

Záródolgozat

Szakképesítés neve: Szoftverfejlesztő

OKJ szám: 54 213 05



Tanulmány, a tanulást segítő webalkalmazás

Készítette: Bollók Ákos Dániel

Budapest

2021

Tartalom

Bevezetés	3
A záródolgozat témájának indoklása	3
Felhasználói dokumentáció.....	4
Regisztráció.....	4
Belépés.....	4
Lecke létrehozása.....	5
Profil	11
Rendszerkövetelmények	12
Fejlesztői dokumentáció	12
Felhasznált technológiák bemutatása.....	12
MySQL	12
HTML (HyperText Markup Language, azaz hiperszöveges jelölőnyelv)	12
CSS (Cascading Style Sheets, azaz „Lépcsőzetes stíluslapok”)	12
Bootstrap	13
JavaScript.....	14
jQuery	14
PHP	14
JSON, azaz JavaScript Object Notation.....	14
Ajax, azaz Asynchronous JavaScript and XML	15
Adatbázis tervezése.....	15
A programkód tervezése	16
Szerver szekció	17
Connection, azaz kapcsolat	17
User, azaz felhasználó.....	18
Sets, azaz leckék	19
Cards, azaz kártyák	20
Topics, azaz témák.....	21
View, nézet	21

Navbars.php	21
Cards.php	22
Modals.php.....	22
practiceTemplates.php	22
statistics.php.....	22
profile.php.....	22
Kliens szekció	22
Functions.js	23
index.js	25
sets.js.....	26
cards.js	27
practice.js	29
style.css	30
Összegzés és tapasztalatok.....	30
Említésre méltó hibák és megoldásaik, tesztesetek.....	31
Továbbfejlesztési lehetőségek	31
Felhasznált irodalom.....	32
JavaScripthez és jQueryhez:	32
CSS-hez és Bootstraphez:	32
Html-hez:	32
PHP-hoz:	32
SQL-hez:	32

Bevezetés

A záródolgozat témájának indoklása

Záródolgozatom témája egy tanulást segítő webalkalmazás, mely a híres Quizlethez hasonló elven működik. Ez azt jelenti, hogy a felhasználók úgynevezett szókétyákat hozhatnak létre, melyeket leckékbe rendezhetnek és az oldal ezen leckékből különböző, memorizálást segítő játékokat hoz létre. A témaválasztásnál megkérdeztem egy egészségügyi szakon tanuló egyetemista barátomat, hogy mi segítene neki a tanulásban. Azt válaszolta, hogy egy olyan oldal, melyen anatómiához kapcsolódó képeket kell azonosítani, pl.: egy lábszárcsontot, medencecsontot és hasonlókat. Ekkor merült fel bennem egy kvízzjáték elkészítésének lehetősége. Mivel azonban a barátom ios rendszert használ, jómagam pedig elsősorban android felhasználó vagyok, így mind a fejlesztés, mind a tesztelés nehézkes lett volna a célplatformon. Ennek kiküszöbölésére döntöttem úgy, hogy webes környezetben készítem el a programot. Választhattam volna egy cross-platform nyelvet, mint a Java vagy egy úgynevezett Engine-t, mint a Unity, de rendelkeztem már webes tapasztalattal, így ezt találtam a leginkább célravezetőnek. Azonban egy program, mellyel csak képeket lehet memorizálni, nem feltétlenül elég ahhoz, hogy tanulásra alkalmas legyen, hiszen számtalan definíció és képlet is hozzátartozik egy egészségügyi hallgató tanulmányaihoz. Erre ráébredve döntöttem úgy, hogy első sorban egy szókétyák tanulását segítő programot készítek, hiszen az elv ugyanaz és ha az alapokat sikerül elkészíteni utána azt bármilyen feladat típusához könnyen lehet adaptálni. Ezen kétyák állítás - definíció párokból állnak.

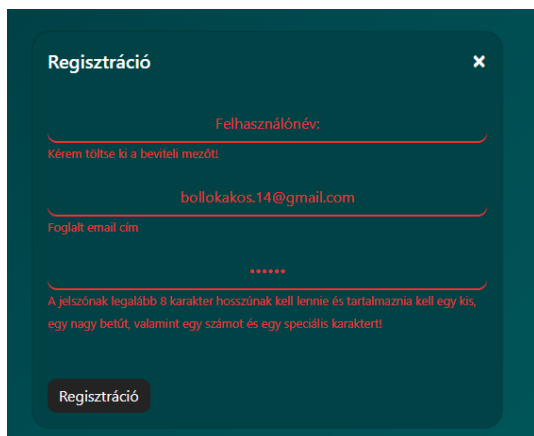
A program tervezésének első fázisát az adatok tárolásának megtervezése jelentette. Majd a program szerkezetének és végül a dizájnjának elkészítése.

Külön kihívást jelentett a kétyák „tanultsági szintjének” nyomon követése. Ehhez az úgynevezett Leitner módszert alkalmaztam, melynek lényege, hogy minden lecke rendelkezik egy doboz azonosítóval, pl.: 1, 2, 3. Ez mutatja, hogy a felhasználó mennyire ismeri az adott kétyán szereplő kifejezést a többi kétyán szereplőkhöz képest. Gyakorlásnál, amennyiben sikerül eltalálnia az adott kifejezést, akkor az egy dobozzal feljebb, ellenkező esetben pedig egy dobozzal lejjebb kerül. Ez azért fontos, mert a program a kisebb sorszámú kétyákat részesíti előnyben gyakorlásnál, így segítve a tanulást.

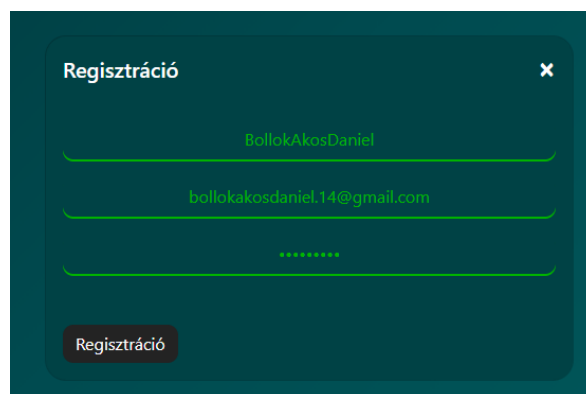
Felhasználói dokumentáció

Regisztráció

A főoldalon, a felső navigációs sávon látható „regisztráció” gombra kattintva megjelenik a regisztrációs űrlap. Itt kell megadnia a kívánt felhasználónevet, email címet és jelszót. A felhasználónév nem tartalmazhat speciális karaktereket. Az email címnek egyeznie kell a hivatalos formátummal (valaki@valami.com/.hu/stb.). A jelszónak legalább nyolc karakter hosszúnak kell lennie, és tartalmaznia kell legalább egy nagy betűt, számot és speciális karaktert. Amennyiben a megadott adatok nem felelnek meg a követelményeknek, akkor az alkalmazás nem engedi a regisztrációt és pirossal kiemelve a beviteli mezőket, alattuk megjeleníti a hozzájuk tartozó hibaiüzenetet. Helyes adatok esetén a beviteli mezőket zöld sáv veszi körül és véglegesíthető a regisztráció.

A screenshot of a registration form titled "Regisztráció" with a close button (X). The form has three input fields: "Felhasználónév:", "Foglalt email cím", and a password field with a strength indicator. The first and third fields are outlined in red, indicating errors. Below the first field, a red message says "Kérem töltse ki a beviteli mezőt!". Below the password field, a red message says "A jelszónak legalább 8 karakter hosszúnak kell lennie és tartalmaznia kell egy kis, egy nagy betűt, valamint egy számot és egy speciális karaktert!". A "Regisztráció" button is at the bottom.

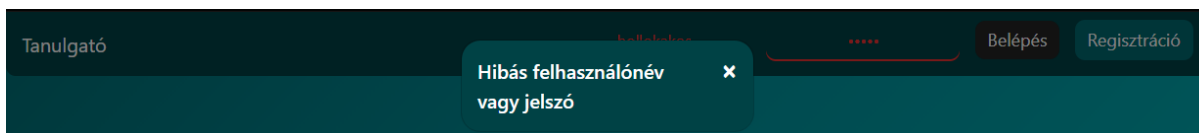
2. ábra Regisztráció – hibaiüzenet

A screenshot of the same registration form, but with successful data entry. The input fields are outlined in green. The "Felhasználónév:" field contains "BollokAkosDaniel", the "Foglalt email cím" field contains "bollokakosdaniel.14@gmail.com", and the password field contains "*****". A "Regisztráció" button is at the bottom.

1. ábra Regisztráció - helyes adatakkal

Belépés

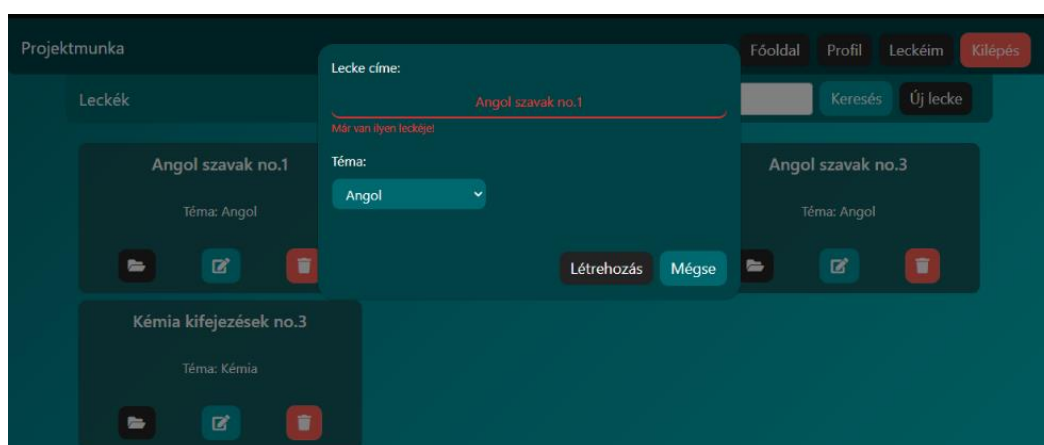
A belépéshez a navigációs sáv felhasználónév és jelszó mezőit kell kitöltenie helyes adatokkal, különben hibaiüzenet jelenik meg.

