findAll, findOne, update, delete, deleteAll, findAllPublished*):

```
const db = require("../models");
const Szavak = db.szotar;


exports.create = (req, res) => {
  if (!req.body.title) {
    res.status(400).send({ message: "A tartalom nem lehet üres!" });
    return;
  }


  const szavak = new Szavak({
    title: req.body.title,
    description: req.body.description,
    published: req.body.published ? req.body.published : false
  });


  szavak
    .save(szavak)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
```

[12]https://bezkoder.com/node-express-mongodb-crud-rest-api/

```
        message:
          err.message || "Hiba lépett fel a szavak készítése közben."
      });
    });
};


exports.findAll = (req, res) => {
  const title = req.query.title;
  var condition = title ? { title: { $regex: new RegExp(title), $options: "i" }
} : {};

  Szavak.find(condition)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Hiba lépett fel a szavak lekérése közben."
      });
    });
};


exports.findOne = (req, res) => {
  const id = req.params.id;

  Szavak.findById(id)
    .then(data => {
      if (!data)
        res.status(404).send({ message: "Nem található ilyen szó ezzel
az ID-vel: " + id });
      else res.send(data);
    })
    .catch(err => {
      res
        .status(500)
        .send({ message: "Hiba történt a szó lekérdezése közben ezzel
az ID-vel:" + id });
    });
};

exports.update = (req, res) => {
  if (!req.body) {
    return res.status(400).send({
      message: "A frissítendő adatok nem lehetnek üresek!!"
    });
  }

  const id = req.params.id;
```

```
  Szavak.findByIdAndUpdate(id, req.body, { useFindAndModify: false })
    .then(data => {
      if (!data) {
        res.status(404).send({
          message: `Nem lehet frissíteni a szót ezzel az ID-vel: ${id}.
Lehet nincs ilyen szó!`
        });
      } else res.send({ message: "A szó sikeresen frissítve lett." });
    })
    .catch(err => {
      res.status(500).send({
        message: "Hiba történt a szó frissitésekor a következő ID-vel:" + id
      });
    });
};

exports.delete = (req, res) => {
  const id = req.params.id;

  Szavak.findByIdAndRemove(id, { useFindAndModify: false })
    .then(data => {
      if (!data) {
        res.status(404).send({
          message: `Nem törölhető ezzel az ID-vel rendelkező szó: ${id}.
Lehet nincs ilyen szó!`
        });
      } else {
        res.send({
          message: "A szó sikeresen törölve lett!"
        });
      }
    })
    .catch(err => {
      res.status(500).send({
        message: "Nem sikerült törölni a szót ezzel az ID-vel: " + id
      });
    });
};

exports.deleteAll = (req, res) => {
  Szavak.deleteMany({})
    .then(data => {
      res.send({
        message: `${data.deletedCount} A szavak sikeresen törölve lettek!`
      });
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Valami hiba történt az összes szó eltávolítása közben."
```

```
      });
    });
};


exports.findAllPublished = (req, res) => {
  Szavak.find({ published: true })
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Hiba történt a szavak beolvasása közben."
      });
    });
};
```

[12] Útvonalak beállítása

Amikor a kliens végpont kérést küld, HTTP-kérés (*get,post,put, delete*) felhasználásával, meg kell határozni, hogyan reagáljon erre a szerver, az útvonalak beállításával. Létrehozzuk az *app/routes* könyvtáron belül a *szotar.routes.js* fájlt a következő tartalommal:

```
module.exports = app => {
  const szotar = require("../controllers/szavak.controller.js");

  var router = require("express").Router();

  router.post("/", szotar.create);

  router.get("/", szotar.findAll);

  router.get("/published", szotar.findAllPublished);

  router.get("/:id", szotar.findOne);

  router.put("/:id", szotar.update);

  router.delete("/:id", szotar.delete);

  router.delete("/", szotar.deleteAll);

  app.use("/api/szotar", router);
};
```

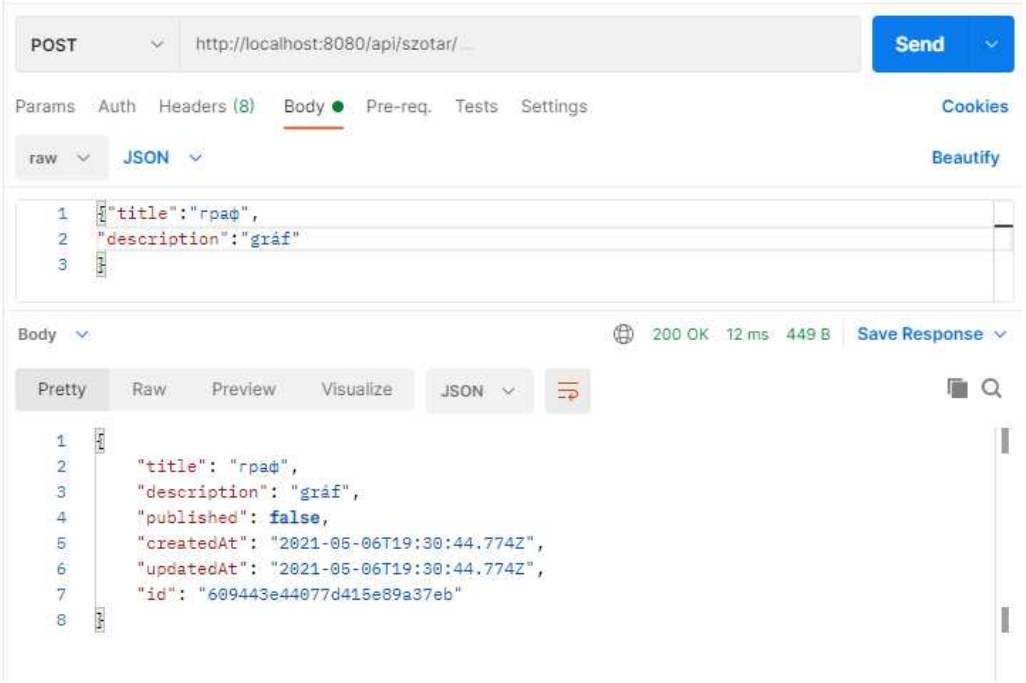
Ezzel a lépéssel meg is van a backend, a *node server.js* paranccsal futatthatjuk az alkalmazásunkat.

[12] https://bezkoder.com/node-express-mongodb-crud-rest-api/

A Postman népszerű API klienst használva létrehozhatunk, megoszthatunk, tesztelhetünk és dokumentálhatunk API-kat.

Mi az a Postman API?

