Web technológiák 2.

Féléves feladat

Szótárfüzet alkalmazás

Készítette: **Palencsár Enikő**

Neptun kód: **YD11NL**

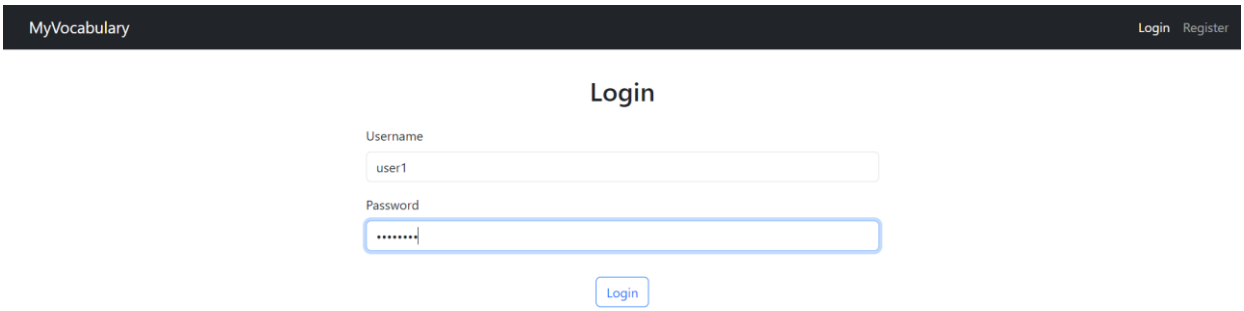
Tartalomjegyzék

1. Felhasználói felület

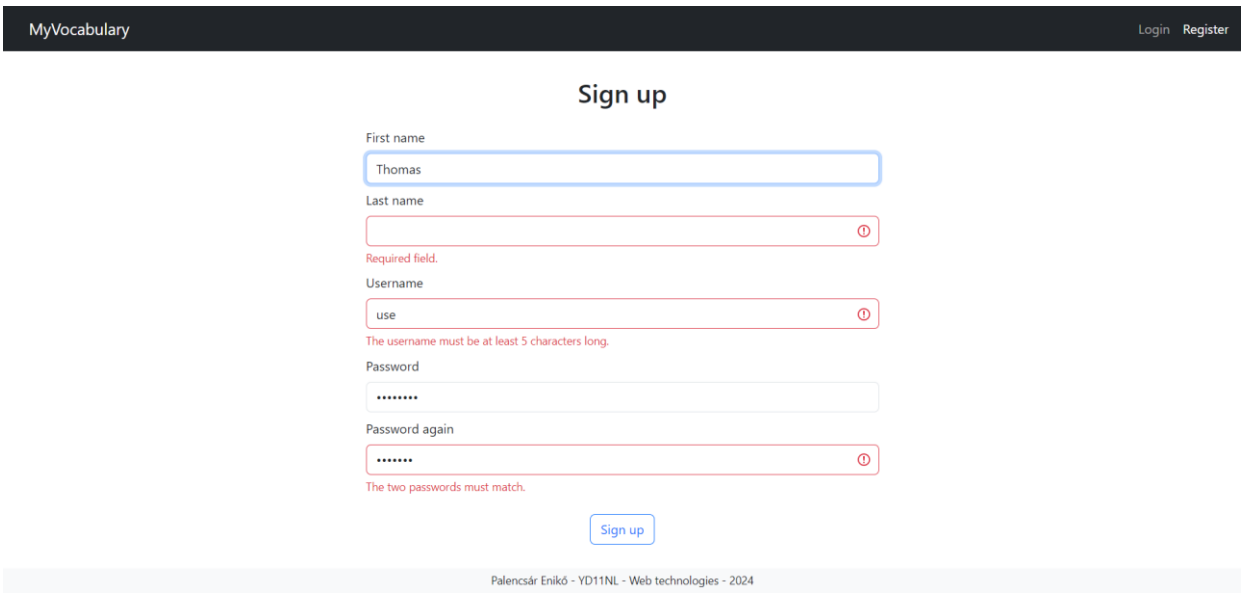
A MyVocabulary egy online szótárfüzet alkalmazás, melyben a regisztrált felhasználók szószedeteket, topic-okat hozhatnak létre, ezekhez szavakat adhatnak hozzá, melyeket aztán szódolgozat-szerűen gyakorolni is tudnak. Minden felhasználó minden topic-hoz rendelkezik teljeskörű (create, update, delete) hozzáféréssel, de a gyakorlások közül csak a sajátjait látja, melyhez topic-onként grafikonokkal illusztrált statisztikát is megjeleníthet.

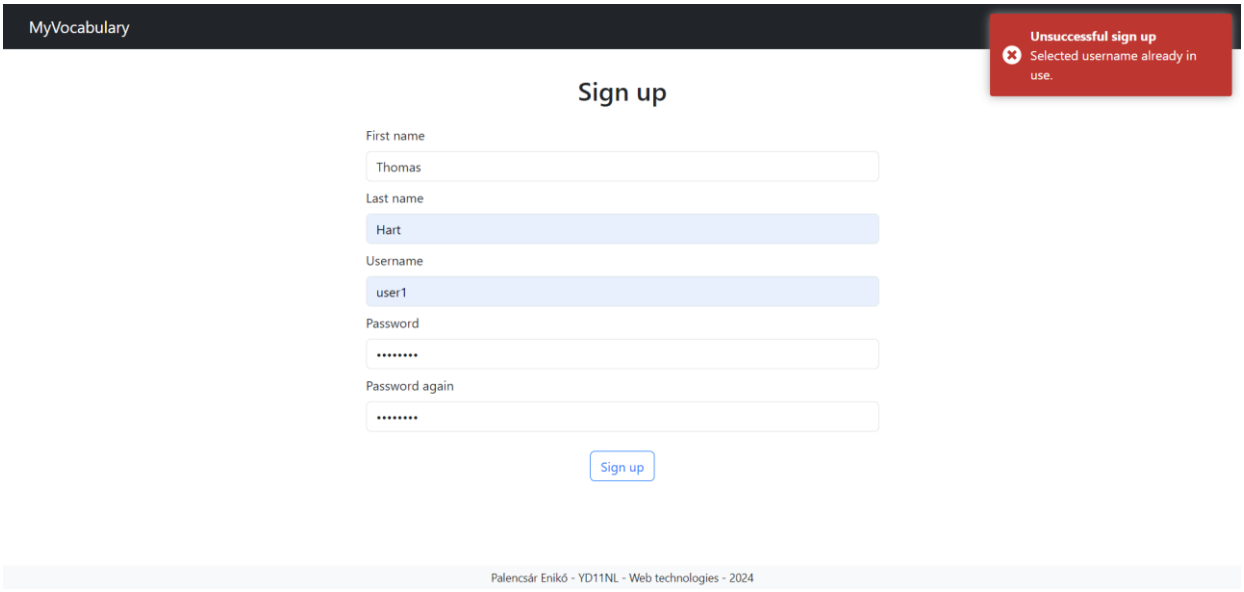
1.1. Bejelentkezés és regisztráció

A szószedetekhez csak bejelentkezett felhasználók férhetnek hozzá, bejelentkezéskor a felhasználónév és a jelszó megadása szükséges. A bejelentkezés legfeljebb két hétre szól, ennek lejártakor a rendszer automatikusan kijelentkeztet.