A screenshot of a login form titled "Tanulmány" with a close button (X). The form has two input fields: "Felhasználónév" and "Jelszó". The "Felhasználónév" field is outlined in red, indicating an error. A red message box says "Hibás felhasználónév vagy jelszó". There are "Belépés" and "Regisztráció" buttons at the bottom.

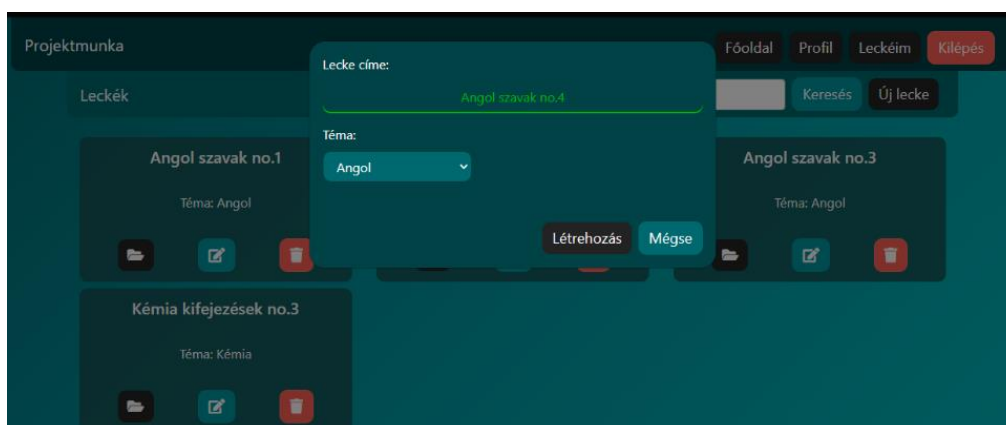
3. ábra Belépés - Hibás felhasználónév vagy jelszó

Lecke létrehozása

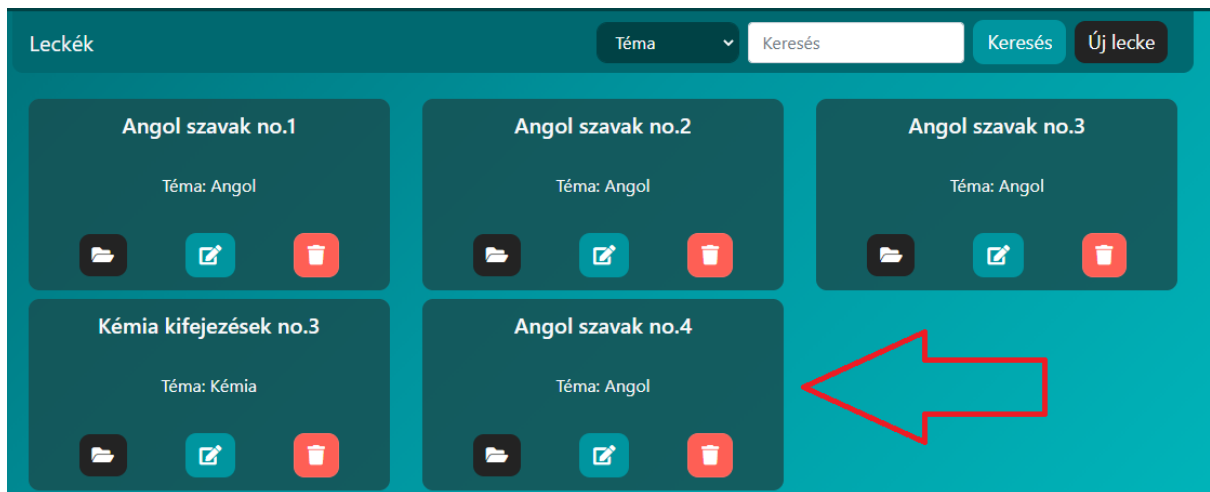
Sikeres belépés esetén a főoldal jelenik meg. A navigációs sávon kiválaszthatja, hogy melyik menüpontot kívánja megnyitni. A „Leckéim” pont alatt tudja kezelni a leckéket és a hozzájuk tartozó kártyákat. Itt egy második navigációs sávot is látni fog, melyen a leckéi között tud keresni, akár a témák szerint kiválasztani őket. Az új lecke gombra kattintva egy leckét létrehozó űrlap jelenik meg, melyben megadhatja, a létrehozni kívánt lecke címét és témáját. Amennyiben a létrehozni kívánt lecke címe megegyezik egy, már létező leckéjének címével, akkor hibaüzenetet kap és nem tudja létrehozni a kártyát. Ellenkező esetben a kártya létrejön és az oldal frissíti magát.



4. ábra Lecke létrehozása - hibaüzenet



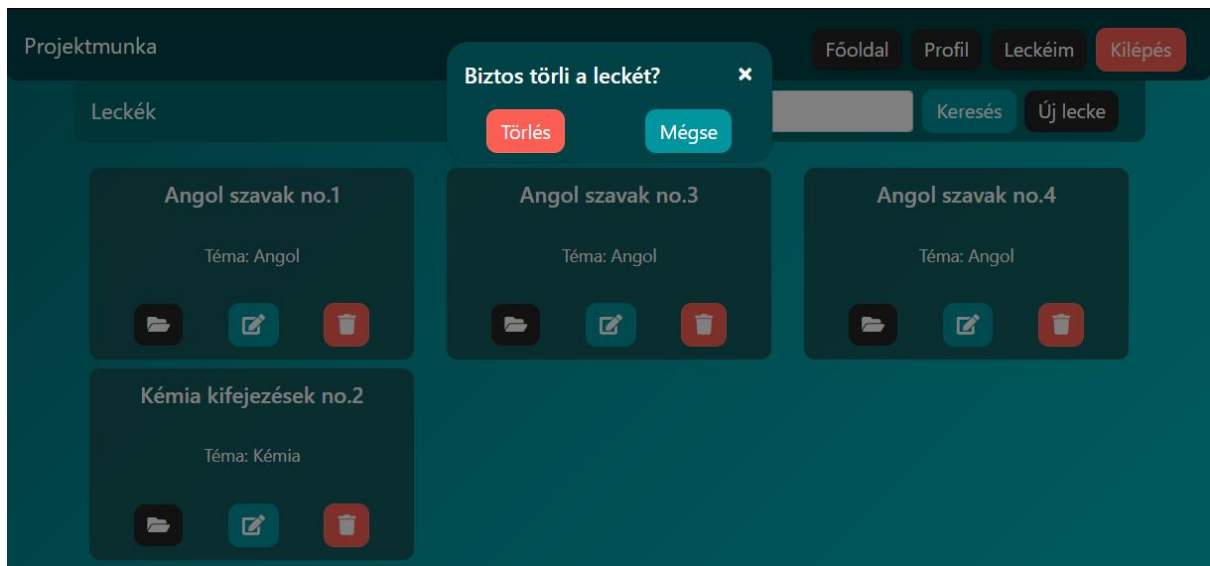
5. ábra Lecke létrehozása - Helyes adatok



6. ábra Létrehozott lecke

Minden lecke láblécében található három gomb, melyek balról jobbra haladva a megnyitás, szerkesztés és törlés funkciókért felelősek.

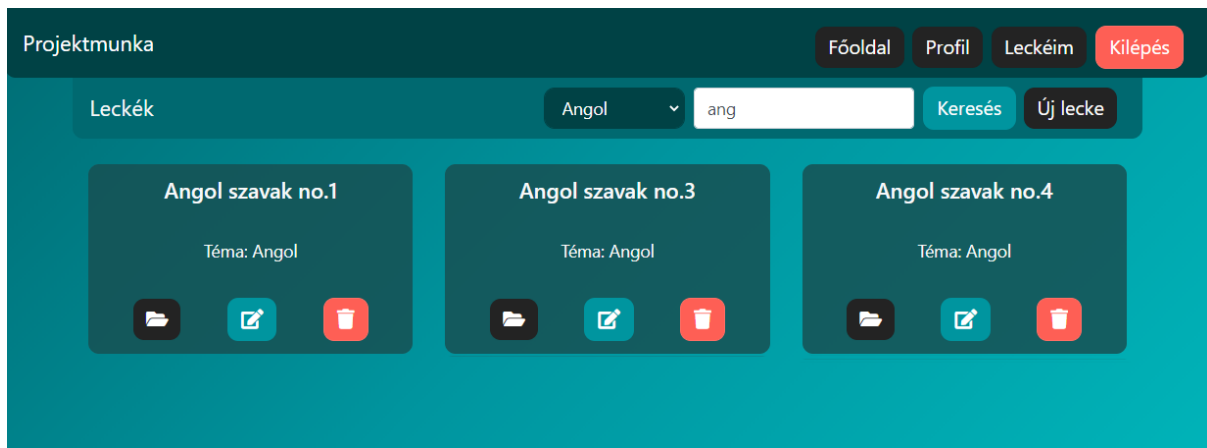
Törlés esetén egy információs ablak jelenik meg, melyben jóvá kell hagynia a törlési szándékát. Amennyiben véglegesen törli a leckét, akkor az eltűnik a képernyőn látható leckék közül, illetve az adatbázisból is törlődik.