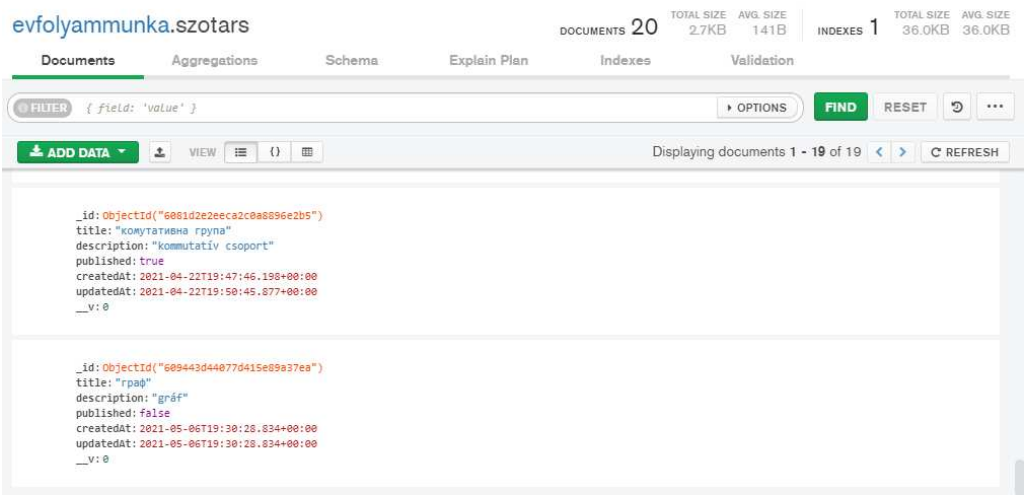
A Postman egy API-kliens, amely megkönnyíti a fejlesztők számára az API-k létrehozását, megosztását, tesztelését és dokumentálását. Ez úgy történik, hogy a felhasználók egyszerű és összetett HTTP/s kéréseket hozhatnak létre és menthetnek, valamint elolvashatják válaszaikat. [13].



2.2. ábra Képernyőmentés a Postmanról

[13]https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis

Miután létrehoztuk a szót, ellenőrizzük a MongoDB adatbázisunkat.



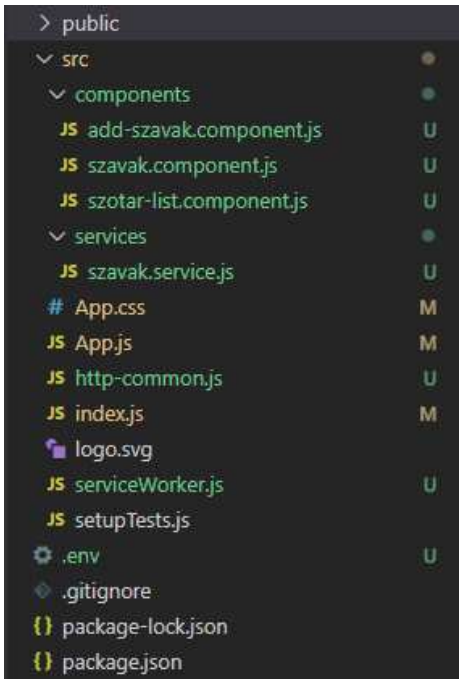
2.3. ábra Képernyőmentés a MongoDBről

2.2. Frontend

Létrehozunk egy mappát, azon belül megnyitjuk a terminált és a következő paranccsal létrehozzuk az alkalmazásunkat:

npx create-react-app react-crud

Miután a folyamat végbement, további mappákat és fájlokat hozunk létre.



2.4. ábra Képernyőmentés az alkalmazás struktúrájáról

[14] Bootstrap importálása az alkalmazásba

Futattjuk a következő parancsot: *npm install boostrap*

Megnyitjuk a *src/App.js* fájlt, és az alábbiak szerint módosítjuk a kódot:

```
import React, { Component } from "react";
import "bootstrap/dist/css/bootstrap.min.css";

class App extends Component {
  render() {
  }
}

export default App;
```

[14] https://www.bezkoder.com/react-crud-web-api/

[14] React Router hozzáadása az alkalmazáshoz

Lefutattjuk az alábbi parancsot: *npm install react-router-dom*

Megnyitjuk a *src/index.js* fájlt, amelynek a következőképpen kell kinéznie:

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

import App from "./App";
import * as serviceWorker from "./serviceWorker";

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);

serviceWorker.unregister();
```

Navbar hozzáadása az alkalmazáshoz

Nyissuk meg a *src/App.js* fájlt, ez a komponens lesz a gyökere a mi alkalmazásunknak, ez tartalmazni fog egy Navbart, Switch objektumot, több útvonallal (Route). Minden útvonal egy React komponenshez fog vezetni.

```
import React, { Component } from "react";
import { Switch, Route, Link } from "react-router-dom";
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";

import AddSzavak from "./components/add-szavak.component";
import Szavak from "./components/szavak.component";
import SzotarList from "./components/szotar-list.component";

class App extends Component {
  render() {
    return (
      <div>
        <nav className="navbar navbar-expand navbar-dark bg-dark">
          <Link to={"/szotar"} className="navbar-brand">
            Ukrán-magyar szótár
          </Link>
          <div className="navbar-nav mr-auto">
            <li className="nav-item">
              <Link to={"/szotar"} className="nav-link">
                Szótár
              </Link>
```

[14] https://www.bezkoder.com/react-crud-web-api/

```
            </li>
            <li className="nav-item">
              <Link to={"/szerkesztes"} className="nav-link">
                Hozzáadás
              </Link>
            </li>
          </div>
        </nav>

        <div className="container mt-3">
          <Switch>
            <Route exact path={["/", "/szotar"]} component={SzotarList} />
            <Route exact path="/szerkesztes" component={AddSzavak} />
            <Route path="/szotar/:id" component={Szavak} />
          </Switch>
        </div>
      </div>
    );
  }
}

export default App;
```

[14] Az Axios telepítése a HTTP kliens számára

A *src* mappa alatt, létrehozunk egy *http-common.js* nevű fájlt a következő kóddal:

```
import axios from "axios";

export default axios.create({
  baseURL: "http://localhost:8080/api",
  headers: {
    "Content-type": "application/json"
  }
});
```

Létrehozunk egy olyan szolgáltatást, amely ezt a fenti axios objektumot fogja használni a HTTP kérések küldéséhez. A CRUD műveletek végrehajtásához meghivjuk a HTTP kéréseknek megfelelő get,post, put, delete metódust. Módosítjuk a *services/-szavak.service.js* nevezetű fájlt a következőképpen:

```
import http from "../http-common";

class SzavakDataService {
  getAll() {
    return http.get("/szotar");
  }
```

[14] https://www.bezkoder.com/react-crud-web-api/

```
  get(id) {
    return http.get(`/szotar/${id}`);
  }

  create(data) {
    return http.post("/szotar", data);
  }

  update(id, data) {
    return http.put(`/szotar/${id}`, data);
  }

  delete(id) {
    return http.delete(`/szotar/${id}`);
  }

  deleteAll() {
    return http.delete(`/szotar`);
  }

  findByTitle(title) {
    return http.get(`/szotar?title=${title}`);
  }
}

export default new SzavakDataService();
```

Ezután 3 komponenst építünk a korábban már meghatározott 3 útvonalnak megfelőlen:

add-szavak.component.js

szavak.component.js

szotar-list.component.js

A projekt mappájában létrehozunk .env fájlt a következő tartalommal:

```
PORT=8081
```

Ezzel beállítottuk az alkalmazásunkat, hogy a 8081-es porton fusson.

Az alkalmazásunkat az *npm start* paranccsal indithatjuk el.

3. Autentikáció és autorizáció

Ebben a fejezetben bemutatom hogyan hítozható létre teljes MERN stack JWT hitelesítéssel. Ez egy bejelentkezési és regisztrációs alkalmazás lesz MongoDB + Express + React.js + Node.js alkalmazásával. Lesz egy újabb háttérszerverünk amely a Node.js Express-t jsonwebtokennel használja az autentikációhoz és az autorizációhoz. Mongoose-t fogunk használni a MongoDB adatbázissal való kapcsolathoz és a jelenlegi React front-endbe fogunk építeni új komponenseket.

3.1. JSON Web Token

Mi az a JWT?

A JWT vagy a JSON Web Token egy nyílt szabvány, amellyel két fél – egy kliens és egy szerver – megoszthatja a biztonsági információkat. Minden JWT kódolt JSON-objektumokat tartalmaz, beleértve a jogcímkészletet. A JWT-k kriptográfiai algoritmussal vannak aláírva annak biztosítására, hogy a jogkivonat kiadása után a követeléseket ne lehessen módosítani.

Mik azok a tokenek?

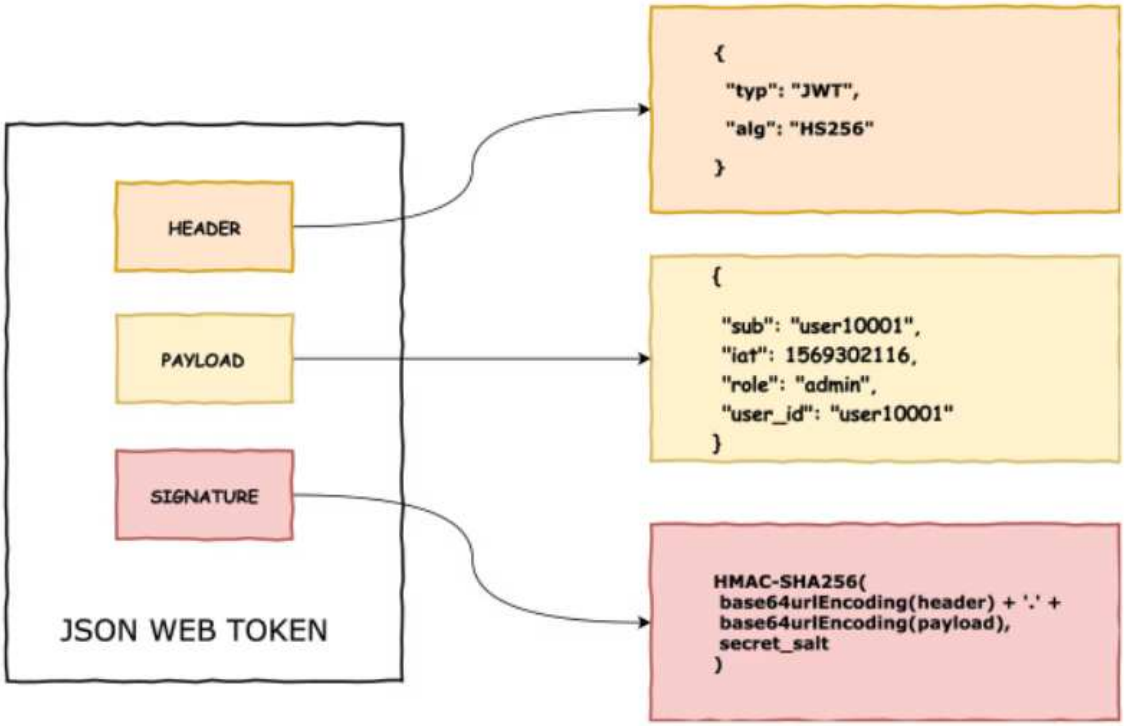
A token egy adatsor, amely valami mást, például egy identitást képvisel. Hitelesítés esetén a nem JWT alapú token egy olyan karaktersorozat, amely lehetővé teszi a fogadó számára a küldő személyazonosságának ellenőrzését. A fontos különbség itt a jelentés hiánya a karaktereken belül.[15]

Hogyan müködik a JWT?

A JWT-k abban különböznek a többi webes tokenektől, hogy követeléseket tartalmaznak. A követelések két fél közötti információtovábbításra szolgálnak. Hogy melyek ezek az állítások, az az adott használati esettől függ. Például egy követelés

[15]https://www.akana.com/blog/what-is-jwt

erősítheti, hogy ki adta ki a tokent, mennyi ideig érvényes, vagy milyen engedélyeket kapott az ügyfél. A JWT egy karakterlánc, amely három részből áll, amelyeket pontok (.) választanak el egymástól, és a base64 használatával soroznak. A legelterjedtebb szerializációs formátumban, a kompakt szerializálásban a JWT valahogy így néz ki: xxxxx.yyyyy.zzzzz.



3.1. ábra A JSON web token struktúrája

3.2. JWT Backend

Egy olyan Node.js Express alkalmazást fogunk létrehozni amelyben a felhasználó regisztrálhat új fiókot vagy már bejelentkezhet a meglévő felhasználónévvel és jelszóval. Bizonyos szerepkör (felhasználó, moderátor, adminisztrátor) szerint hozzáférhet védett tartalmakhoz.
Első lépésben létrehozunk egy mappát, ahol terminálon keresztül telepítjük a szükséges modulokat:

```
npm install express mongoose cors jsonwebtoken bcryptjs --save
```

[16]A gyökérkönyvtárban létrehozunk egy új *server.js* fájlt, itt importáljuk a nekünk szükséges expressz és cors modulokat:

```
const express = require("express");
const cors = require("cors");
const dbConfig = require("./app/config/db.config");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

const db = require("./app/models");
const Role = db.role;

db.mongoose
  .connect(`mongodb://${dbConfig.HOST}:${dbConfig.PORT}/${dbConfig.DB}`, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Sikeres kapcsolat a MongoDB-vel.");
    initial();
  })
  .catch(err => {
    console.error("Kapcsolat hiba", err);
    process.exit();
```

[16]https://www.bezkoder.com/node-js-mongodb-auth-jwt/

```
  });

app.get("/", (req, res) => {
  res.json({ message: "Üdvözöllek az alkalmazásomban." });
});

require("./app/routes/auth.routes")(app);
require("./app/routes/user.routes")(app);


  const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`A szerver ${PORT} -as porton fut.`);
});

function initial() {
  Role.estimatedDocumentCount((err, count) => {
    if (!err && count === 0) {
      new Role({
        name: "user"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("'user' hozzáadva a roles gyűjteményhez");
      });

      new Role({
        name: "moderator"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("'moderator' hozzáadva a roles gyűjteményhez");
      });

      new Role({
        name: "admin"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("'admin' hozzáadva a roles gyűjteményhez");
      });
    }
  });
}
```

MongoDB adatbázis és Mongoose beállítása

db.config.js