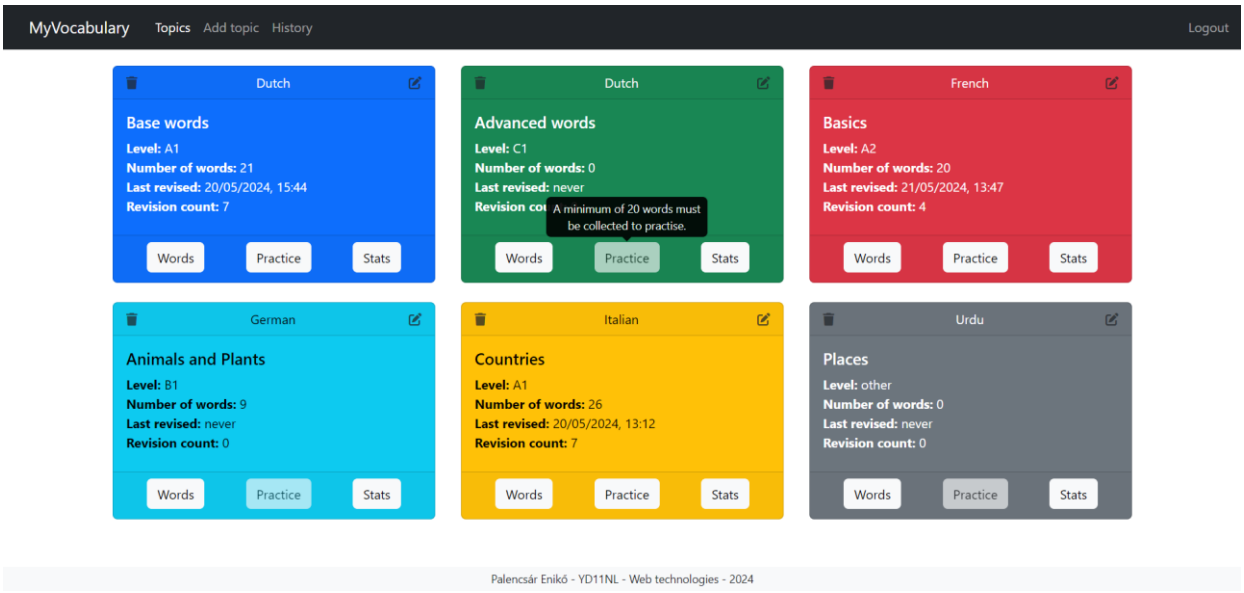
Az oldalra bárki regisztrálhat, ekkor meg kell adni a kereszt- és vezetéknéven túl a felhasználónevet, a jelszót, valamint a jelszó megerősítését. A regisztrációs form hibaüzenetet jelenít meg, ha a megadott adatok érvénytelenek, továbbá felugró hibaüzenet jelenik meg akkor is, ha a kiválasztott felhasználónév már szerepel az adatbázisban. Sikeres regisztráció esetén az oldal automatikusan átirányít a bejelentkezés menüpontra.



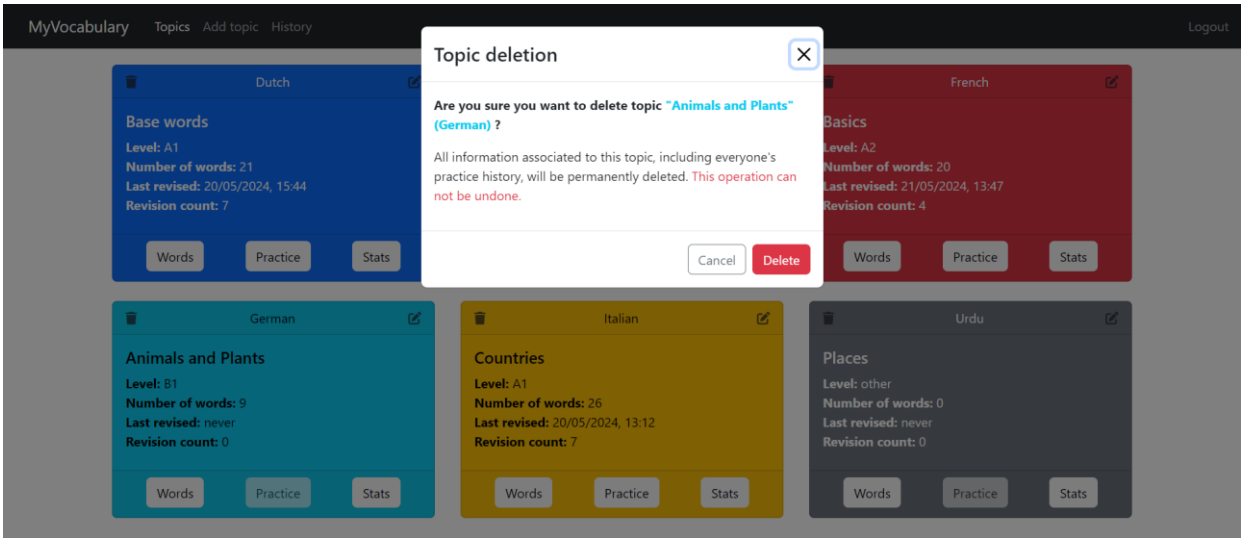


1.2. Topic lista

Az alkalmazás főoldalán listázásra kerül valamennyi adatbázisban szereplő topic. A megjelenített adatok: a topic neve, nyelve, szintje, a tartalmazott szavak száma, az utolsó gyakorlás dátuma, az összes gyakorlás száma. Minden topic-hoz tartozik egy saját színséma a Bootstrap alapszínei közül, mely szintén eltárolásra kerül az adatbázisban, ilyen színben jelenik meg az adott topic-ot reprezentáló Bootstrap kártya. Az egyes kártyákból kiindulva lehetőség nyílik a tartalmazott szavak listájának megjelenítésére és bővítésére, gyakorlófeladat indítására, továbbá a gyakorlási statisztikák megtekintésére. Mivel a gyakorlás minden esetben 20 szóból áll, azoknál a szószedeteknél, melyek ennél kevesebb kifejezést tartalmaznak, a *Practice* gomb le van tiltva, ennek magyarázata az egeret a gomb felé húzva egy tooltip formájában is megjelenik.

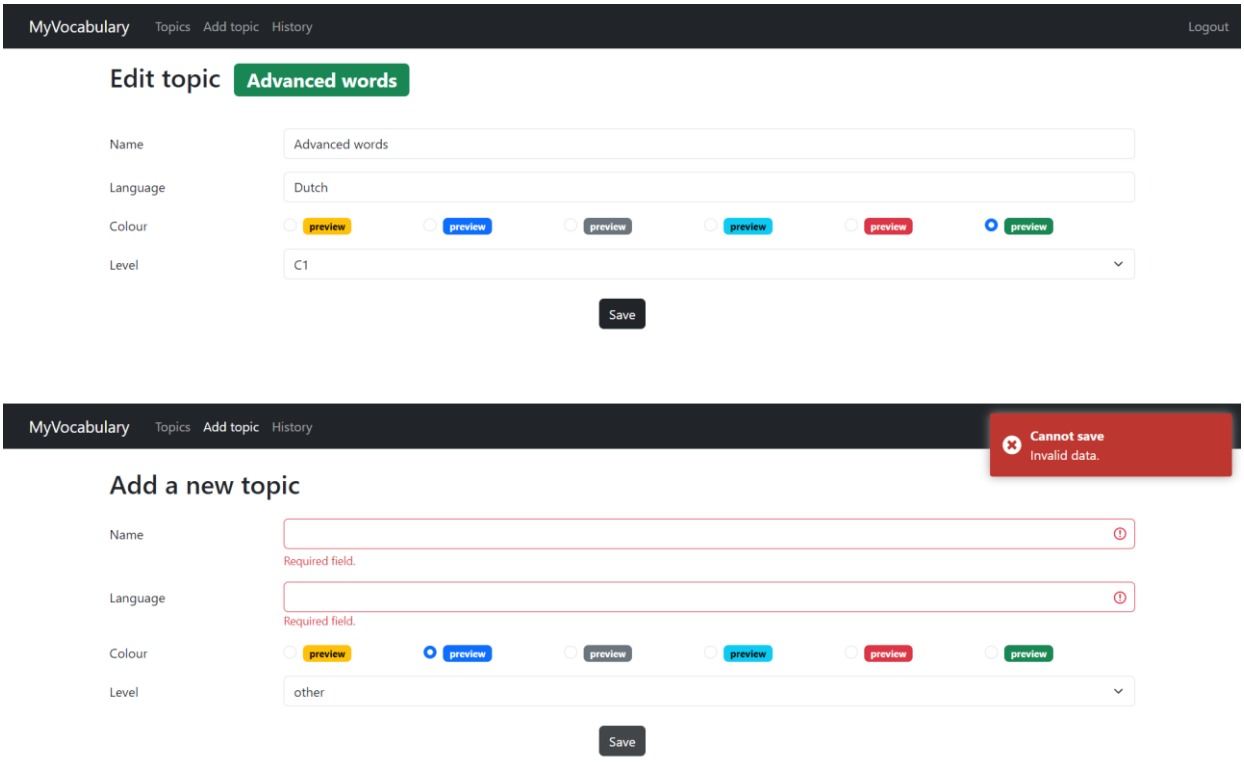


A kártyák tetején lévő szemetesvödör ikonra kattintva a topic egy párbeszédablak jóváhagyását követően törölhető, míg a toll ikonra kattintva a topic alapadatainak szerkesztésére nyílik lehetőség.