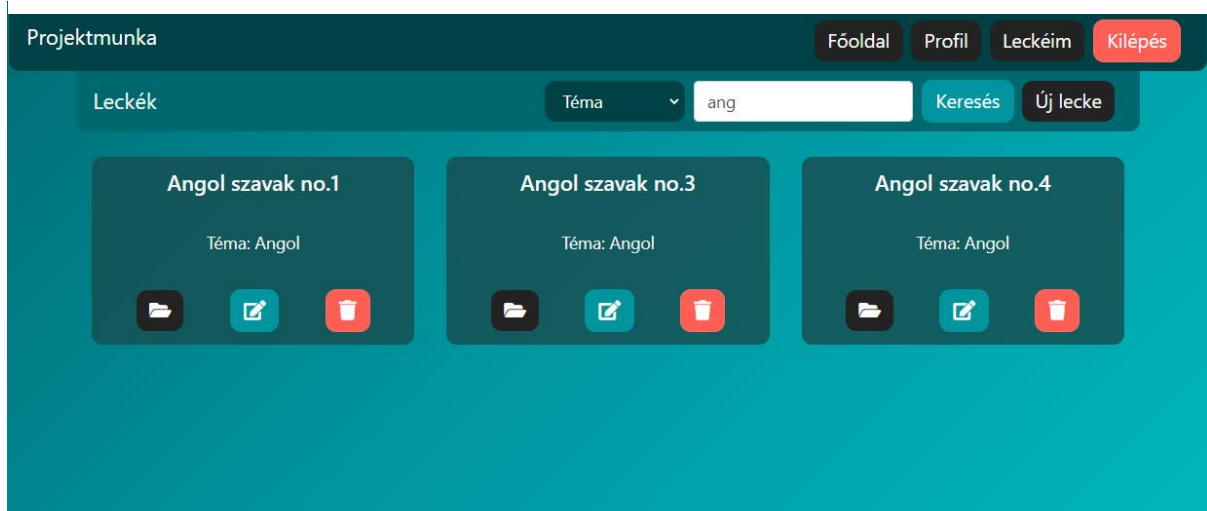
7. ábra Törlési szándék megerősítése

A lecke megnyitása gombra kattintva az oldal frissít és a leckéhez tartozó kártyákat jeleníti meg.

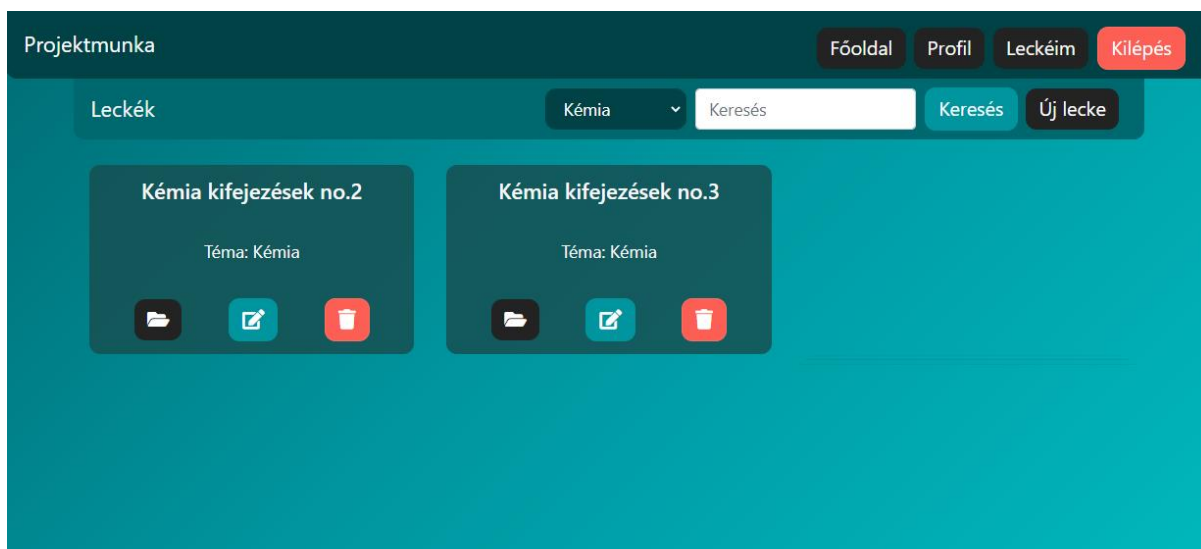
A leckék között lehet keresni is, illetve téma szerinti szűrőt beállítani. Ha van megadva keresendő szöveg, akkor minden olyan lecke megjelenik, melynek szövegében megtalálható az adott kifejezés. Ha van szűrő beállítva, akkor minden olyan lecke megjelenik, melyeknek témája megegyezik az adott témával. Ha mind szűrő, mind keresendő szöveg van beállítva, akkor csak a feltételeknek megfelelő leckék jelennek meg. Amennyiben se szűrő, se keresendő szöveg nincs beállítva, úgy, amennyiben rendelkezik leckékkel, akkor mindegyik megjelenik.



8. ábra Keresés - szöveggel és szűrővel

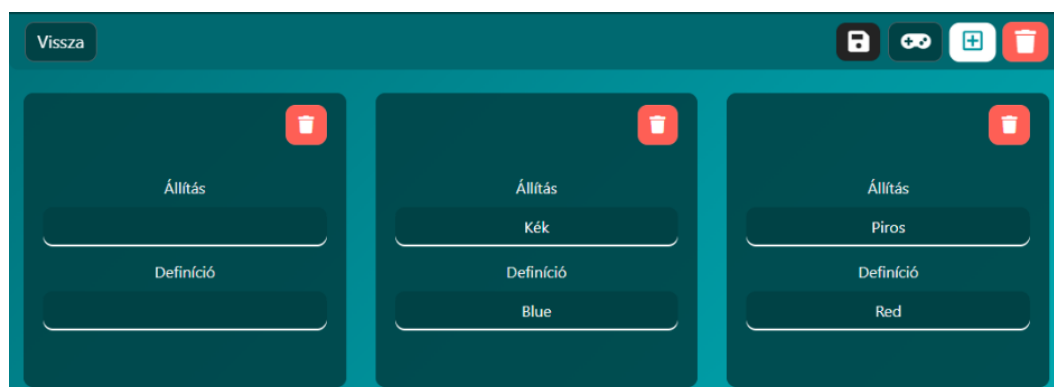


9. ábra Keresés - csak szöveggel



10. ábra Keresés - csak szűrővel

Lecke megnyitása esetén a második navigációs sáv frissül. Tartalma az alábbi négy funkcióért felelős gomb lesz; lecke mentése, lecke gyakorlása, kártya hozzáadása, lecke törlése. Új kártya létrehozása esetén egy üres kártya jelenik meg, melybe beírhatja az adott állítást és definíciót.



11. ábra Új kártya hozzáadása

Kártya törlése esetén az adott kártya elhalványodik, majd törlődik az oldalról, illetve az adatbázisból is.

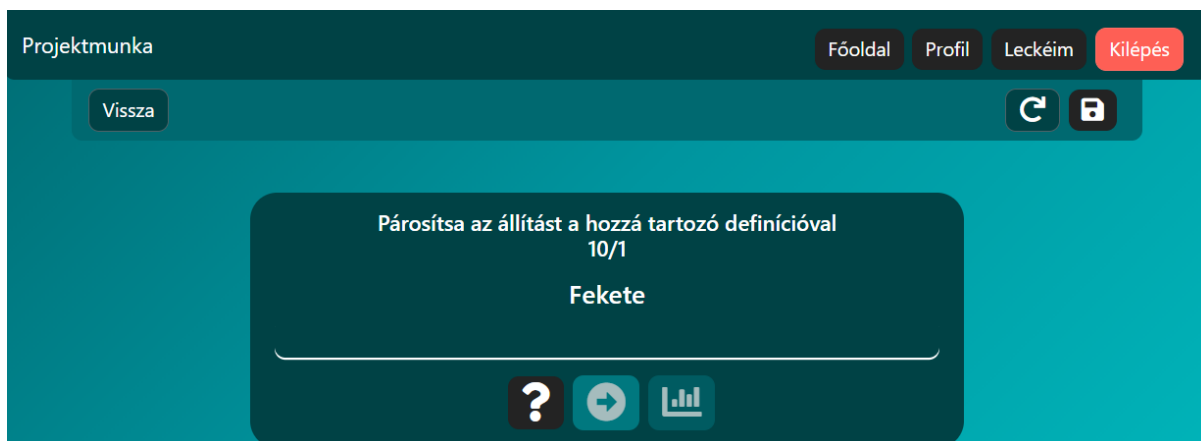
A kártyákat módosítás után el kell mentenie. Ha az oldalt el akarja hagyni és nem mentett adatai vannak, akkor egy hibaüzenet jelenik meg, és meg kell erősítenie a távozási szándékát. Ha elhagyja az adott oldalt, akkor az újonnan hozzáadott kártyák, illetve a módosított kártyák tartalmai nem kerülnek eltárolásra.