```
module.exports = {
  HOST: "localhost",
  PORT: 27017,
  DB: "login_db"
};
```

Kétféle mongoose modellt hozunk létre, egyet a felhasználónak másikat a szerepkörnek.

models/*role.model.js*

```
const mongoose = require("mongoose");
const Role = mongoose.model(
  "Role",
  new mongoose.Schema({
    name: String
  })
);
module.exports = Role;
```

models/*user.model.js*

```
const mongoose = require("mongoose");
const User = mongoose.model(
  "User",
  new mongoose.Schema({
    username: String,
    email: String,
    password: String,
    roles: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Role"
      }
    ]
  })
);
module.exports = User;
```

Ezután létrehozzuk az **app/models**/*index.js* fájlt a következő tartalommal:

```
const mongoose = require('mongoose');
mongoose.Promise = global.Promise;
const db = {};
db.mongoose = mongoose;
db.user = require("./user.model");
db.role = require("./role.model");
db.ROLES = ["user", "admin", "moderator"];
module.exports = db;
```

Ezen mongoose modellek az adatbázisban létrehozzák a user, roles gyűjteményeket és formai követelményt adnak meg. Tehát egy új fiók létrehozásához szükséges felhasználónév, email, jelszó, valamint meg kell adni legalább egy szerepkört (jelen esetben: admin, moderator, user).

Auth kulcs konfigurálás

A jsonwebtoken függvények, például a verify() vagy a sign() olyan algoritmust használnak, amelynek titkos kulcsra van szüksége (Stringre) a token kódolásához és dekódolásához.

Az **app/config** mappában hozzuk létre a következő *auth.config.js* fájlt:

```
module.exports = {
    secret: "login-secret-key"
  };
```

Köztesszoftver-függvények létrehozása

A regisztráció ellenőrzéséhez kettő funkcióra van szükség:

-Ellenőrizni kell a felhasználónév és az email cím ismétlődéseit.

-Ellenőrizni kell, hogy létezik-e a megadott szerepkör.

middlewares/*verifySignUp.js*

```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;
checkDuplicateUsernameOrEmail = (req, res, next) => {
  // felhasználónév
  User.findOne({
    username: req.body.username
  }).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    if (user) {
      res.status(400).send({ message: "Hiba! Felhasználónév foglalt!" });
      return;
    }
    // email
    User.findOne({
      email: req.body.email
    }).exec((err, user) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }
      if (user) {
```

```
        res.status(400).send({ message: "Hiba! Foglalt az e-mail cím!" });
        return;
      }
      next();
    });
  });
};
checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: `Hiba! A következő szerep: ${req.body.roles[i]} nem létezik!`
        });
        return;
      }
    }
  }
  next();
};
const verifySignUp = {
  checkDuplicateUsernameOrEmail,
  checkRolesExisted
};
module.exports = verifySignUp;
```

Az autorizáció és autentikáció feldolgozásához ellenőrizni kell, hogy a token jogosan áll-e a rendelkezésünkre illetve, hogy adott felhasználó szerepei tartalmazzák-e a neki szükséges szerepköröket.

middlewares/*authJwt.js*

```
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const User = db.user;
const Role = db.role;
verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ message: "Nincs megadva token!" });
  }
  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Jogtalan hozzáférés!" });
    }
    req.userId = decoded.id;
    next();
  });
};
```

```
isAdmin = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
        for (let i = 0; i < roles.length; i++) {
          if (roles[i].name === "admin") {
            next();
            return;
          }
        }
        res.status(403).send({ message: "Adminisztrátori szerepkör szükséges!" });
        return;
      }
    );
  });
};
isModerator = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
        for (let i = 0; i < roles.length; i++) {
          if (roles[i].name === "moderator") {
            next();
            return;
          }
        }
        res.status(403).send({ message: "Moderátori szerepkör szükséges!" });
        return;
      }
```

```
    );
  });
};
const authJwt = {
  verifyToken,
  isAdmin,
  isModerator
};
module.exports = authJwt;
```

middlewares/*index.js*

```
const authJwt = require("./authJwt");
const verifySignUp = require("./verifySignUp");
module.exports = {
  authJwt,
  verifySignUp
};
```

Vezérlők létrehozása

Az azonosításnak két fő funckiója van:

Regisztráció: egy új felhasználót hoz létre az adatbázisban.(Automatikusan felhasználói szerepkört kap, ha nem ad meg szerepet)

Bejelentkezés: megkeresi a felhasználónevet az adatbázisban, ha létezik akkor bcrypt segítségével összehasonlítja a az adatbázisban lévő felhasználónévvel és jelszóval, ha egyezik akkor jsonwebtoken használatával generál egy tokent és visszaadja a felhasználói információkat és a hozzáférési tokent.

controllers/*auth.controller.js*

```
const config = require("../config/auth.config");
const db = require("../models");
const User = db.user;
const Role = db.role;
var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");
exports.signup = (req, res) => {
  const user = new User({
    username: req.body.username,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8)
  });
  user.save((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    if (req.body.roles) {
```

```
      Role.find(
        {
          name: { $in: req.body.roles }
        },
        (err, roles) => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }
          user.roles = roles.map(role => role._id);
          user.save(err => {
            if (err) {
              res.status(500).send({ message: err });
              return;
            }
            res.send({ message: "Felhasználó sikeresen regisztrálva!" });
          });
        }
      );
    } else {
      Role.findOne({ name: "user" }, (err, role) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
        user.roles = [role._id];
        user.save(err => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }
          res.send({ message: "Felhasználó sikeresen regisztrálva!" });
        });
      });
    }
  });
};
exports.signin = (req, res) => {
  User.findOne({
    username: req.body.username
  })
    .populate("roles", "-__v")
    .exec((err, user) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }
      if (!user) {
        return res.status(404).send({ message: "Nincs ilyen felhasználó." });
      }
      var passwordIsValid = bcrypt.compareSync(
```

```
        req.body.password,
        user.password
      );
      if (!passwordIsValid) {
        return res.status(401).send({
          accessToken: null,
          message: "Helytelen jelszó!"
        });
      }
      var token = jwt.sign({ id: user.id }, config.secret, {
        expiresIn: 86400
      });
      var authorities = [];
      for (let i = 0; i < user.roles.length; i++) {
        authorities.push("ROLE_" + user.roles[i].name.toUpperCase());
      }
      res.status(200).send({
        id: user._id,
        username: user.username,
        email: user.email,
        roles: authorities,
        accessToken: token
      });
    });
};
```

controllers*/user.controller.js*