1.3. Topic hozzáadása és módosítása

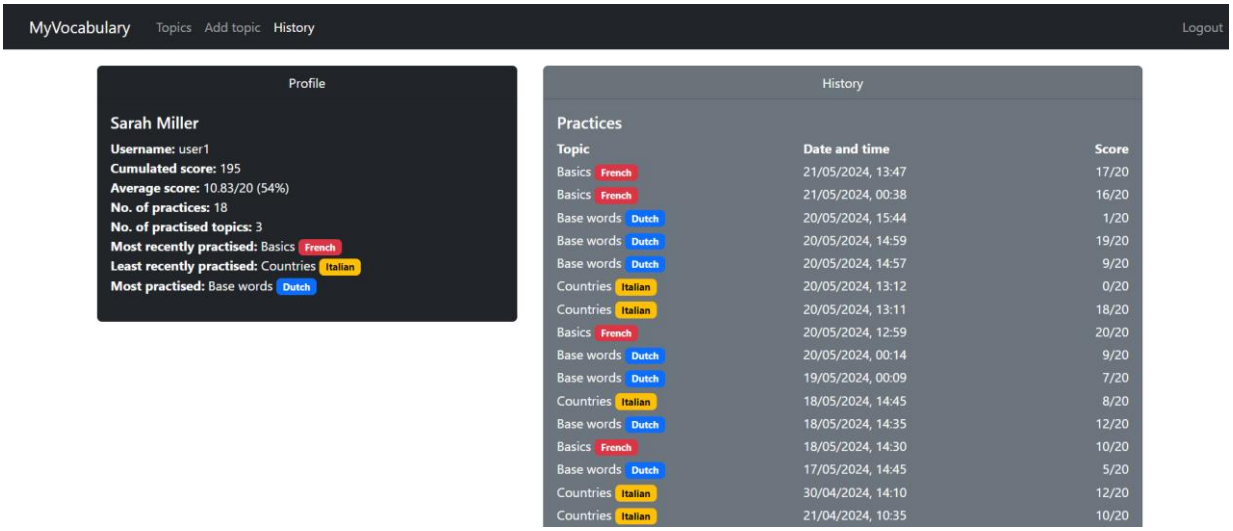
A szószedetek alapadatainak szerkesztését validált form biztosítja. Itt szöveges beviteli mezőben megadható a topic neve és nyelve, rádiógombbal kiválasztható a szószedethez társított színséma, illetve egy legördülő listában megjelölhetjük a szószedet szintjét is. Hibás kitöltés esetén a megfelelő beviteli mező alatt hibaüzenet jelenik meg, és a mentés folyamata nem indul el. A meglévő szószedet szerkesztésén túl új topic hozzáadására is lehetőség nyílik az *Add topic* menüpontot kiválasztva.



1.4. Profil és gyakorlások

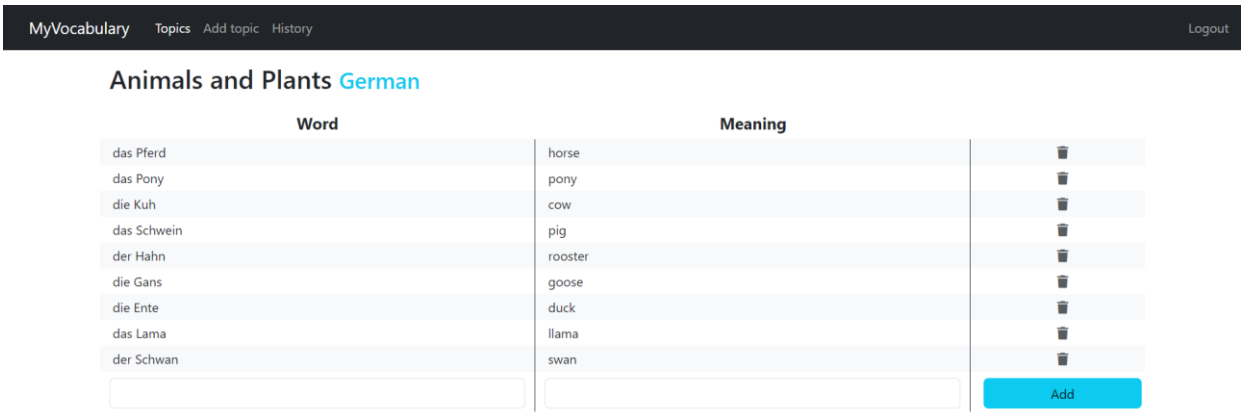
A *History* menüpontban a bejelentkezett felhasználó megtekintheti a saját alapadatait (teljes név, felhasználónév), valamint az eddigi összes gyakorlása részleteinek összesítő táblázatát, időpont szerint csökkenő sorrendben. A profil tartalmaz továbbá egy összesítést a felhasználó

eredményeiről, melynek részét képezi a kumulált pontszám, az átlagpontszám, a gyakorlások száma, az érintett szószedetek száma, a legutóbb, a legrégebben, és a legtöbbet gyakorolt szószedet neve.



1.5. Szavak

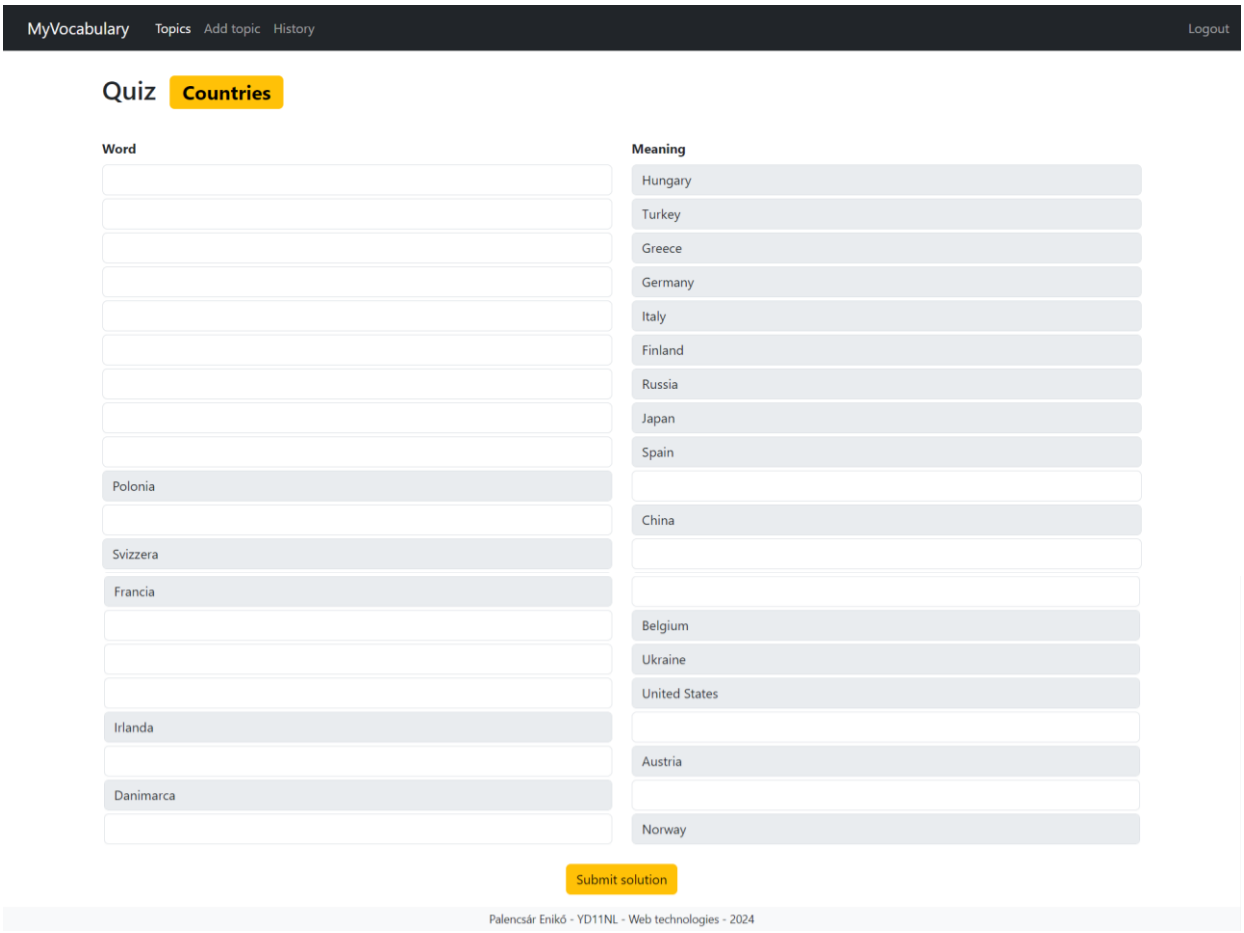
A *Words* oldalon az adott szószedethez tartozó szavak listája érhető el. A párok melletti szemetesvödör ikonra kattintva az adott pár törölhető a szószedetből. A szótárfüzetszerű elrendezés utolsó sorában lehetőség nyílik új szó-jelentés pár hozzáadására (egyik mező sem lehet üres, ez ellenőrzésre kerül a beküldés előtt).