12. ábra Mentésre figyelmeztető üzenet

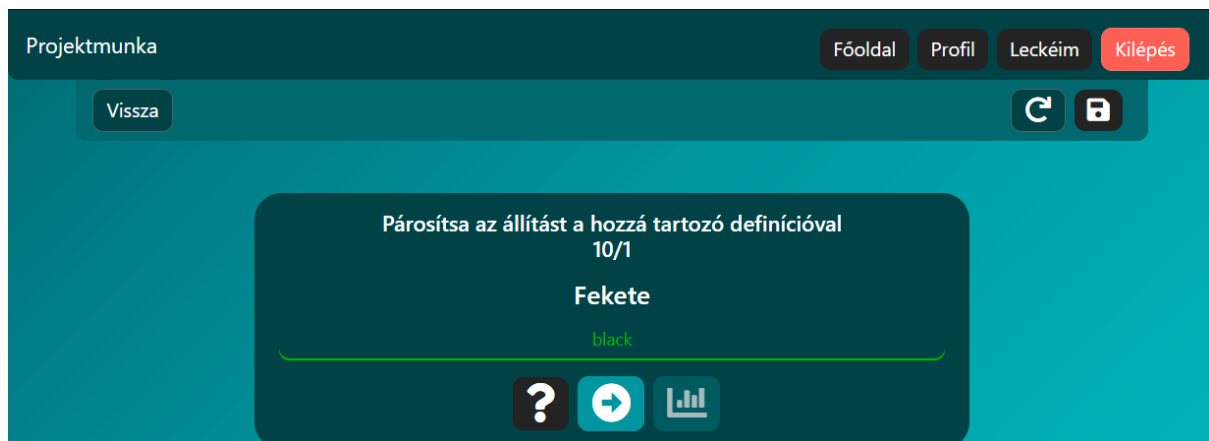
Kártya mentése esetén az adatok frissülnek, az újonnan hozzáadott kártya az oldal aljára kerül.

A „lecke gyakorlása” funkciójú gombra kattintva az oldalon megjelenik egy „doboz”, melyben található a párosítandó adatok állítás része, egy beviteli mező (ide kell beírni a megadandó szóhoz tartozó definíciót), illetve három gomb.

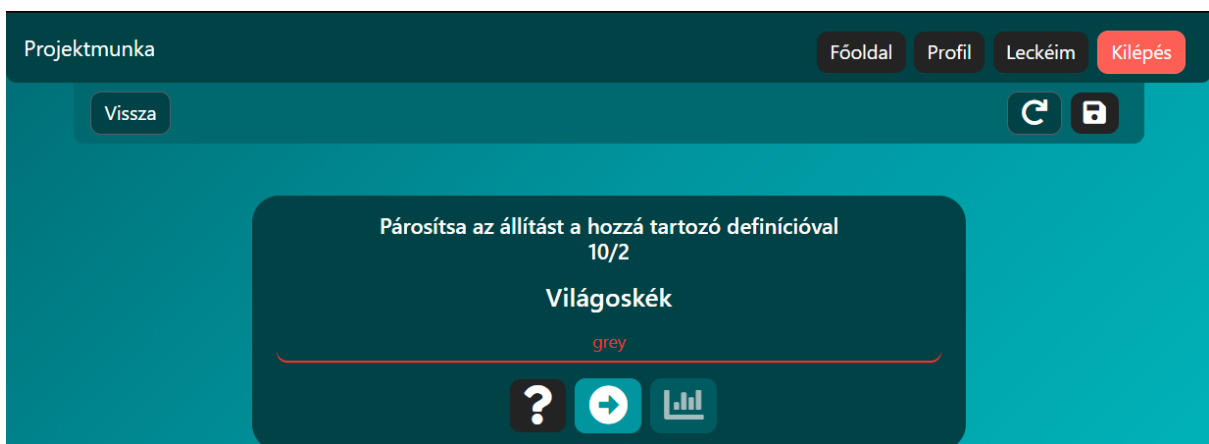


13. ábra Gyakorló ablak

Az első gomb, melyen egy kérdőjel ikonja látható, felelős a beírt szó ellenőrzéséért. Csak akkor lehet rákattintani, ha a beviteli mező nem üres. Az ellenőrzés az enter billentyű lenyomására is megtörténik. Ha helyes szót írt be, akkor a beviteli mező zöld keretet kap, ellenkező esetben piros lesz.



14. ábra Gyakorló ablak - helyes válasz



15. ábra Gyakorló ablak - hibás válasz

A mellette található gombbal lehet, választadás után, a következő szóra lépni. Balról az utolsó gombbal lehet statisztikát kérni a gyakorlás végén. A statisztika megjeleníti az eltalált és elhibázott szókétyák állítás-definíció páryait.

Projektmunka			Főoldal	Profil	Leckéim	Kilépés
Visza						
Eltalált szavak						
	Állítás	Definíció				
1	Fekete	Black				
2	Barna	Brown				
3	Szürke	Grey				
4	Fehér	White				
5	rózsaszín	pink				
6	Zöld	Green				
7	Lila	Purple				
8	Narancssárga	Orange				
Elhibázott szavak						
	Állítás	Definíció				
1	Világoskék	Lightblue				
2	Cián	Cyan				

16. ábra Statisztika

A gyakorlás állását ugyanúgy el kell mentenie, ha mentetlen állással rendelkezik, akkor itt is egy hibaüzenetet kap és meg kell erősítenie a kilépési szándékát.

Profil

A profil gombra kattintva egy űrlap jelenik meg, ahol módosíthatja adatait (felhasználónév, email cím, jelszó).

17. ábra Profilt módosító ablak

Rendszerkövetelmények

Webalkalmazás lévén a programnak nincs nagy rendszerkövetelménye és külön telepítést sem igényel. Egy böngészőre és internetkapcsolatra van csupán szükség hozzá. Azonban a jQuery és a Bootstrap keretrendszerek csak bizonyos böngészőket támogatnak. A böngésző verziója lehetőleg legyen a legfrissebtől egyvel korábbi verziószerű.

Fejlesztői dokumentáció

Felhasznált technológiák bemutatása

MySQL

Az adatbázishoz a MySQL-re esett a választásom, mely egy relációs adatbáziskezelő rendszer. Ez azt jelenti, hogy az adatok, logikailag csoportosítva, úgynevezett táblákba rendezve tárolódnak a szerveren. Ezek a táblák egy szokásos táblázathoz hasonlíthatóak. Köztük kapcsolat létesíthető, mely kritikus fontosságú, mivel ezáltal a logikailag kapcsolódó, de nem szorosan összetartozó adatok külön táblába rendezhetőek, átláthatóbbá téve az adatbázist.

Mivel webes környezetben készítettem a programot, így a HTML, a CSS és a JavaScript felhasználása adekvát volt. CSS-hez a Bootstrap, Javascript-hez pedig a jQuery keretrendszert használtam. Szerveroldali nyelvnek a PHP-t választottam, mivel már rendelkeztem a nyelv alapjainak ismeretével.

HTML (HyperText Markup Language, azaz hiperszöveges jelölőnyelv)

Egy leíró nyelv, melyet weboldalak készítéséhez fejlesztettek ki és ma már internetes szabványnak számít. Lényege, hogy a weboldal, mint dokumentum, úgynevezett tag-ekből (ejtsd: „teg”), azaz címkékből áll. Tekintve, hogy a programnyelvek többsége angolul íródott, s íródik mind a mai napig, ezért dolgozatomban olykor elkerülhetetlen, hogy angol szakkifejezéseket használjak, de igyekszek mindenhol fordítással szolgálni.

CSS (Cascading Style Sheets, azaz „Lépcsőzetes stíluslapok”)

A html önmagában nem elegendő egy piacképes weboldal, webalkalmazás elkészítéséhez, mivel a címkék csak az oldal gerincét alkotják. Azért, hogy felhasználóbarát megjelenése legyen egy adott oldanak, stíluslapot, esetleg többet is, szokás hozzá csatolni. A

stíluslapon belül az oldalon található címkékre hivatkozhatunk. Ennek a hivatkozásnak többféle módja is létezik, melyek erősségük szerint szintekre oszthatók. Még nem esett szó a címkék két, stílusbeállítás szempontjából fontos tulajdonságáról. Ezek az `id` (azonosító) és a `class` (osztály). Fontos, hogy adott azonosítóval csak egy címke látható el, adott osztállyal viszont korlátlan mennyiségű. Ezek a jelölők a stíluslapon található hivatkozásnál játszanak szerepet, ugyanis alapvetően három módon hivatkozhatunk egy címke, ezek erősség szerinti növekvő sorrendben a következők: `tipus` (azaz a címke típusa, pl.: `div`, `span`), `osztály` és `azonosító`. A hivatkozás erősségétől függően fogja a böngésző a címkehez társítani a stílusbeállítást. Azaz, ha pl.: egy `div` (doboz) elem rendelkezik osztállyal és azonosítóval is, és ezek között átfedés van (pl.: mindkettőre be van állítva a háttérszín), akkora az erősebb hivatkozásban meghatározott beállítást alkalmazza majd a böngésző a címke. Ezen felül készíthetünk hierarchikus hivatkozást is a stíluslapon, melyben meghatározhatjuk, hogy az adott beállítások, pl.: egy `div`-en belüli `div`-en belüli űrlapra vonatkozzanak. Ezen felül a csatolt stíluslapok sorrendje is szerepet játszik. Kijelenthető, hogy azonos erősségű hivatkozások esetén a böngésző a fordítás során később talált stílusbeállítást fogja alkalmazni. Ez azt jelenti, hogy ha két különböző stíluslapot is az oldalunkhoz kapcsolunk, akkor a később kapcsolt lapon belüli beállítások mennek végbe. Ez azért történik, mert a böngésző futási időben, sorról-sorra fordítja az oldal kódját. Amennyiben azt szeretnénk, hogy egy adott stílusbeállítás lépjen érvénybe minden esetben, úgy az adott beállítás után kell írni az „!important” kulcsszót. Így a böngésző minden esetben az érintett stílust fogja alkalmazni, függetlenül a stíluslapok hierarchiájától és a hivatkozások erősségétől.