```
exports.allAccess = (req, res) => {
    res.status(200).send("Nyilvános tartalom.");
  };
  exports.userBoard = (req, res) => {
    res.status(200).send("Felhasználói tartalom.");
  };
  exports.adminBoard = (req, res) => {
    res.status(200).send("Admin tartalom.");
  };
  exports.moderatorBoard = (req, res) => {
    res.status(200).send("Moderátor tartalom.");
  };
```

Útvonalak meghatározása

A köztesszoftvert összekombináljuk a vezérlő funkciókkal. Útjainkat két részre bontjuk: autentikáció és autorizáció.

Autentikáció:

```
POST /api/auth/signup
POST /api/auth/signin
```

routes*/auth.routes.js*

```
const { verifySignUp } = require("../middlewares");
const controller = require("../controllers/auth.controller");
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });
  app.post(
    "/api/auth/signup",
    [
      verifySignUp.checkDuplicateUsernameOrEmail,
      verifySignUp.checkRolesExisted
    ],
    controller.signup
  );
  app.post("/api/auth/signin", controller.signin);
};
```

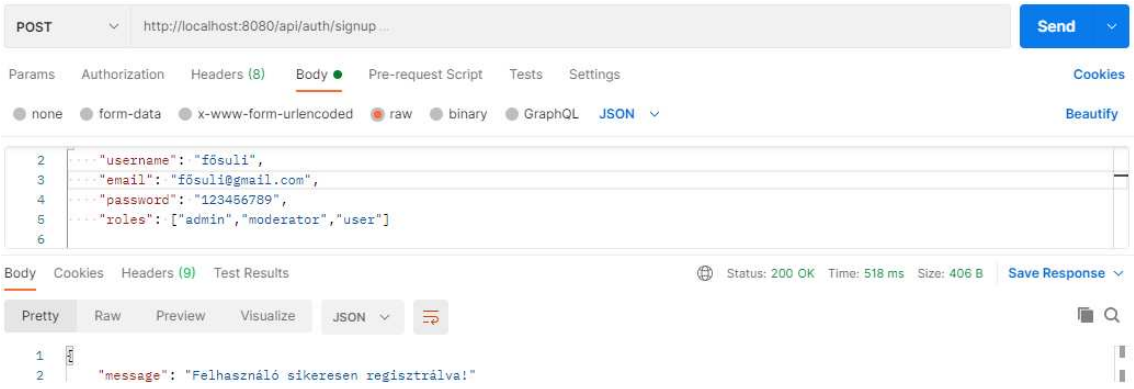
Autorizáció:

```
GET /api/test/all
GET /api/test/user
GET /api/test/mod
GET /api/test/admin
```

routes/*user.routes.js*

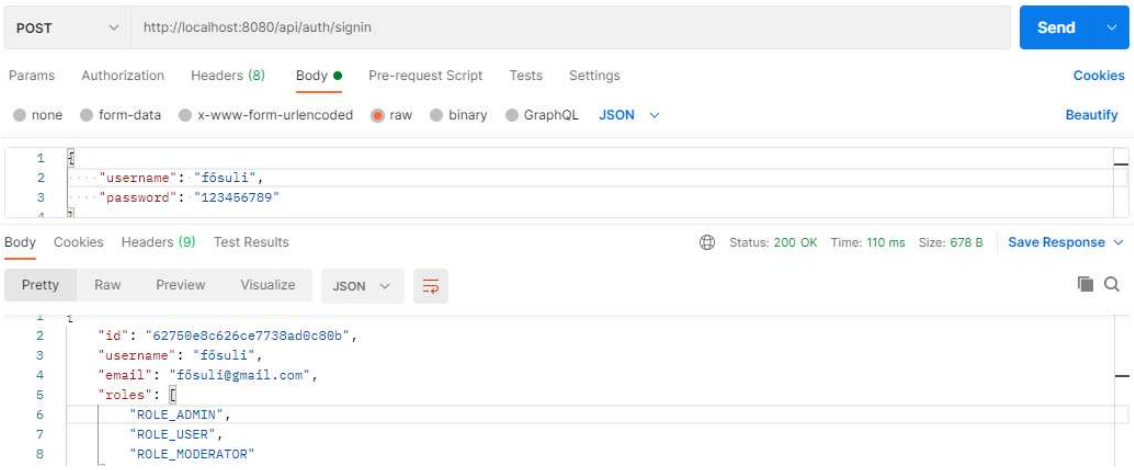
```
const { authJwt } = require("../middlewares");
const controller = require("../controllers/user.controller");
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });
  app.get("/api/test/all", controller.allAccess);
  app.get("/api/test/user", [authJwt.verifyToken], controller.userBoard);
  app.get(
    "/api/test/mod",
    [authJwt.verifyToken, authJwt.isModerator],
    controller.moderatorBoard
  );
  app.get(
    "/api/test/admin",
    [authJwt.verifyToken, authJwt.isAdmin],
    controller.adminBoard
  );
};
```

A *node server.js* paranccsal futattjuk az alkalmazást. Postman API-n keresztül leteszteljük, regisztrálunk egy felhasználót.



3.2. ábra Képernyőmentés a Postmanról

Bejelentkezünk az új felhasználó adataival.



3.3. ábra Képernyőmentés a Postmanról

3.3. JWT Frontend

[17]A meglévő React alkalmazásunkhoz a **services** könyvtárhoz hozzáadunk 3 fájlt:

```
auth-header.js
auth.service.js
user.service.js
```

A hitelesítési szolgáltatás (*auth.service.js*) a következő fontos metódusokat fogja nekünk biztosítani:

login(): POST felhasználónév, jelszó elmenti a JWT-t a helyi tárhelyre.

logout(): eltávolítja a JWT-t a helyi tárhelyről.

register(): POST felhasználónév, e-mail, jelszó.

getCurrentUser(): az eltárolt felhasználói információk lekérése (beleértve a JWT-t is).

```
import axios from "axios";

const API_URL = "http://localhost:8080/api/auth/";

class AuthService {
  login(username, password) {
    return axios
      .post(API_URL + "signin", {
        username,
        password
      })
      .then(response => {
        if (response.data.accessToken) {
          localStorage.setItem("user", JSON.stringify(response.data));
        }

        return response.data;
      });
  }

  logout() {
    localStorage.removeItem("user");
  }
  register(username, email, password) {
    return axios.post(API_URL + "signup", {
      username,
      email,
      password
    });
```

[17]https://www.bezkoder.com/react-jwt-auth/

```
  }
  getCurrentUser() {
    return JSON.parse(localStorage.getItem('user'));;
  }
}

export default new AuthService();
```

Vannak módszereink is az adatok szerverről való lekérésére. Abban az esetben, ha védett tartalomhoz férünk hozzá, a HTTP-kérésnek engedélyezési fejléc szükséges. Ha van egy bejelentkezett felhasználó hozzáférési tokennel (JWT), adja vissza a HTTP engedélyezési fejlécet. Ellenkező esetben adjon vissza egy üres objektumot.

auth-header.js