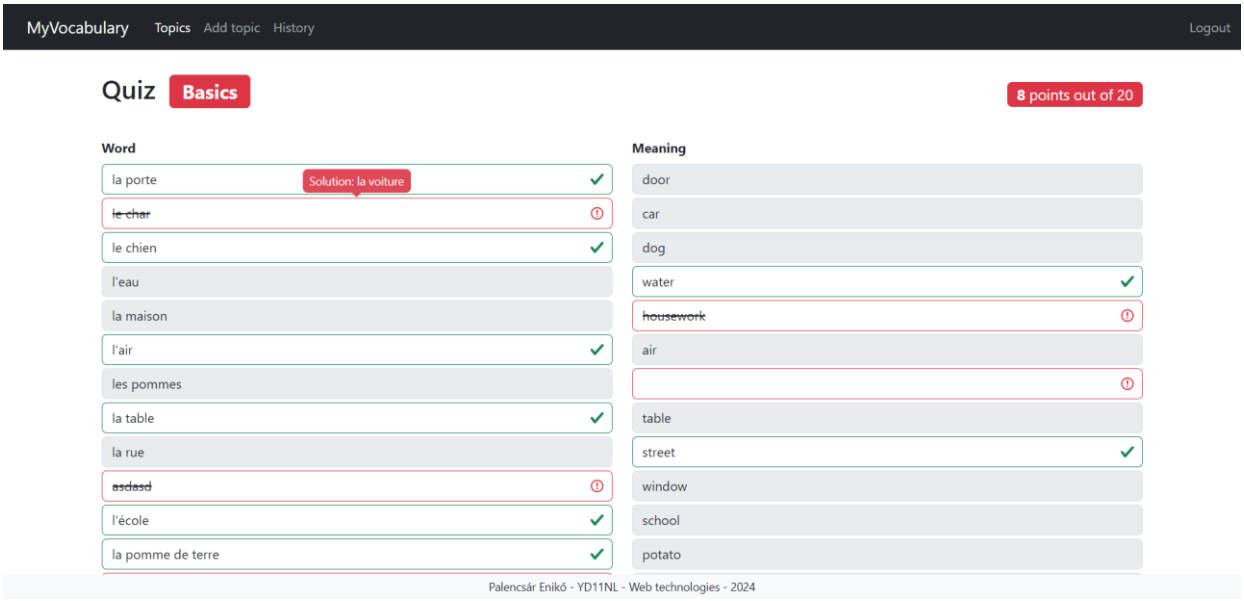


1.6. Gyakorlás

A *Practice* oldalon egy adott témakör szavainak gyakorlására nyílik lehetőség. A gyakorlás kezdetén 20 szó kerül véletlenszerűen kiválasztásra a szószedetből, melyeknek vagy az idegen nyelvű (75% valószínűséggel), vagy a forrásnyelvi (25% valószínűséggel) verziója kerül kikérdezésre, a másik verzió pedig egy inaktív input mezőbe kerül. A megoldás a *Submit solution* gombra kattintva adható be, ekkor a felhasználói válaszok az esetleges leading és trailing üres karakterek leválasztását követően automatikusan kiértékelésre kerülnek. A helyes válaszok mellett zöld pipa, a helytelen megoldások mellett pedig piros felkiáltójel jelenik meg, az addig írható beviteli mezők csak olvashatóvá válnak. Ezenkívül minden helytelen megoldás áthúzásra kerül, és az egeret fölé mozgatva tooltipben megjelenik az elvárt helyes válasz. Az összpontszám kijelzésére felül, a topic neve mellett kerül sor. A javítás áttekintését követően lehetőség nyílik erről az oldalról

átnavigálni az adott szószedethez tartozó statisztikákat megjelenítő lapra.

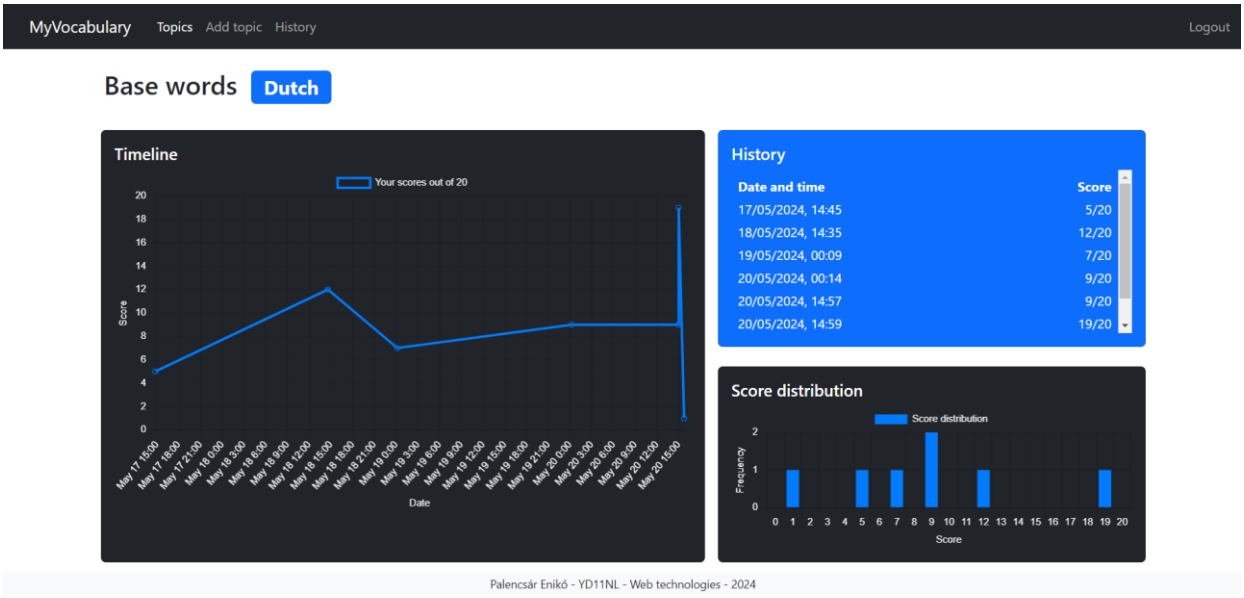




1.7. Statisztika

A *Stats* oldalon egy adott topic-ra vonatkozóan tekinthetőek meg a bejelentkezett felhasználó gyakorlásainak eredményei. Jobb oldalon egy listaszerű felsorolása látható a gyakorlásoknak időrendben, az elért pontszámok kijelzésével. Alatta az elért eredmények eloszlásának oszlopdiagramja jelenik meg, a vízszintes tengelyen az elérhető pontszámokkal, a függőlegesen az

adott pontszámot eredményező gyakorlások számával. Baloldalt a gyakorlások idővonala látható, ez a grafikon tetszőlegesen zoomolható, így az éves, a havi, a heti, és a napi eredmények alakulása is nyomon követhető a segítségével.