Bootstrap

Ma már nem feltétlen szükséges egy weboldalhoz minden CSS beállítást a fejlesztőnek megírnia, ugyanis az évek során készültek úgynevezett keretrendszerek, melyek közül a legelterjedtebb a Bootstrap. Egy keretrendszer egy már létező nyelvre épül rá, egyszerűsítve annak funkcióit. A Bootstrap számtalan, előre deklarált osztállyal rendelkezik, melyek között találhatók mind az űrlapokra, mind a navigációs sávokra és egyéb, alapvetően kényes beállításokat igénylő problémákra elkészített sablonokkal. Ezen felül rendelkezik egy úgynevezett „grid-system”-el, melynek jelentése: keretrendszer. Ez teszi lehetővé az oldal tartalmának könnyű és reszponzív felosztását. Reszponzívnek nevezhető egy oldal, amennyiben a tartalmának elrendezése kijelzőméret szerint változik. A reszponzivitás az okostelefonok megjelenését követően vált fontos kritériummá, ugyanis addig minden weboldal ugyan azzal a megjelenéssel rendelkezett kis, közepes és nagy kijelzőkön is. Azonban, ma már,

amennyiben piacképes oldalt szeretnénk készíteni, elengedhetetlen, hogy az reszponzív legyen. A már említett „grid-system” ezt az alábbi módon teszi lehetővé; Az oldal tartalma egy container (tároló) osztállyal rendelkező div-ben van. Ezen belül row (sor) osztályú divek találhatók, melyekben pedig col (column, azaz oszlop) osztállyal ellátott div-ek foglalnak helyet. Ez azt jelenti, hogy az oldal tartalma sorokra és a sorokon belüli oszlopokra van osztva. Egy sort tizenkettő képzeletbeli egységre lehet osztani és az oszlop osztályban megadhatjuk, hogy ebből a tizenkettőből az adott oszlop mennyit foglaljon el, pl.: A col-6 azt jelenti, hogy a tizenkettőből hatot, azaz a sor felét foglalja el az adott oszlop. Ezen jelölőkben az is megadható, hogy mekkora kijelzőn lépjenek érvénybe, így pl.: ami egy okostelefonon két egységet foglal el, az egy asztali számítógépen négyet: `<div class="col-xs-2 col-lg4"></div>` .

JavaScript

A JavaScript egy magasszintű (2015 óta az Objektum Orientáltságot is támogató) szkriptnyelv. A böngészőbe épített JavaScript engine (motor) fordítja le futási időben. Segítségével egy oldal dinamikussá tehető. Ez azt jelenti, hogy a különböző, többnyire a felhasználó által elindított eseményekhez, pl.: kattintás, valamilyen kódot rendelünk, mely változást idéz elő az oldalon. Ez lehet egy egyszerű színváltoztatástól kezdve, egészen a szerverre történő adatküldésig bármi. Ma már az oldal reszponzivitása mellett a dinamikusság is alapvető követelménynek számít. JavaScript-ben is a már említett azonosító és osztály jelölőkkel hivatkozhatunk a html címkékre és hozzájuk úgynevezett eventlistener (eseménykezelő) függvényeket kapcsolhatunk. Ezekben a függvényekben adható meg, hogy adott eseményre mi történjen az oldalon.

jQuery

Akárcsak a CSS, a JavaScript is rendelkezik számos keretrendszerrel. Ezek egyike a jQuery, mely azért vált népszerűvé, mert a html elemekre történő hivatkozást, illetve az eseménykezelők létrehozását egyszerűsítette le drasztikus módon.

PHP

A PHP egy szerveroldali szkriptnyelv, mellyel az oldal dinamikussá tehető, illetve adatbázishoz kapcsolható. Segítségével az oldalba, külső fájl használata nélkül, kód ágyazható. A programkódot a szerver php motorja fordítja le. Támogatja az OOP szemléletet.

JSON, azaz JavaScript Object Notation

A JSON egy, az interneten történő adattovábbításra használt, minimális helyigényű, ezáltal effektívnek mondható adatszerkezet. Kulcs-érték (key-value) párokba rendezett

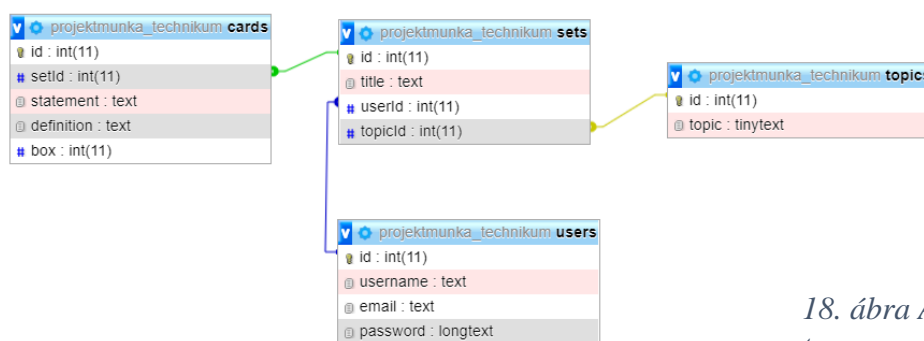
objektumokból áll. Ezen objektumok többszörösen egymásba ágyazhatók, ezáltal lehetővé téve komplex adatok gyors, minimális helyigényű továbbítását.

Ajax, azaz Asynchronous JavaScript and XML

A szerverrel alapvetően két módon lehet információt cserélni. Ezek a szinkronos és az aszinkronos kommunikáció. Előbbi mára már viszonylag elavultnak minősül és webalkalmazásokban nem, vagy csak ritkán használják, mivel rontja a felhasználói élményt. Lényege, hogy amennyiben adatot szeretnénk cserélni a szerverrel az oldalt újra kell tölteni. Ezzel szemben az aszinkronos kommunikáció a háttérben zajlik és nem igényel oldalfrissítést, növelve mind az oldal hatékonyságát, mind a felhasználói élményt. Aszinkronos lekérdezések végrehajtásához webes környezetben az úgynevezett AJAX ((Asynchronous JavaScript and XML) technikát használják. Vanilla JavaScript (ezentúl Vanilla Js), azaz keretrendszer nélküli JavaScript használata esetén ez egy viszonylag bonyolult algoritmust kíván meg ezért én a jQuery könyvtár beépített \$.ajax() metódusát használtam.

Adatbázis tervezése

Az adatbázis esetében a legnagyobb kihívást az jelentette, hogy miként tároljam el a felhasználó leckéit és az azon belül a kártyákat, illetve a kártyákban található állítás-definíció párokat. Jelenleg öt táblából áll az adatbázis, melyek 1-n, azaz egy a többhöz kapcsolattal csatlakoznak egymáshoz. A users (felhasználók) táblában tárolódik az azonosító, felhasználónév, email cím és a jelszó is, ez utóbbi természetesen titkosítva van. Ehhez kapcsolódik a sets, azaz a leckéket tároló tábla, melyben a leckék tárolódnak. Minden leckének van egy címe, témája és felhasználója. A topicId (téma azonosító) mező a topics (témák) tábla, míg a userId (felhasználóazonosító) a users tábla id (azonosító) mezőjére tartalmaz referenciát. A leckékre a cards (kártyák) tábla setId (lecke azonosító) mezője hivatkozik. A cards tábla ezen felül rendelkezik egy id, statement (állítás) és definition (definíció) mezővel is.



18. ábra Adatbázis
terve

A programkód tervezése

A fejlesztés során először egy klasszikus megközelítést választottam, melynek lényege, hogy a különböző funkciók különböző oldalakon találhatók. Ez azt jelenti, hogy volt egy oldal a belépés és regisztráció számára, egy a leckék, a kártyák és a felhasználói adatok kezelésére. Később azonban úgy döntöttem, hogy egy modernebb megközelítésre váltok, mely az SPA, azaz Single Page Application nevet viseli. A webalkalmazások, nagy része például a Facebook, és a Netflix is ezen az elven működik. Lényege, hogy az alkalmazás egy oldalból áll, melynek tartalma dinamikusan változik a felhasználói interakciók függvényében. Kisebb problémát jelentett, hogy míg az említett oldalak a React.js könyvtárat és Node.js környezetet használják, melyek a JavaScriptre épülnek és kifejezetten webalkalmazások fejlesztéséhez készültek, addig az általam ismert és használt jQuery és PHP inkább a klasszikus weboldalak készítéséhez optimálisak. Ez azonban nem tántorított el az ötlettől, maradtam az SPA megvalósítás mellett.

Az alkalmazást az MVC, azaz Model-View-Control (Modell-Nézet-Vezérlő) programtervezési minta alapján igyekeztem elkészíteni. Ennek két megvalósítása lehetséges. Az általam használt az, melyben a felhasználó csak a vezérlővel áll kapcsolatban, a Nézet és a Modell részekhez nincs közvetlen hozzáférése. A Modellben az alkalmazás által használt információk, míg a Nézet elemében az adatok megjelenítésének mikéntje kerül meghatározásra. A Vezérlő ezen részek és a felhasználó által közvetít, az utóbbi által végrehajtott interakciók alapján.