```
export default function authHeader() {
    const user = JSON.parse(localStorage.getItem('user'));

    if (user && user.accessToken) {

      return { 'x-access-token': user.accessToken };
    } else {
      return {};
    }
  }
```

user.service.js

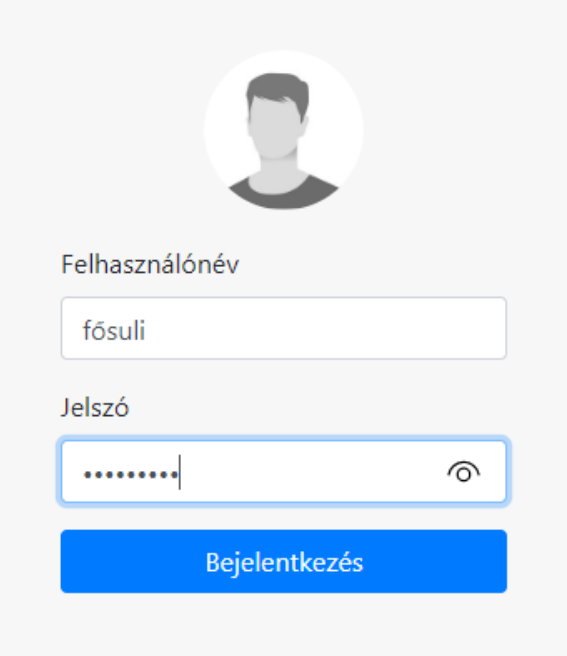
```
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8080/api/test/';
class UserService {
  getPublicContent() {
    return axios.get(API_URL + 'all');
  }
  getUserBoard() {
    return axios.get(API_URL + 'user', { headers: authHeader() });
  }
  getModeratorBoard() {
    return axios.get(API_URL + 'mod', { headers: authHeader() });
  }
  getAdminBoard() {
    return axios.get(API_URL + 'admin', { headers: authHeader() });
  }
}
export default new UserService();
```

Komponensek az autentikációhoz

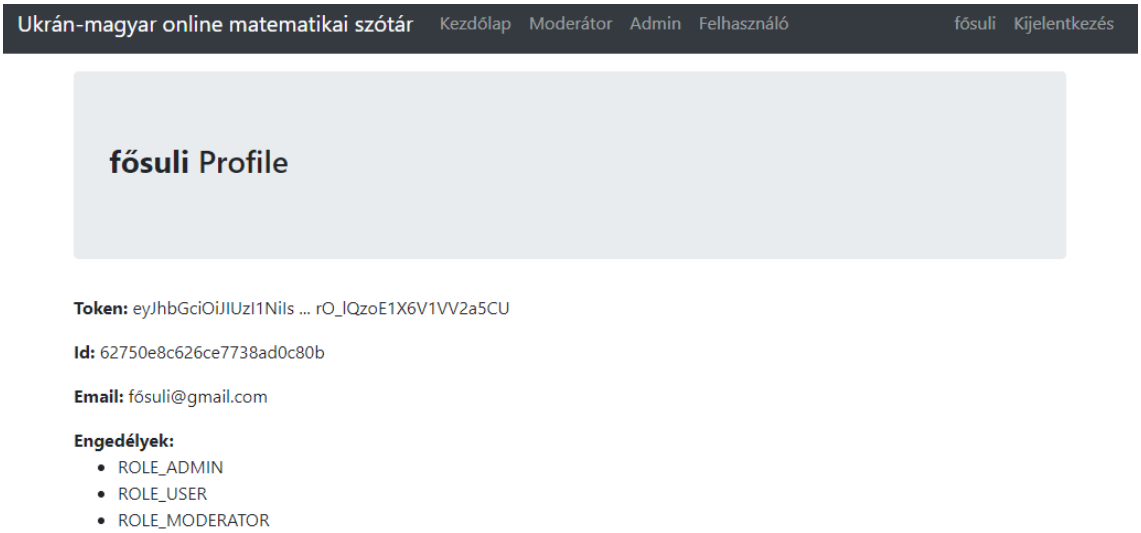
Bejelentkezésnél lesz egy űrlap felhasználónévvel és jelszóval:

A mezők kitöltése kötelező, szóval ellenőriznünk is kell azokat.



3.4. ábra Képernyőmentés a bejelentkezésről

Ha rendben van az ellenőrzés behívjuk az *AuthService.login()* metódust, utána átirányitjuk a felhasználót a Profil oldaljához.



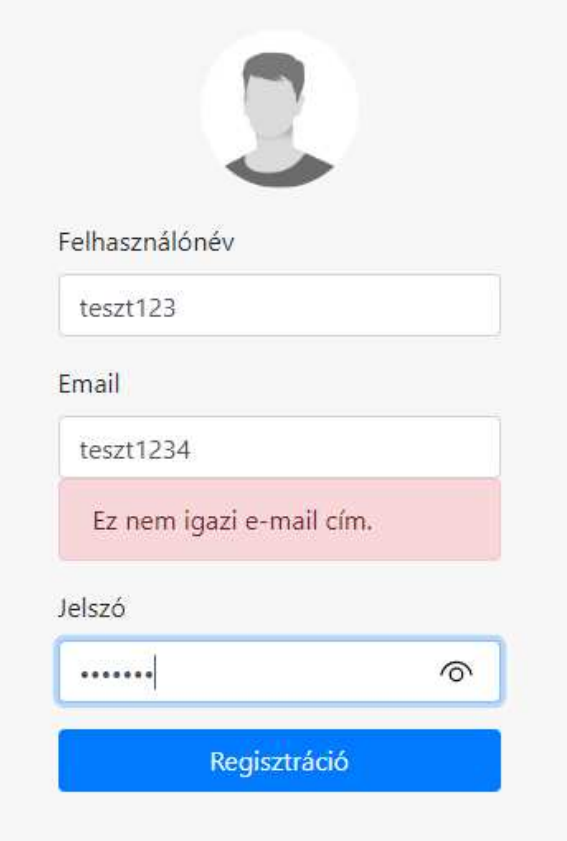
3.5. ábra Képernyőmentés a profilról

A regisztráció hasonló a bejelentkezéshez, itt megadunk néhány formai követelményt:

Felhasználónév: 3 és 20 karakter között legyen.

Email: kötelező az email formátum.

Jelszó: 6 és 40 karakter között legyen.



3.6. ábra Képernyőmentés a regisztrációról

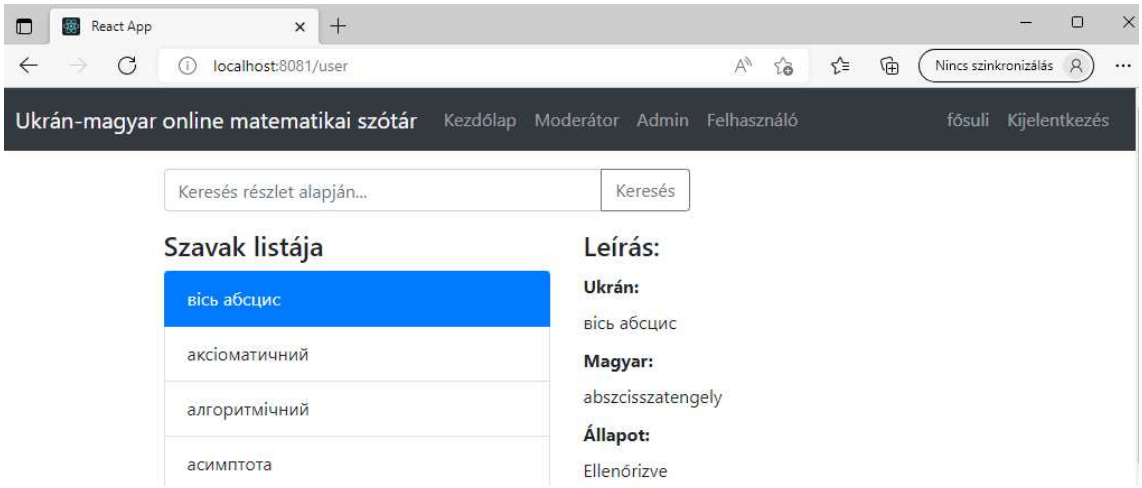
Útvonalak és navigációs sáv

Az *App.js*-hez hozzáadjuk a navigációs sávot és beállítjuk az útvonalakat. Ez az egész alkalmazásunk gyökere, a navigációs sávot úgy kell beállítani, hogy bejelentkezési állapot és az aktuális szerepkör szerint változzon.

Kezdőlap: minden esetben elérhető, nem szükséges a bejelentkezés.

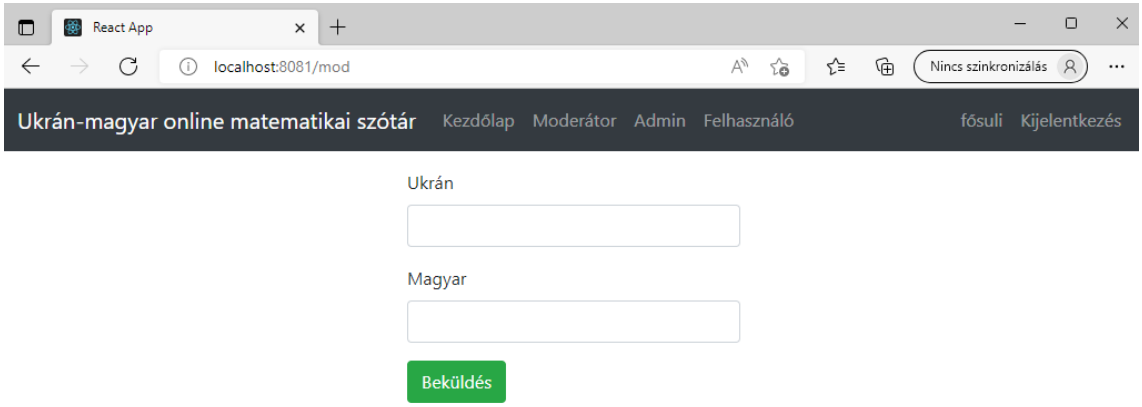
Regisztráció és bejelentkezés: addig elérhető míg a felhasználó nem jelentkezett be, utána kijelentkezésre változik.

Felhasználó: vissza adja az értéket az aktuális felhasználóról. Látható a szótárban lévő szavak listája.



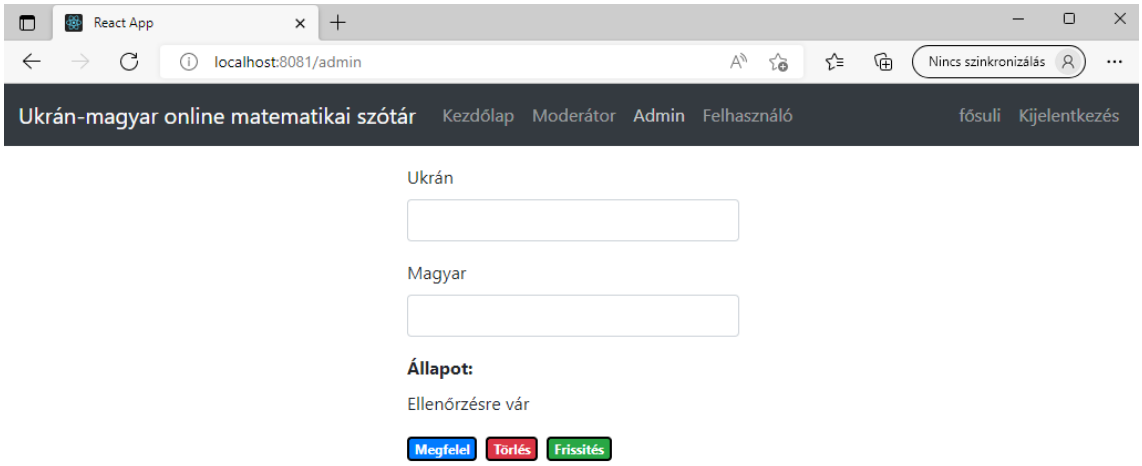
3.7. ábra Képernyőmentés a felhasználói lapról

Moderátor: elérhető ha a felhasználó jogosult moderátori engedélyekkel, mi esetünkben a szótárhoz képes hozzáadni új szavakat.



3.8. ábra Képernyőmentés a moderátori lapról

Admin: elérhető ha a felhasználó jogosult adminisztrátori engedélyekkel, képes a CRUD összes műveletére (hozzáadni, frissíteni, törölni, lekérni).



3.9. ábra Képernyőmentés az adminisztrátori lapról