1.8. Elérhetőség

A MyVocabulary alkalmazáshoz tartozó forrásfájlokat, az adatbázisba importálható JSON fájlokat, és a futtatás lépéseit tartalmazó Github repository linkje: **https://github.com/enikop/Web_technologies_2**

2. Kód

A MyVocabulary alkalmazás frontendje az Angular keretrendszer alkalmazásával került kialakításra, komponensekből és service-ekből épül fel, melyek nyelve TypeScript. A weboldalak egységes és reszponzív megjelenését a Bootstrap keretrendszer biztosítja. A backend (server jegyzék) szintén TypeScriptben íródott, TypeORM leképzést használva MongoDB adatbázis driverrel. Fontos megjegyezni, hogy a TypeORM alapvetően relációs adatbázisokhoz, nem pedig dokumentum adatbázisokhoz lett kitalálva, ezért bizonyos funkciói nem, vagy csak korlátozottan elérhetők MongoDB-t használva. Ennek ismeretében a továbbiakban mindenképpen érdemes fontolóra venni a Mongoose használatát a TypeORM helyett. A szerver Node.js környezetben fut, az Express keretrendszert használva.

2.1. Entity

A TypeORM rendszerében az adatbázisszerkezetet úgynevezett entitások segítségével írhatjuk le, melyek @Colum() annotációval ellátott adattagjai tábla mezőknek, MongoDB esetén JSON tulajdonságoknak felelnek meg. A feladat megoldása során három alapvető entitással dolgoztam, melyek egy Word nevű Topic entitásba beágyazott elemmel egészültek ki:

- User: regisztrált felhasználó, tulajdonságai: kereszt- és vezetéknév, felhasználónév, jelszó
- Topic: szószedet, tulajdonságai: név, nyelv, színséma (társított Bootstrap szín neve), szint, szavak listája (a topic kollekció elemeibe közvetlenül beágyazva)
    - Word: szó, tulajdonságai: forrásnyelvi szó, célnyelvi szó
- Practice: gyakorlás, tulajdonságai: időpont, elért pontszám, a gyakorló felhasználó felhasználóneve, a gyakorolt szószedet azonosítója

A felhasználónevek egyediségét az `@Index({ unique: true })` annotáció garantálja. Minden entitás rendelkezik továbbá _id néven egy ObjectId-val, ami a MongoDB kollekciók tagjai esetén alapkövetelmény.

```
@Entity()
export class User {
    @ObjectIdColumn()
    _id: ObjectId

    @Column()
    firstName: string

    @Column()
    lastName: string

    @Column()
    @Index({ unique: true })
    username: string

    @Column()
    password: string;
}
```

```
@Entity()
export class Topic {
    @ObjectIdColumn()
    _id: ObjectId

    @Column()
    name: string

    @Column()
    language: string

    @Column({
      type: 'enum',
      enum: Colour,
      default: Colour.Blue
    })
    colour: string = Colour.Blue

    @Column({
      type: 'enum',
      enum: Level,
      default: Level.Other
    })
    level: string = Level.Other

    @Column(type => Word, {array: true})
    words: Word[] = []
}
```

```
export class Word {
    @Column()
    source: string

    @Column()
    target: string

    constructor(source: string, target: string) {
      this.source = source;
      this.target = target;
    }
}
@Entity()
export class Practice {
    @ObjectIdColumn()
    _id: ObjectId

    @CreateDateColumn()
    timestamp: string

    @Column()
    score: number

    @Column()
    username: string

    @Column()
    topicId: string
}
```

Megfigyelhető, hogy az alkalmazott adatszerkezet egyedkapcsolat szinten nem konzisztens, használ a dokumentum adatbázisokra jellemző beágyazást (Word-Topic), és relációs jellegű idegenkulccsal jelölt kapcsolatokat egyaránt (Practice-User, Practice-Topic). Ez utóbbinak fő oka a tárolt adatok jellege, a lekérdezések egyszerűségének fenntartása, továbbá a redundancia és a sokszoros beágyazások elkerülésének igénye. Az entitásoknak megfelelő egyedszerkezettel 3 különböző kollekció kerül az adatbázisba, *melyekhez a fent linkelt Github repository-ban találhatók importálható adatok (nem kötelező, a program fut nélkülük is, csak regisztrálni kell, és mindent az alapoktól felépíteni programon belül).*

2.2. Controller

Az adatelérést a backenden controller osztályok biztosítják, melyek a create-read-update-delete mintát valósítják meg, egységes metódusaik: *getAll, getOne, create, update, delete*. Említendő még a hibakezelő metódus, ami duplikált adat hiba esetén 422 (Unprocessable Entity) HTTP státusszal küld vissza hibaüzenetet a kérelmezőnek, így a frontenden lehetőség nyílik *„A megadott felhasználónév már használatban van”* hibaüzenet megjelenítésére. A *UserController* a fent említetteken kívül definiál egy *login* metódust is, ahol sikeres azonosítás esetén a válasz JWT tokent tartalmaz a bejelentkezett User felhasználói nevével. Ezt a frontenden el kell tárolni, majd minden szerver felé küldött kérés Authorization fejlécében elhelyezni, csak ekkor van lehetőség az adatbázisban tárolt adatok elérésére. Ez alapján létrehoztam egy *getAuthenticated* metódust is a *UserController* osztályban, amely a beérkező tokenből kiolvassa a felhasználónevet, és visszaadja

a bejelentkezett User objektumot. A *PracticeController* két ráadás metódust is tartalmaz, melyeket meghívva lehetőség nyílik a gyakorlások felhasználónév alapján történő lekérésére, valamint a gyakorlások felhasználónév és topidId alapján történő lekérésére. Érdemes még kiemelni azt, hogy a *TopicController delete* metódusa nem csupán a szószedetet, hanem a rá vonatkozó valamennyi gyakorlást is törli, valamint, hogy új gyakorlás felvétele előtt mindig ellenőrzésre kerül az, hogy a megadott *username* és *topicId* létező felhasználót és szószedetet jelölnek-e.

A kontrollerek útvonalhoz rendelése a *routes.ts* fájlban felépített express Router alapján az *index.ts* fájlban történik. A kontrollereknek nem minden metódusa felhasználásra, csak azok, melyeket a frontend Service osztályai használnak:

```
export function getRoutes(){
  const router = express.Router();
  const userController = new UserController();
  router.post('/user', userController.create);
  router.post('/user/login', userController.login);
  router.post('/user/authenticated', checkUser, userController.getAuthenticated);

  const topicController = new TopicController();
  router.get('/topic', checkUser, topicController.getAll);
  router.get('/topic/:id', checkUser, topicController.getOne);
  router.post('/topic', checkUser, topicController.create);
  router.put('/topic', checkUser, topicController.update);
  router.delete('/topic/:id', checkUser, topicController.delete);

  const practiceController = new PracticeController();
  router.get('/practice', checkUser, practiceController.getAll);
  router.get('/practice/byuserandtopic/:username/:topicId', checkUserIdentity,
      practiceController.getFiltered);
  router.get('/practice/byuser/:username', checkUserIdentity,
      practiceController.getUserFiltered);
  router.post('/practice', checkUser, practiceController.create);

  return router;
}
```