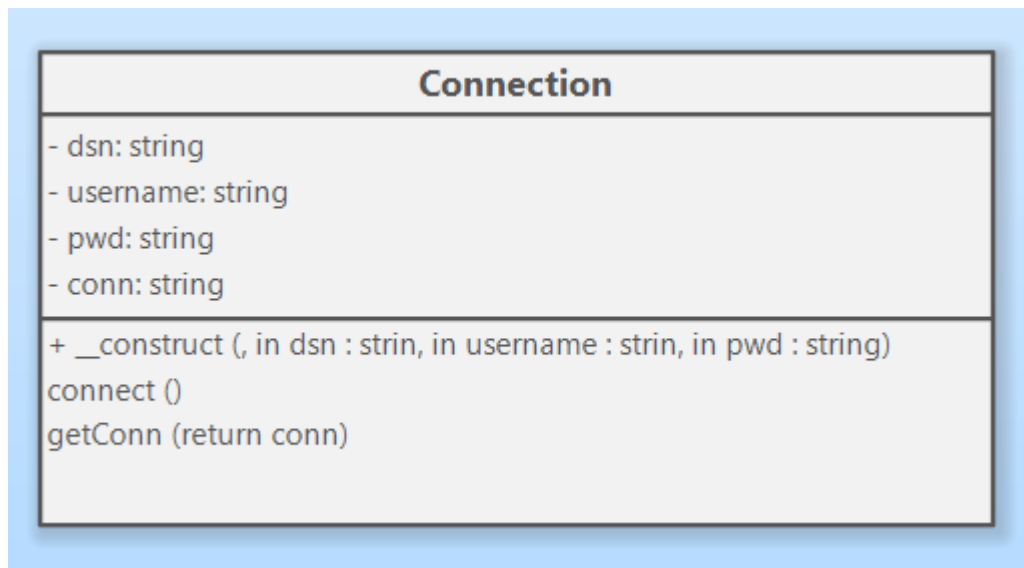
A kódot logikailag két részre osztottam, ezek a client és a server. Előbbibe került a kliens által látható oldal és a hozzá tartozó szkriptek, stíluslapok, utóbbiba pedig az MVC modell mentén felépített kódok, melyek a háttérben futnak és kapcsolatban állnak az adatbázissal.

A Vezérlőben található egy `process.php` fájl, mely a kliens felől érkező kéréseket (request) dolgozza fel és hívja meg a szükséges kódokat. Az ezen kódokat tartalmazó fájlokra a `process.php` első soraiban történik hivatkozás a `require_once()` metódus segítségével.

PHP-ban az `include()` és a `require()` metódusok szolgálnak a fájlok összekapcsolására, melyek között az a különbség, hogy előbbi a fájl hiányában is megkísérel lefutni, utóbbi viszont hibát dob. Ezáltal célszerű az utóbbit választani, megelőzve a lehetséges hibák előfordulását. A külső fájlok importálását követően hívódik meg a `session_start()` metódus. Ennek indoklása a tesztelési dokumentáció részben található.

Szerver szekció

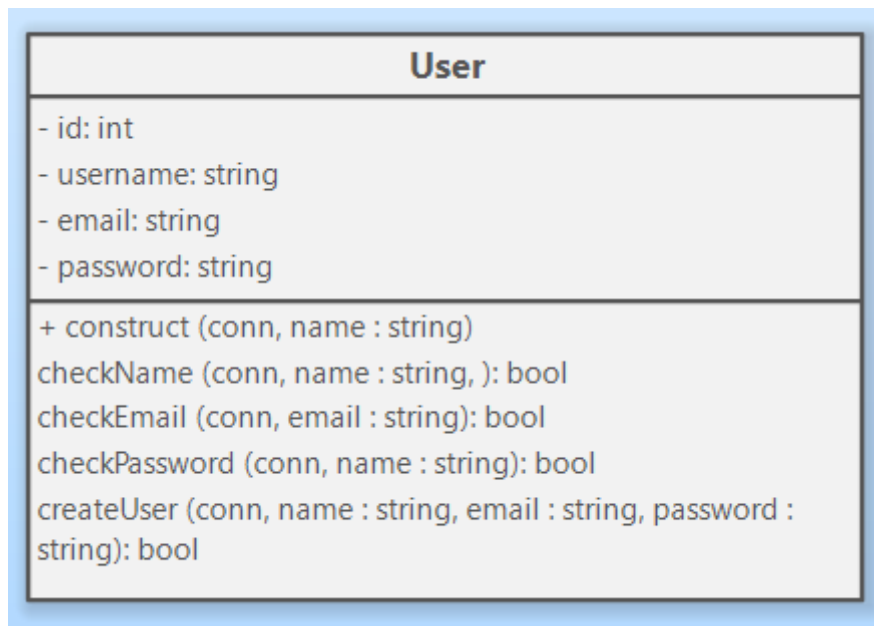
Connection, azaz kapcsolat



19. ábra Connection osztály

A Connection osztály felelős az adatbázissal való kapcsolat fenntartásáért. PHP nyelven többféle lehetőségünk van a kapcsolat létesítésére, én a PDO-t (PHP Data Object) választottam, melynek előnye, hogy adatbázis-kezelőnyelv váltása esetén nem szükséges a teljes kód újraírása, elegendő bizonyos részeket újraírni, továbbá támogatja a prepared statement névre hallgató technikát. Ennek lényege, hogy az adott sql utasításokat előre deklaráljuk, viszont a lekérdezendő, rögzítendő vagy módosítandó adatokat csak később társítjuk hozzájuk. Ez az úgynevezett SQL injection támadások elleni védekezésben hasznos. Efféle támadás esetén a támadó, felismerve a serveren tárolt sql lekérdezés szerkezetét azt meg tudja szakítani és egy hozzáírt allekérdezéssel végzetes károkat okozhat az adatbázisunkban, az adatok ellopásáról nem is beszélve. A Connection osztály négy adattaggal, azaz attribútummal rendelkezik. Ezek a kapcsolathoz szükséges dsn, mely az adatbázis típusát és szolgáltatóját tárolja, a dbname, melyben az adatbázis neve kerül tárolásra és a felhasználónév, valamint a jelszó, melyekkel az adatbázishoz kapcsolódunk. Ezen adatok az osztály konstruktorában kerülnek átadásra, majd egy try-catch blokkon belül az osztályból készített példányon, meghívásra kerül annak connect() metódusa, mely létrehozza az adatbázissal a kapcsolatot.

User, azaz felhasználó

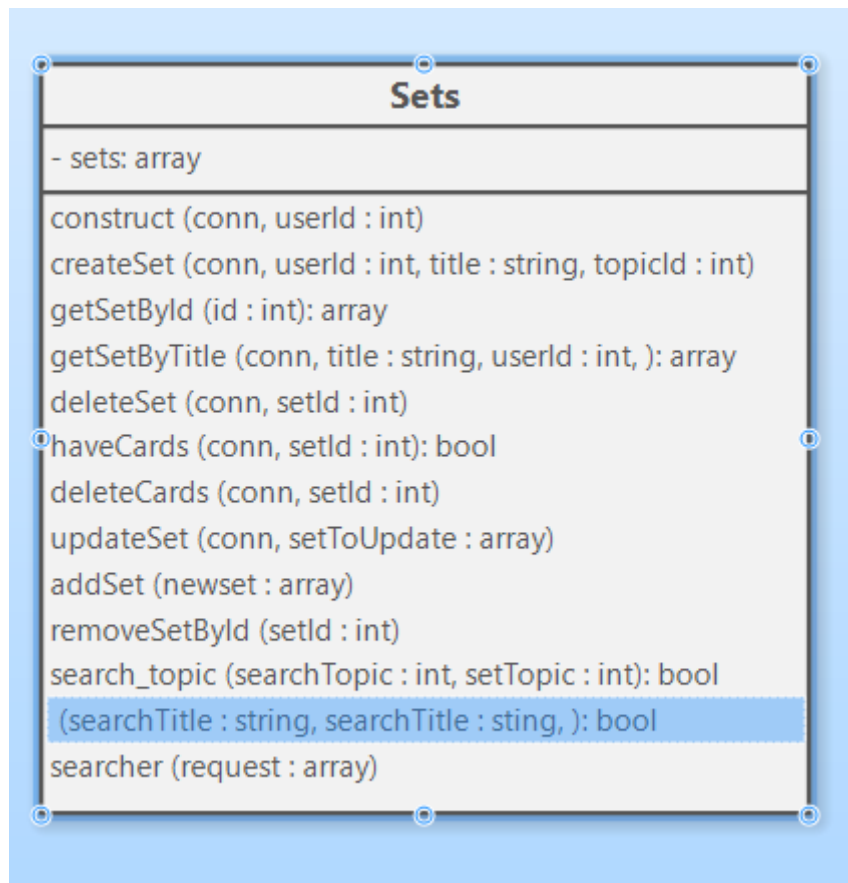


20. ábra User osztály

A User osztály négy attribútummal rendelkezik, melyek a felhasználói adatokat tárolják; id, username, email, password (azonosító, felhasználónév, email cím és jelszó). Ezek az osztály konstruktorában kapnak értéket, viszont ehhez a konstruktor csak egy, a kapcsolatért felelős conn változót, valamint a felhasználó nevét tartalmazó name változót vesz át. Azért elegendő csak ezen két változó, mert az osztály példányosítása csak a belépéskor történik meg, melynek során az osztályban található statikus metódusokkal a felhasználó által beírt felhasználónév és jelszó ellenőrzésre kerül. Az osztály példányosítása csak akkor megy vége, ha a megadott adatok helyesek, ellenben hiba keletkezik. Az adatok ellenőrzésére tökéletesek az említett statikus metódusok, mert ezek meghívásához nem szükséges osztálypéldányt létrehozni, meghívhatók az osztály nevéen keresztül is. Értelemszerűen mindegyik statikus metódus az általa ellenőrizendő adatot kapja bemeneti paraméterként, visszatérése pedig logikai érték lesz, az adatbázisból való lekérdezés alapján, pl.: a checkName() metódus a felhasználónevet ellenőrzi és, amennyiben az adatbázisban egyezést talál úgy igaz, ellenkező esetben pedig hamis értékkel tér vissza. Ez azért lényeges, mivel így ezen metódusok mind a regisztráció, mind a belépés során felhasználhatóak. Az osztály rendelkezik egy további statikus metódussal, ez a createUser(), mely a regisztráció esetén, amennyiben nincs egyezés az adatbázisban található felhasználónevek és email címek, valamint a felhasználó által megadott adatok között, meghívásra kerül és rögzíti az új felhasználót. Az osztály adatainak módosítására is

rendelkezésre állnak metódusok, melyek akkor kerülnek meghívásra, amikor a felhasználó módosítani kívánja adatait. Ezen metódusok is ellenőrzik az adatbázisban található adatokat és a módosítás csak akkor történik meg, ha nincs egyezés.