```
class App extends Component {
  constructor(props) {
    super(props);
    this.logOut = this.logOut.bind(this);
    this.state = {
      showModeratorBoard: false,
      showAdminBoard: false,
```

```
    currentUser: undefined,
  };
}
componentDidMount() {
  const user = AuthService.getCurrentUser();
  if (user) {
    this.setState({
      currentUser: user,
      showModeratorBoard: user.roles.includes("ROLE_MODERATOR"),
      showAdminBoard: user.roles.includes("ROLE_ADMIN"),
    });
  }
}
logOut() {
  AuthService.logout();
}
render() {
  const { currentUser, showModeratorBoard, showAdminBoard } = this.state;
  return (
    <div>
      <nav className="navbar navbar-expand navbar-dark bg-dark">
        <Link to={"/home"} className="navbar-brand">
          Ukrán-magyar online matematikai szótár
        </Link>
        <div className="navbar-nav mr-auto">
          <li className="nav-item">
            <Link to={"/home"} className="nav-link">
              Kezdőlap
            </Link>
          </li>
          {showModeratorBoard && (
            <li className="nav-item">
              <Link to={"/mod"} className="nav-link">
                Moderátor
              </Link>
            </li>
          )}
          {showAdminBoard && (
            <li className="nav-item">
              <Link to={"/admin"} className="nav-link">
                Admin
              </Link>
            </li>
          )}
          {currentUser && (
            <li className="nav-item">
              <Link to={"/user"} className="nav-link">
                Felhasználó
              </Link>
            </li>
          )}
```