2.3. Egyéb szerver fájlok

Az *index.ts* fájl a backend belépési pontja, egyetlen async függvényt tartalmaz, ami felépíti az express szervert. Itt megtörténik a kapcsolódás a *data-source.ts* fájlban specifikált adatbázishoz, és a *routes.ts* fájlban megadott útvonalhoz rendelések végrehajtása. A fenti kódpéldában látható *checkUser* és *checkUserIdentity* middleware függvények a felhasználók azonosítását szolgálják az azonosítási HTTP fejlécben átvett JWT token alapján, ezek definícióját a *protect-routes.ts* fájl tartalmazza. A *checkUser* ellenőrzi, érvényes-e a kapott token, ha nem, kivételt dob, és nem kerül át a vezérlés a megadott kontrollerhez. A *checkIdentity* ezen kívül kiolvassa a tokenből a felhasználónevet is, és ha ez megegyezik a megadott útvonalban szereplő :username paraméterrel, csak akkor engedi folytatódni a feldolgozást. Ezáltal a felhasználók garantáltan csak a saját gyakorlásaik eredményeihez férhetnek hozzá.

```
export const checkUser = expressjwt({
  secret: "8sxYVG3yJG",
  algorithms: ["HS256"]
});

export const checkUserIdentity = (req, res, next) => {
  const token = getToken(req);
  if (!token) {
    throw new UnauthorizedError('credentials_required', { message: 'Unauthorized.'
});
  }
  const decoded = jwt.verify(token, "8sxYVG3yJG") as JwtPayload;
  const bearerUsername = decoded.username;
  const requestedUsername = req.params.username;
  if (bearerUsername != requestedUsername) {
    throw new UnauthorizedError('invalid_token', { message: 'Cannot access data of
another user.' });
  }
  next();
}

export const getAuthenticatedUsername = (req) => {
  const token = getToken(req);
  const decoded = jwt.verify(token, "8sxYVG3yJG") as JwtPayload;
  return decoded?.username ?? '';
}

const getToken = (req) => {
  if (
    req.headers.authorization &&
    req.headers.authorization.split(" ")[0] === "Bearer"
  ) {
    return req.headers.authorization.split(" ")[1];
  } else if (req.query && req.query.token) {
    return req.query.token;
  }
  return null;
}

export const handleAuthorizationError = (err, req, res, next) => {
  if (err.name === "UnauthorizedError") {
    if (err.message == "Cannot access data of another user.") {
      return res.status(403).send({ error: 'Cannot access data of another user.' });
    }
    return res.status(401).send({ error: 'Authentication is required for this
operation.' });
  } else {
    next(err);
  }
};
```

A *getAuthenticatedUsername* függvényt a *UserController* osztály *getAuthenticated* metódusa hívja meg az aktuálisan bejelentkezett felhasználó felhasználói nevének lekérésére, a *handleAuthorizationError* pedig a router konfigurálásakor megadott hibakezelő metódus, ami azonosítatlan hozzáférési kísérlet esetén 401-es (Unauthorized), jogosulatlan hozzáférési kísérlet esetén pedig 403-as (Forbidden) státuszkódot küld vissza a kérelmezőnek.

2.4. Models

A projekt gyökerében létrehozott *models* jegyzék *index.ts* fájlja tartalmazza a kliens-szerver adatátvitelkor használt DTO-kat, ezeknek a frontenden használt, bővített változatait, egy dátumot adott alakra hozó *formatTimestamp* függvényt, valamint a szószedeteknél a színséma és a szint lehetséges értékeit megadó enumokat.

```
export enum Level {
  A1 = 'A1',
  A2 = 'A2',
  B1 = 'B1',
  B2 = 'B2',
  C1 = 'C1',
  C2 = 'C2',
  Other = 'other'
}

export enum Colour {
  Yellow = 'warning',
  Blue = 'primary',
  Grey = 'secondary',
  Turquoise = 'info',
  Red = 'danger',
  Green = 'success'
}

export function formatTimestamp(timestamp: string) {
  const date = new Date(timestamp);
  let options: Intl.DateTimeFormatOptions = {
    day: "numeric", month: "numeric", year: "numeric",
    hour: "2-digit", minute: "2-digit"
  };

  return date.toLocaleDateString("en-GB", options);
}
```

2.5. Service

A backend metódusainak hívására a frontenden a *src/app/services* mappa *Service* osztályai szolgálnak. Itt találhatók továbbá a felhasználói azonosítást menedzselő szolgáltató és interceptor függvények, valamint a pillanatnyilag mindössze a gyakorlások feladatszámának lekérésére használható *RevisionService* is.

A TopicService, a UserService és a PracticeService nagyon hasonló szerkezetűek, az */api* útvonalra adnak le GET, PUT, DELETE és POST HTTP kéréseket, melyek a megadott proxy konfiguráció miatt a localhost:4200-as portról automatikusan átirányításra kerülnek a localhost:3000-es portra, ahol a szerver szolgál ki. Az osztályok szerkezete az alábbi mintát követi, természetesen csak a szükséges metódusokkal:

```
@Injectable({
  providedIn: 'root'
})
export class TopicService {

  http = inject(HttpClient);

  getAll() {
     return this.http.get<TopicDTO[]>('api/topic');
  }

  getOne(id: string) {
    return this.http.get<TopicDTO>('api/topic/' + id);
  }

  create(topic: TopicDTO) {
    return this.http.post<TopicDTO>('api/topic', topic);
  }

  update(topic: TopicDTO) {
    return this.http.put<TopicDTO>('api/topic', topic);
  }

  delete(id: string) {
    return this.http.delete('api/topic/' + id);
  }
}
```

Az AuthService osztály a felhasználók azonosítását teszi lehetővé azáltal, hogy a bejelentkezéskor kapott JWT tokent eltárolja a böngésző lokális tárolójában. Ezt a tokent le lehet kérdezni, el lehet távolítani. Ez az osztály tartalmazza azokat a metódusokat is, amelyek a frontend egyes útvonalainak védelmére lettek felhasználva (például hogy csak bejelentkezett felhasználó tudja a topic-okat megtekinteni, de bejelentkezett felhasználó már ne férjen hozzá a bejelentkezés oldalhoz):

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private TOKEN_KEY = 'accessToken';
  private router = inject(Router);

  setToken(token: string) {
    localStorage.setItem(this.TOKEN_KEY, token);
  }
```

```
  getToken(): string | null {
    return localStorage.getItem(this.TOKEN_KEY);
  }

  removeToken() {
    localStorage.removeItem(this.TOKEN_KEY);
  }

  isLoggedIn(): boolean {
    return !!this.getToken();
  }

  preventGuestAccess(): boolean {
    const isLoggedIn = this.isLoggedIn();

    if (!isLoggedIn) {
      this.router.navigateByUrl('/login');
    }

    return isLoggedIn;
  }

  preventAuthAccess(): boolean {
    const isLoggedIn = this.isLoggedIn();

    if (isLoggedIn) {
      this.router.navigateByUrl('/');
    }

    return !isLoggedIn;
  }
}
```

app.routes.ts részlet:

```
{
    path: 'topics',
    component: TopicListComponent,
    canActivate: [ () => inject(AuthService).preventGuestAccess() ]
}
```

Az *accessTokenInterceptor* és az *unauthorizedInterceptor* middleware függvények a HTTP kliens konfigurálásakor kerülnek megadásra. Ezeken minden kliensről kimenő kérés átfut. Az *accessTokenInterceptor* feladata, hogy a localStorage-ban tárolt tokent, amennyiben az létezik, belehelyezi az */api* útvonalra kimenő kérések azonosítási fejlécébe. Az *unauthorizedInterceptor* a kimenő kérésekkel nem foglalkozik, hanem a beérkező választ figyeli, és amennyiben annak státuszkódja 401 vagy 403, átirányítással lekezeli a jogosultságsértést, érvénytelen token esetén eltávolítva azt a localStorage-ból.

2.6. Component

A frontend legfontosabb elemeit a következő Angular komponensek alkotják:

1. app:
   Befoglaló komponens, menüszalagot és láblécet jelenít meg.
2. login:
   Bejelentkezési felület.
3. practice-list
   Felhasználói profil és összes gyakorlás kijelzése, néhány összesítéssel.
4. practice-quiz
   Szódolgozat jellegű gyakorló feladat egy adott szószedethez, ellenőrzéssel, az eredmény megjelenítésével és mentésével.
5. register
   Regisztrációs felület, validált form.
6. topic-add
   Szószedet hozzáadására szolgáló felület, beágyazza a topic-formot.
7. topic-edit
   Szószedet módosítására szolgáló felület, beágyazza a topic-formot, átadva neki input paraméterként a szerkesztendő TopicDTO objektumot.
8. topic-form
   Form topic szerkesztésére (ha van értéke a topic input paraméternek) és hozzáadására (ha nem kap értéket a topic input paraméter).
9. topic-list
   Az összes szószedetet listázó felület, főoldal.
10. topic-stats
    Adott szószedetre vonatkozó gyakorlásokat összesítő felület Chart.js grafikonokkal.
11. word-form
    Form szó hozzáadására topic-hoz.
12. word-list
    Adott szószedet szavainak listázására szolgáló felület, beágyazza a word-formot.

A komponensekben a felhasználók tájékoztatásához az ngx-toastr könyvtár felugró üzeneteit (toast) alkalmaztam.

Néhány kódpélda:

practice.quiz.component.html