Sets, azaz leckék



21. ábra Sets osztály

A leckéket tartalmazó osztály egy adattaggal rendelkezik, mely egy, a leckéket tároló tömb. Konstruktorában az adatbázissal való kapcsolatért felelős conn, valamint a felhasználót azonosító userId változókat veszi át és kérdezi le, majd tárolja el a felhasználó leckéit. Ezen osztály számos metódussal rendelkezik, melyek a leckéken végzendő műveletek végrehajtásáért (hozzáadás, módosítás, törlés, keresés), valamint azok segítéséért felelősek.

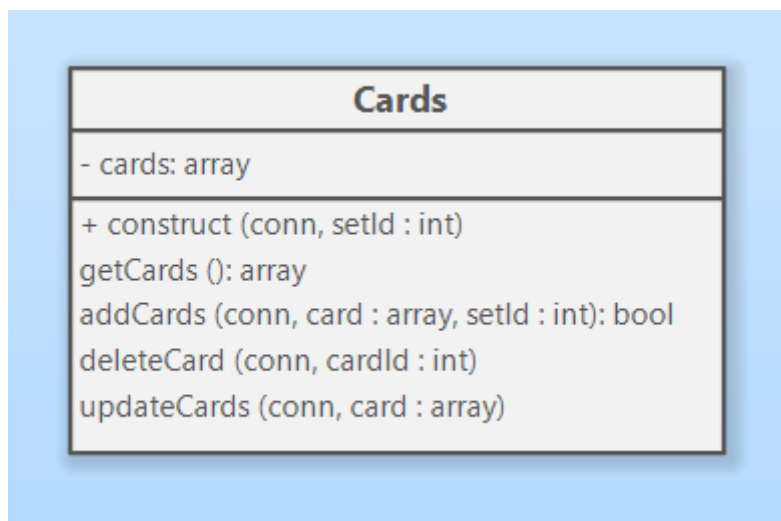
A konstruktor egy sql lekérdezés során a megkísérli lekérdezni a felhasználóhoz tartozó leckék adatait és azokat, amennyibe léteznek, eltárolni.

A `createSet()` metódus a `conn` változón kívül a `userId`, `title` és a `topicId` bemeneti paramétereket veszi át, ezek alapján próbálja létrehozni az új leckét, sikertelen létrehozás esetén hibát dob.

Az `updateSet()` megkeresi a kiválasztott leckét és annak címét, valamint témáját a bemeneti paraméterek alapján módosítja az adatbázisban.

A `searcher()` egy tömböt kap bemeneti paraméterként, melyben a keresendő kifejezés és téma található. Ezek alapján keresést folytat a felhasználó leckéi között és egyezés esetén a talált leckékkel tér vissza.

Cards, azaz kártyák

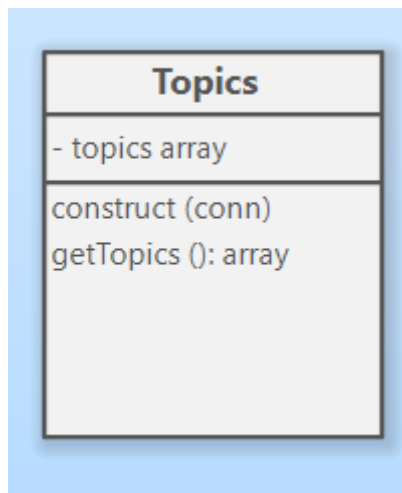


22. ábra Cards osztály

A kártyákat tartalmazó osztály a leckéket tartalmazó osztály mintájára készült. Konstruktórában a `conn` és a `setId` változók kerülnek átadásra, melyek alapján az adott leckéhez tartozó kártyák kerülnek be az osztályban deklarált tömbbe. Metódusai ugyanúgy a kártyákon végezhető műveletekért, lekérdezés, hozzáadás, módosítás, törlés, felelősek.

Topics, azaz témák

Ez az osztály tárolja az előre definiált témákat. Két metódusa van. Egyik a konstruktor, mely az adatbázisból kiválasztja az említett témákat és azokat egy tömbben, az osztály adattagjaként eltárolja. Másik metódusa a `getTopics()` mely az eltárolt témákat tartalmazó tömbbel tér vissza.



23. ábra Topics osztály

View, nézet

A nézeten belül található a `processElements.php` fájl, mely a lekérdezett html elemek (navigációs sáv, regisztrációs űrlap) létrehozásáért felelős. A fájl elején a különböző elemek létrehozásáért felelős függvényeket tartalmazó fájlok kerülnek importálásra, majd a felhasználótól érkező kérés alapján egy switch-case vezérlési szerkezet eldönti, hogy melyik elemet kell létrehozni. A létrehozott elem az `echo` (azaz kiírás) parancs segítségével a `json_encode()` függvényen belül `element` kulcs alatt, egy asszociatív tömbben tárolódik és kerül továbbításra a felhasználó felé. A `json_encode()` és `json_decode()` függvényekkel lehet PHP-ban JSON objektummá alakítani PHP entitásokat és fordítva. Az elemek létrehozásáért felelős függvények több fájlban tárolódnak, attól függően, hogy navigációs sávot, „modális” ablakot vagy kártyákat akarunk megjeleníteni.

Navbars.php

Attól függően, hogy a felhasználó be van-e lépve vagy sem, többféle navigációs sáv is létrehozható. Az elsődleges, melyben vagy a belépésért felelős űrlap vagy a bejelentkezett felhasználó számára, az alkalmazásban való navigálásra szolgáló gombok találhatóak. A

másodlagos pedig, ami a leckék közötti keresésért vagy a kártyákon végzendő műveletéért felelős gombokat, beviteli mezőket tartalmazza.

Cards.php

Ebben a fájlban található az azon függvények, melyek a leckéket és kártyákat megjelenítő Bootstrap kártyák létrehozásáért felelősek. Ezen Bootstrap kártyák az adatok vizuális elkülönítésére szolgálnak.

Modals.php

A fájl tartalmazza a „modális” ablakok létrehozásáért felelős függvényeket. Itt generálódik a regisztrációs, az új kártya hozzáadó, valamint az üzenet és a megerősítés ablak is.

practiceTemplates.php

Ebben a fájlban található a gyakorló felületet generáló függvény.

statistics.php

A statisztikát készítő függvény foglal helyet a fájlban.

profile.php

Ez a fájl tartalmazza a profilt módosító űrlapot elkészítő függvényt.

Kliens szekció

Itt található az index.php fájl, melynek tartalma folyamatosan frissül a felhasználó és az alkalmazás közötti interakciók során. A fájl elején egy switch-case vezérlési szerkezet vizsgálja a session-ben, location kulcs alatt tárolt adatot. ez határozza meg, hogy a felhasználónak melyik oldalt kell megjeleníteni, méghozzá oly módon, hogy az adott oldalhoz tartozó JavaScript fájl kerül az index.php fejlécébe. Alapvető esetben itt az index.js fájl kerül meghívásra, mely a be nem jelentkezett felhasználó kezeléséért felelős. Továbbá, még ezen fájl előtt, az oldalhoz kapcsolódik a functions.js fájl, melyben az alapvető, minden oldalon szükséges függvények találhatók.

Functions.js

Ebben a fájlban a már említett alapvető metódusok találhatóak, melyek minden oldalon felhasználásra kerülnek. Ezek között található az `addValid()` és `addInvalid()` metódusok, melyek egy kiválasztott html címke hivatkozását kapják bemeneti paraméterként. Feladatuk, hogy az input mezők kinézetét változtassák, pl.: Az `addValid()` kerül meghívásra, amennyiben nem foglalt a regisztrációnál megadott felhasználónév és az `addInvalid()`, amennyiben foglalt.

```
function addInvalid(input) {  
    if ($(input).hasClass("custom-valid")) {  
        $(input).removeClass("custom-valid");  
    }  
    $(input).addClass("custom-invalid");  
}  
  
function addValid(input) {  
    if ($(input).hasClass("custom-invalid")) {  
        $(input).removeClass("custom-invalid");  
    }  
    $(input).addClass("custom-valid");  
}
```

24. ábra `addInvalid` és `addValid` függvények

Az AJAX lekérdezésekre található egy külön eljárás, így nem szükséges minden eseményre teljes aszinkronos lekérdezést írni, elég meghívni az `ajaxCall()` eljárást. A lekérdezés a `process.php` fájl felé irányul. Post metódusú lekérdezés, tehát az adatok rejtve maradnak. Az elküldendő adatok a beépített `JSON.stringify()` függvény segítségével JSON objektummá konvertálódnak. Ezt az objektumot azonosítja a lekérdezés request kulcsa. Az `ajaxCall()` egy tömböt és egy callback függvényt vár bemeneti paraméterként.

```
//Ajax call jQuery  
function ajaxCall(params, callback) {  
    var request = JSON.stringify(params);  
    $.ajax({  
        type: "POST",  
        url: '../server/Control/process.php',  
        data: { request: request },  
        dataType: "json",  
        success: callback  
    });  
}
```

25. ábra `ajaxCall` függvény

A „callback” technika lényege, hogy a függvényeket paraméterként adhatunk át más függvényeknek, így módon befolyásolva, hogy mikor mit hajtson végre a program.