```
          </div>
          {currentUser ? (
            <div className="navbar-nav ml-auto">
              <li className="nav-item">
                <Link to={"/profile"} className="nav-link">
                  {currentUser.username}
                </Link>
              </li>
              <li className="nav-item">
                <a href="/login" className="nav-link" onClick={this.logOut}>
                  Kijelentkezés
                </a>
              </li>
            </div>
          ) : (
            <div className="navbar-nav ml-auto">
              <li className="nav-item">
                <Link to={"/login"} className="nav-link">
                  Bejelentkezés
                </Link>
              </li>
              <li className="nav-item">
                <Link to={"/register"} className="nav-link">
                  Regisztráció
                </Link>
              </li>
            </div>
          )}
        </nav>
        <div className="container mt-3">
          <Switch>
            <Route exact path={["/", "/home"]} component={Home} />
            <Route exact path="/login" component={Login} />
            <Route exact path="/register" component={Register} />
            <Route exact path="/profile" component={Profile} />
            <Route path="/user" component={SzotarList} />
            <Route path="/mod" component={AddSzavak} />
            <Route path="/admin" component={Szavak} />
          </Switch>
        </div>
      </div>
    );
  }
}
export default App;
```

Az alkalmazást az *npm start* paranccsal lehet elindítani.

Összegzés

Szakdolgozatom célja egy stabil alkalmazás kialakítása online ukrán-magyar matematika szótár részére, valamint ennek a rendszernek a tovább fejlesztése MERN stack használatával. Az alkalmazás idén autentikációval és autorizációval bővült, ez a JSON web token (JWT) ezen belül egy új Node.js Express MongoDB háttérszerver alkalmazásával lett megvalósítva. Az autentikáció tartalmazza a regisztrációt (regisztrációnál meg kell felelni a formai követelményeknek), valamint bejelentkezést.

Az autorizáció a felhasználói jogosultságokat tartalmazza. Jelenleg három féle szerepkör létezik az egész applikációban: felhasználói, moderátori, adminisztrátori. Amennyiben nem adjuk meg a szerepkört vagy böngészőben történik regisztráció automatikusan felhasználói jogosultságokat kap az új felhasználó. Védett tartalmak / erőforrások csak akkor elérhetőek az alkalmazásban ha a felhasználó rendelkezik a hozzá szükséges jogosultságokkal.

Irodalomjegyzék

[1] https://www.mongodb.com/mern-stack *(Hozzáférés időpontja: 2021.05.05)*

[2] https://www.mongodb.com *(Hozzáférés időpontja: 2021.05.05)*

[3] https://www.mongodb.com/json-and-bson *(Hozzáférés időpontja: 2022.05.02)*

[4] https://www.oreilly.com/library/view/web-development-with/9781491902288/ch01.html *(Hozzáférés időpontja: 2022.05.02)*

[5] https://expressjs.com/ *(Hozzáférés időpontja: 2021.05.03)*

[6] https://expressjs.com/en/guide/using-template-engines.html *(Hozzáférés időpontja: 2022.05.05)*

[7] https://hu.reactjs.org/ *(Hozzáférés időpontja: 2022.05.05)*

[8] https://github.com/facebook/react/ *(Hozzáférés időpontja: 2021.04.25)*

[9] https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle *(Hozzáférés időpontja: 2021.05.07)*

[10] https://nodejs.dev/learn *(Hozzáférés időpontja: 2021.05.04)*

[11] https://www.javascripttutorial.net/nodejs-tutorial/what-is-npm/ *(Hozzáférés időpontja: 2021.05.04)*

[12] https://bezkoder.com/node-express-mongodb-crud-rest-api/ *(Hozzáférés időpontja: 2021.05.06)*

[13] https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis *(Hozzáférés időpontja: 2021.05.06)*

[14] https://bezkoder.com/react-crud-web-api/ *(Hozzáférés időpontja: 2021.05.06)*

[15] https://www.akana.com/blog/what-is-jwt *(Hozzáférés időpontja: 2022.05.02)*

[16] https://www.bezkoder.com/node-js-mongodb-auth-jwt/
(Hozzáférés időpontja: 2022.05.02)

[17] https://www.bezkoder.com/react-jwt-auth/
(Hozzáférés időpontja: 2022.05.03)

[18] https://stackoverflow.com *(Hozzáférés időpontja: 2021.04.27)*

[19] https://www.pluralsight.com *(Hozzáférés időpontja: 2021.04.29)*

Ábrák jegyzéke

Mellékletek

A következő linken érhető el a szótár fájlai:

https://github.com/IstvanKovach/Ukranmagyarszotarv2 *(Hozzáférés időpontja: 2022.05.07)*

A szakdolgozatban szereplő képek forrásai:

https://www.bocasay.com/how-does-the-mern-stack-work/ *(Hozzáférés időpontja: 2022.05.07)*

https://www.edureka.co/blog/interview-questions/react-interview-questions/ *(Hozzáférés időpontja: 2022.05.07)*

https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle *(Hozzáférés időpontja: 2022.05.07)*

https://sureshdsk.dev/how-json-web-token-jwt-authentication-works *(Hozzáférés időpontja: 2022.05.07)*

Резюме

Метою моєї дипломна робота є розробка стабільної програми для онлайн українсько-угорського математичного словника та подальший розвиток цієї системи за допомогою стека MERN. Цього року додаток було розширено за допомогою аутентифікації та авторизації, цей веб-токен JSON (JWT) було реалізовано за допомогою нового серверного сервера Node.js Express MongoDB. Аутентифікація включає реєстрацію (формальні вимоги повинні бути виконані для реєстрації) та вхід.

Авторизація включає привілеї користувача. Наразі в додатку є три типи ролей: користувач, модератор, адміністратор. Якщо не вказати роль або реєстрацію браузера, нові права користувача автоматично надаються. Захищений вміст/ресурси доступні в програмі лише за умови, що користувач має необхідні дозволи.

UNICHECK

Ім'я користувача:
Моца Андрій Андрійович

ID перевірки:
1011109916

Дата перевірки:
09.05.2022 14:12:46 EEST

Тип перевірки:
Doc vs Internet

Дата звіту:
09.05.2022 14:23:34 EEST

ID користувача:
100006701

Назва документа: **Kovács_István**

Кількість сторінок: **51** Кількість слів: **8304** Кількість символів: **53771** Розмір файлу: **1.15 MB** ID файлу: **1011008908**

21.9%
Схожість

Найбільша схожість: **12%** з Інтернет-джерелом (https://vntalking.com/node-authentication-va-authorization-su-dung-jw..

21.9% Джерела з Інтернету | 111 ..... Сторінка 53

Пошук збігів з Бібліотекою не проводився

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Nyilatkozat

Alulírott, Kovács István, 014. Középiskolai oktatás (Matematika) képzési program hallgatója, kijelentem, hogy a dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskolán, a Matematika és Informatika Tanszéken készítettem, 014. Középiskolai oktatás (Matematika) BSc diploma megszerzése végett.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola könyvtárában a kölcsönözhető könyvek között helyezik el.