```
<div class="container" *ngIf="topic">
  <div class="row mb-4 mt-4">
    <div class="col-sm-6 mb-1 mt-1">
      <h2>
        Quiz
        <small class="ms-2 badge text-bg-{{topic.colour}}">{{topic.name}}</small>
      </h2>
    </div>
    <div class="col-sm-6 mb-1 mt-1">
      <div class="h-100 text-end d-flex justify-content-end align-items-center"
       *ngIf="isSolutionShown">
        <h4 style="margin: 0">
          <small class="badge text-bg-{{topic.colour}}">{{calculateScore()}} <span
```

```
            style="font-weight: normal;">points
             out of {{exerciceNumber}}</span></small>
        </h4>
      </div>
    </div>
  </div>
  <div class="row mb-2">
    <div class="col-6"><b>Word</b></div>
    <div class="col-6"><b>Meaning</b></div>
  </div>
  <form [formGroup]="wordForm">
    <div formArrayName="words">
      <div *ngFor="let wordControl of wordFormArray.controls; let i = index"
       [formGroupName]="i" class="row mb-1">
        <div class="col-6">
          <div class="form-group">
            <input type="text" [readOnly]="isSolutionShown" autocomplete="off"
             class="form-control input-sm"
              formControlName="target" [id]="'target-' + i"
              [ngClass]="{ 'crossed-out is-invalid': isSolutionShown &&
                    words[i].targetBlank && !words[i].correct, 'is-valid':
                    isSolutionShown && words[i].targetBlank && words[i].correct }"
              placement="top"
              [ngbTooltip]="words[i].targetBlank && !words[i].correct ?
                    'Solution: '+words[i].target : null"
              tooltipClass="my-custom-class">
          </div>
        </div>
        <div class="col-6">
          <div class="form-group">
            <input type="text" [readOnly]="isSolutionShown" autocomplete="off"
             class="form-control input-sm"
              formControlName="source" [id]="'source-' + i"
              [ngClass]="{ 'crossed-out is-invalid': isSolutionShown &&
                    !words[i].targetBlank && !words[i].correct, 'is-valid':
                    isSolutionShown && !words[i].targetBlank && words[i].correct }"
              placement="top"
              [ngbTooltip]="!words[i].targetBlank && !words[i].correct ?
                    'Solution: '+words[i].source : null"
              tooltipClass="my-custom-class">
          </div>
        </div>
      </div>
    </div>
    <div class="mt-4 text-center" [hidden]="isSolutionShown">
      <button type="submit" class="btn btn-{{topic.colour}}"
       (click)="submitSolution()">Submit solution</button>
    </div>
  </form>
  <div class="mt-4 text-center" [hidden]="!isSolutionShown">
```

```
    <button class="btn btn-{{topic.colour}}"
      (click)="navigateToStats()">Stats</button>
  </div>
</div>
```

practice.quiz.component.ts

```
import { Component, ViewEncapsulation, inject } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { TopicService } from '../services/topic.service';
import { PracticeService } from '../services/practice.service';
import { ExtendedWordDTO, PracticeDTO, TopicDTO, WordDTO } from '../../../models';
import { RevisionService } from '../services/revision.service';
import { CommonModule } from '@angular/common';
import { FormArray, FormBuilder, FormControl, FormGroup, ReactiveFormsModule } from
'@angular/forms';
import { NgbTooltipModule } from '@ng-bootstrap/ng-bootstrap';
import { UserService } from '../services/user.service';
import { ToastrService } from 'ngx-toastr';

@Component({
  selector: 'app-practice-quiz',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule, NgbTooltipModule],
  templateUrl: './practice-quiz.component.html',
  styleUrl: './practice-quiz.component.css',
  encapsulation: ViewEncapsulation.None,
})
export class PracticeQuizComponent {
  private router = inject(Router);
  private topicService = inject(TopicService);
  private practiceService = inject(PracticeService);
  private userService = inject(UserService);
  private currentRoute = inject(ActivatedRoute);
  private formBuilder = inject(FormBuilder);
  private toastr = inject(ToastrService);

  username = '';

  topic !: TopicDTO;
  words: ExtendedWordDTO[] = [];
  exerciceNumber = inject(RevisionService).getExerciceNum();
  wordForm = this.formBuilder.group({
    words: this.formBuilder.array<FormGroup>([])
  });
  isSolutionShown = false;

  ngOnInit(): void {
    const topicId = this.currentRoute.snapshot.params['id'];
    this.userService.getAuthenticated().subscribe({
```

```
    next: (user) => {
      this.username = user.username;
    },
    error: (err) => {
      this.toastr.error('Failed to identify current user.', 'Cannot load');
    }
  });
  this.topicService.getOne(topicId).subscribe({
    next: (topic) => {
      this.topic = topic;
      this.createExtendedWords(topic.words);
      this.initForm();
    },
    error: (err) => {
      this.toastr.error('Failed to load topic details due to a server error.',
      'Cannot load');
    }
  });
}

createExtendedWords(words: WordDTO[]) {
  if (words.length < this.exerciceNumber) {
    this.router.navigateByUrl('/topics');
    return;
  } else {
    const selectedWords = this.getRandomWords(words, this.exerciceNumber);
    this.words = selectedWords.map(word => this.convertToExtendedWordDTO(word));
  }
}

getRandomWords(words: WordDTO[], count: number): WordDTO[] {
  const shuffled = words.sort(() => 0.5 - Math.random());
  return shuffled.slice(0, count);
}

convertToExtendedWordDTO(word: WordDTO): ExtendedWordDTO {
  return {
    source: word.source,
    target: word.target,
    targetBlank: Math.random() < 0.75,
    answer: '',
    correct: true
  };
}

initForm() {
  const wordControls = this.words.map((word) => {
    return this.formBuilder.group({
      target : new FormControl({ value: word.targetBlank ? '' : word.target,
      disabled: !word.targetBlank }),
```

```
        source: new FormControl({ value: word.targetBlank ? word.source : '',
        disabled: word.targetBlank })
      });
    });

    const wordFormArray = this.formBuilder.array(wordControls);
    this.wordForm.setControl('words', wordFormArray);
  }

  get wordFormArray(): FormArray {
    return this.wordForm.get('words') as FormArray;
  }
  submitSolution() {
    if(this.isSolutionShown) return;
    // Access the array inside wordForm
    const wordsArray = this.wordFormArray.value;

    this.words.forEach((word, index) => {
      this.checkWord(wordsArray[index], word);
    });
    const practice = {
      timestamp: (new Date()).toISOString(),
      score: this.calculateScore(),
      username: this.username,
      topicId: this.topic._id.toString()
    }
    this.createPractice(practice as PracticeDTO);
  }

  createPractice(practice: PracticeDTO){
    this.practiceService.create(practice).subscribe({
      next: () => {
        this.isSolutionShown = true;
      },
      error: (err) => {
        this.toastr.error('Failed to save results due to a server error, try
again.', 'Error');
      }
    });
  }

  navigateToStats(){
    this.router.navigateByUrl('/topics/stats/'+this.topic._id.toString());
  }

  checkWord(toCheck: any, word: ExtendedWordDTO){
    if (word.targetBlank){
      word.answer = toCheck.target.trim();
      word.correct = word.answer == word.target;
    } else {
```

```
      word.answer = toCheck.source.trim();
      word.correct = word.answer == word.source;
    }
  }

  calculateScore(){
    return this.words.filter(word => word.correct).length;
  }
}
```

topic-stats.component.html

```
<div class="container" *ngIf="topic">
  <h2 class="mb-4 mt-4">
    {{topic.name}}
    <small class="ms-2 badge text-bg-{{topic.colour}}">{{topic.language}}</small>
  </h2>
  <div class="row">
    <div class="col-md-7 p-2">
      <div class="card text-bg-dark" [class.h-100]="practices.length != 0">
        <div class="card-body">
          <h5 class="card-title">Timeline</h5>
          <div class="card-text w-100">
            <div class="text-center" [hidden]="practices.length != 0">
              <i>You have not practised yet.</i>
            </div>
            <canvas #chart [hidden]="practices.length == 0"
            style="height:420px"></canvas>
          </div>
        </div>
      </div>
    </div>
    <div class="col-md-5 p-2">
      <div class="card text-bg-{{topic.colour}}">
        <div class="card-body bg-transparent">
          <h5 class="card-title">History</h5>
          <div class="card-text p-2" style="max-height: 200px; width: 100%;
          overflow-x:hidden; overflow-y: auto;">
            <div class="row text-center" *ngIf="practices.length == 0">
              <i>You have not practised yet.</i>
            </div>
            <div class="row" *ngIf="practices.length > 0">
              <div class="col-sm-8"><b>Date and time</b></div>
              <div class="col-sm-4 text-end"><b>Score</b></div>
            </div>
            @for(practice of practices; track $index){
            <div class="row pt-1">
              <div class="col-sm-8">{{formatTimestamp(practice.timestamp)}}</div>
              <div class="col-sm-4 text-end">
                {{practice.score}}/{{scoreMaximum}}
```

```
            </div>
          </div>
          }
        </div>
      </div>
    </div>
    <div class="card mt-4 text-bg-dark">
      <div class="card-body bg-transparent">
        <h5 class="card-title">Score distribution</h5>
        <div class="card-text">
          <div class="text-center" [hidden]="practices.length != 0">
            <i>You have not practised yet.</i>
          </div>
          <canvas #distChart style="height:175px"
          [hidden]="practices.length == 0"></canvas>
        </div>
      </div>
    </div>
  </div>
</div>
```

topic-stats.component.ts

```
import { Component, ElementRef, OnInit, ViewChild, inject } from '@angular/core';
import { PracticeService } from '../services/practice.service';
import { PracticeDTO, TopicDTO, convertColourToHex, formatTimestamp } from
'../../../models';
import { ActivatedRoute, Router } from '@angular/router';
import { TopicService } from '../services/topic.service';
import { CommonModule } from '@angular/common';
import { RevisionService } from '../services/revision.service';
import { Chart, registerables } from 'chart.js';
import 'chartjs-adapter-date-fns';
import zoomPlugin from 'chartjs-plugin-zoom';
import { UserService } from '../services/user.service';
import { ToastrService } from 'ngx-toastr';

@Component({
  selector: 'app-topic-stats',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './topic-stats.component.html',
  styleUrl: './topic-stats.component.css'
})
export class TopicStatsComponent implements OnInit {
  @ViewChild('chart') lineChart!: ElementRef<HTMLCanvasElement>;
  @ViewChild('distChart') distChart!: ElementRef<HTMLCanvasElement>;
```

```
private currentRoute = inject(ActivatedRoute);
private practiceService = inject(PracticeService);
private topicService = inject(TopicService);
private userService = inject(UserService);
private toastr = inject(ToastrService);
private router = inject(Router);

topic!: TopicDTO;
practices: PracticeDTO[] = [];
scoreMaximum = inject(RevisionService).getExerciceNum();
timelineChart !: Chart<any>;
distributionChart !: Chart<any>;

ngOnInit(): void {
  Chart.register(...registerables, zoomPlugin);
  const topicId = this.currentRoute.snapshot.params['id'];
  this.topicService.getOne(topicId).subscribe({
    next: (topic) => {
      this.topic = topic;
    },
    error: (err) => {
      this.toastr.error('Failed to load topic details due to a server error.',
      'Cannot load');
      this.router.navigateByUrl('/');
    }
  });
  this.userService.getAuthenticated().subscribe({
    next: (user) => {
      this.loadPractices(user.username);
    },
    error: (err) => {
      this.toastr.error('Failed to identify current user.', 'Cannot load');
      this.router.navigateByUrl('/');
    }
  });
}

loadPractices(username: string) {
  const topicId = this.currentRoute.snapshot.params['id'];
  this.practiceService.getFiltered(username, topicId).subscribe({
    next: (practices) => {
      practices.sort((a, b) => a.timestamp.localeCompare(b.timestamp));
      this.practices = practices;
    },
    error: (err) => {
      this.toastr.error('Failed to load practice details due to a server error.',
      'Cannot load');
      this.router.navigateByUrl('/');
    }
  });
```

```
}

ngAfterViewInit() {
  const interval = setInterval(() => {
    if (this.practices) {
      clearInterval(interval);
      if (this.practices.length > 0) {
        this.drawTimeline();
        this.drawDistributionChart();
      }
    }
  }, 100);
}

drawTimeline() {
  const ctx = this.lineChart.nativeElement.getContext('2d');

  if (!ctx) {
    throw new Error('Failed to get 2D context');
  }
  const data = this.practices.map(practice => ({
    x: new Date(practice.timestamp),
    y: practice.score
  }));

  this.timelineChart = new Chart(ctx, {
    type: 'line',
    data: {
      datasets: [{
        label: 'Your scores out of ' + this.scoreMaximum,
        backgroundColor: 'rgba(0, 0, 0, 0.1)',
        borderColor: convertColourToHex(this.topic.colour),
        data: data,
        fill: false
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      scales: {
        x: {
          type: 'time',
          time: {
            minUnit: 'minute',
            displayFormats: {
              hour: 'MMM dd H:mm',
              minute: 'MMM dd H:mm',
            }
          },
          title: {
```

```
        display: true,
        text: 'Date',
        color: 'white'
      },
      ticks: {
        color: 'white'
      }
    },
    y: {
      beginAtZero: true,
      max: this.scoreMaximum,
      title: {
        display: true,
        text: 'Score',
        color: 'white'
      },
      ticks: {
        color: 'white'
      }
    }
  },
  plugins: {
    legend: {
      labels: {
        color: 'white'
      }
    },
    tooltip: {
      titleColor: 'white',
      bodyColor: 'white',
      footerColor: 'white'
    },
    zoom: {
      limits: {
        x: { min: Date.parse('2024-01-01T00:00:00Z'), max:
        Date.parse((data[data.length - 1].x.getFullYear() + 1)
         + '-01-01T00:00:00Z') }
      },
      pan: {
        enabled: true,
        mode: 'x'
      },
      zoom: {
        wheel: {
          enabled: true
        },
        pinch: {
          enabled: true
        },
        mode: 'x'
```

```
          }
        }
      }
    }
  });
}

createFrequencyDistribution(): number[] {
  const scoreDistribution = new Array(this.scoreMaximum + 1).fill(0);
  this.practices.forEach(practice => {
    scoreDistribution[practice.score]++;
  });
  return scoreDistribution;
}

drawDistributionChart() {
  const ctx = this.distChart.nativeElement.getContext('2d') as
  CanvasRenderingContext2D;
  const frequencyDistribution = this.createFrequencyDistribution();
  const labels = Array.from({ length: this.scoreMaximum + 1 }, (_, i) =>
  i.toString());
  if (!ctx) {
    throw new Error('Failed to get 2D context');
  }
  this.distributionChart = new Chart(ctx, {
    type: 'bar',
    data: {
      labels: labels,
      datasets: [{
        label: 'Score distribution',
        backgroundColor: convertColourToHex(this.topic.colour),
        borderColor: convertColourToHex(this.topic.colour),
        data: frequencyDistribution,
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      scales: {
        x: {
          title: {
            display: true,
            text: 'Score',
            color: 'white'
          },
          type: 'linear',
          position: 'bottom',
          ticks: {
            stepSize: 1,
            color: 'white'
```

```
            }
          },
          y: {
            title: {
              display: true,
              text: 'Frequency',
              color: 'white'
            },
            beginAtZero: true,
            ticks: {
              stepSize: 1,
              color: 'white'
            }
          }
        },
        plugins: {
          legend: {
            labels: {
              color: 'white'
            }
          },
          tooltip: {
            titleColor: 'white',
            bodyColor: 'white',
            footerColor: 'white'
          }
        }
      }
    });
  }

  formatTimestamp(timestamp: string) {
    return formatTimestamp(timestamp);
  }

}
```