A `toArray()` metódus alakítja tömbbé a felhasználó által megadott adatokat az `ajaxCall()` számára. Bemeneti paraméterként először az adat típusát kell megadni, melyet majd a `process.php` ellenőriz le, pl.: `getElement`, `register`, `login`. A típust követően az elküldésre szánt adatokat kell megadni, melyek egy tömbhöz adódnak és ez a tömb lesz a metódus visszatérési értéke.

```
function toArray(type, content1, content2, content3, content4) {  
    var inputs = [];  
    inputs[0] = type;  
    inputs[1] = content1;  
    if (typeof content2 !== "undefined") { inputs[2] = content2 }  
    if (typeof content3 !== "undefined") { inputs[3] = content3 }  
    if (typeof content4 !== "undefined") { inputs[4] = content4 }  
    return inputs;  
}
```

26. ábra `toArray` függvény

A további, az adatok ellenőrzéséért felelős, függvények is itt találhatók. A `checkChars()` metódus ellenőrzi le, hogy a felhasználó által megadott adat megfelel-e a formai követelményeknek, pl.: email cím esetén tartalmaz @ jelet, a felhasználónév pedig csak betűkből és számokból áll. A `testInput()` metódus az XSS (Cross-Site Scripting) támadások kliens oldali kivédéséért felelős. Megakadályozza, hogy a felhasználó az input címkékbe JavaScript kódot írjon, azáltal, hogy lecseréli a címkéket nyitó és záró szimbólumokat html entitásokra.

```
function checkChars(input, type) {
  const nameexp = /^[a-zA-Z0-9áéíóöüűüÁÉÍÓÖÜÜ]+$/;
  const emailexp = /^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
  const pwdexp = /^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[!@#$%^&*~]).{8,}$/;
  switch (type) {
    case "name":
      return input.match(nameexp);
    case "email":
      return input.match(emailexp);
    case "pwd":
      return input.match(pwdexp);
  }
}
```

27. ábra checkChars függvény

Mivel a navigációhoz szükséges egy univerzális átirányító függvény, így ezt is ebben a fájlban helyeztem el, eseménykezelő formájában. Ellenőrzi a „custom-nav-link” osztály jelölővel ellátott elemekre való kattintást és elindítja az átirányítást. Mivel a kártyák, illetve a gyakorlás esetében felmerülhet mentetlen állás, így erre figyelnie kell a függvénynek. Ez oly módon történik, hogy a böngésző memóriájában egy unsaved változó van eltárolva, amikor mentetlen adatok szerepelnek a felhasználónál. Ezt figyeli az átirányító függvény és mentetlen állás esetén jóváhagyásra váró üzenetet küld a felhasználónak.

index.js

A fájl elején deklarálásra kerül három, logikai értéket tároló globális változó, melyek a regisztráció során megadott felhasználónév, email cím, és jelszó hiba státuszának nyomonkövetéséhez szükségesek. Azért szükséges, hogy globális láthatósággal rendelkezzenek, mert több függvény is kezeli őket és így nem kell minden alkalommal átadni értéként az adott változót. Alapértelmezett értékük a true (igaz), mivel a regisztráció véglegesítésénél ezek kerülnek ellenőrzésre.

Ezután egy navigációs sáv és egy regisztrációs űrlap kerül lekérdezésre. A navigációs sávban található regisztráció gomb felelős az űrlap megjelenítéséért. Az adott beviteli mezőkhöz hozzá van rendelve a blur jQuery esemény, mely az adott címke fókuszvesztését figyeli. Erre csak a felhasználói élmény növelése érdekében van szükség. Amennyiben ezen esemény végbe megy, az adott címke tartalma egy változóba kerül. Ezen a változón hívódik először a testInput, majd,

ha nem üres a beviteli mező, akkor a `checkChars()` metódus. Ha üres a mező vagy a `checkChars()` negatív értékkel tér vissza, akkor az `addInvalid()` metódus kerül meghívásra a bemeneti mezőn, illetve a hozzá tartozó `small` címkén, mely a hibaüzenetet jeleníti meg. Amennyiben a megadott érték megfelel a formai követelménynek, úgy hívódik az `ajaxCall()`, melynek `callback` metódusában kerül kezelésre a szervertől kapott válasz. Ha a válasz `state` kulcsa `error` értékkel rendelkezik, akkor az adat hiba állapotát tároló változó igaz értéket kap és ismételten az `addInvalid()` kerül meghívásra. Ellenkező esetben a hiba változó negatív értéket kap és az `addValid()` lesz meghívva. A regisztrációs űrlapon található regisztráció gomb küldi el a szerverre a rögzítendő adatokat, amennyiben a hiba változók negatív értéket tartalmaznak. Sikeres regisztráció esetén egy modális ablakot kér le a szerverről az alkalmazás, melyben a „Sikeres regisztráció” szöveg szerepel. Ez az ablak automatikusan megjelenik a `modal('show')` jQuery parancsnak köszönhetően. Továbbá a regisztrációs adatokat tartalmazó beviteli mezők értéke alaphelyzetbe áll.

A navigációs sávban található a belépés gomb, amennyiben a felhasználónév és jelszó mezők nem üresek, elküldi a szerverre a belépni kívánó felhasználó adatait. Ha a válasz `state` kulcsa `error` értéket tartalmaz, akkor ismételten egy modális ablak kerül lekérésre a szerverről, mely a belépés során felmerült hibaüzenetet jeleníti meg. Ez az üzenet a válasz `msg` kulcsában tárolódik. Sikeres belépés esetén viszont az oldal újra tölt, ezúttal a bejelentkezett felhasználó számára szükséges elemeket lekérdezve a szerverről.

`sets.js`

A leckék menüpontra kattintva a felhasználónak megjelenik egy második navigációs sáv, melyben egy keresősáv és a hozzátartozó témák szerinti szűrő, valamint egy „Új lecke” feliratú gomb található. A navigációs sáv alatt jelennek meg a leckék, létrehozási sorrend szerinti csökkenő sorrendben. Minden lecke egy Bootstrap kártya sablonban jelenik meg, melynek fejlécében a lecke címe, törzsében a lecke témája, láblécében pedig három gomb található. Balról jobbra haladva ezek a gombok a lecke megnyitása, módosítása és törlése funkciókért felelősek.

Új lecke hozzáadása esetén itt is egy modális űrlap jelenik meg, melyben a felhasználó megadhatja a lecke címét és témáját. A regisztrációhoz hasonlóan, az itt megadott adat, azaz a lecke címe is összevetésre kerül az adatbázissal. Amennyiben a felhasználó már rendelkezik adott című leckével, az alkalmazás hibát dob és nem engedi, hogy létrehozza az új leckét. Jelen esetben azért szükséges az adatbázissal összevetni, mert fennáll a lehetősége, hogy adott felhasználó több eszközön is bejelentkezik, ekkor viszont, ha csak a `session`-ben tárolt

leckékkel vetné össze a program a létrehozni kívánt leckét, akkor fennállna a duplikált lecek létrejöttének veszélye. Amennyiben a lecke létrehozása sikeres volt az oldal újra tölt és a létrehozott lecke megjelenik a többi között.

A keresés gomb indítja el a keresést a szerveren található lecek között. Először megvizsgálja, hogy üres-e a kereső mező, illetve van-e kiválasztott szűrő, majd ezeknek megfelelően küldi el az adatot a szerver felé. Találat esetén csak azon lecek jelennek meg az oldalon, melyek megfelelnek a felhasználó által megadott feltételeknek.

A lecke törlése gomb megnyomásakor egy üzenet ablak kérdezi meg a felhasználót, hogy valóban törölni szeretné-e a leckét. Itt kiemelendő, hogy mivel az adott lecke adatai megjelenítő html elemek és a lecke törlését véglegesítő üzenetablak két külön elemét képezi az oldalnak, így az utóbbi nem tudja automatikusan, hogy melyik leckét választotta ki a felhasználó. Ezért a böngésző lokális tárolójában (localStorage), kerül rögzítésre, amiből a törlés véglegesítése esetén az üzenet ablak törlés gombja által kiváltott esemény elküldi a szervernek, mint a törlésre szánt leckét. Fontos, hogy mivel az adott leckére a kártyákat tároló tábla setId idegenkulcsa hivatkozik, ezért, amennyiben a lecke kártyákat tartalmaz, akkor először azok törölődnek az adatbázisból és csak ezután kerül törlésre maga a lecke. A törlés végbemenetelével az adott lecke tárolója elhalványul, majd törölődik az oldal szerkezeti egységei közül. Ez egy plusz funkció, mely a felhasználói élményt hivatott növelni. A lecke megnyitása gomb location típusú lekérdezést indít az oldal felé, melyben a cards kulcsszó a leckéhez tartozó kártyák lekérdezését és megjelenítését idézi elő. Az oldal automatikusan újra tölt, a lecke kártyáit a cards.js kérdezi le és kezeli.

Lecke módosítása esetén egy, a létrehozó ablakhoz hasonló űrlap jelenik meg, ahol a felhasználó megadhatja az új adatokat. Amennyiben még nincs a megadni kívánt címmel egyező című leckéje a felhasználónak és választott a lehetséges témák közül, úgy az adatok rögzítésre kerülnek a szerveren és a felhasználó oldal tartalma frissül.

A fájlban ezen felül helyet kap egy Sets osztály, melyben a lecek adatai tárolódnak.

cards.js

Az oldal először a szükséges navigációs sávot kérdezi le, mely hasonló funkciót lát el, mint a lecek oldalon megjelenő másodlagos sáv, viszont itt értelemszerűen a kártyákra vonatkozó funkciók elvégzéséért felelős gombok található. Ezek balról jobbra haladva a lecke mentése, gyakorlása, új kártya hozzáadása, valamint a lecke törlése gombok. Ezután lekérdezési kísérlet indul, mely során a process.php lekérdezi a leckéhez tartozó kártyákat.

Amennyiben nincsenek, úgy egy hibaüzenet fogadja a felhasználót, melyben a „A lecke jelenleg nem tartalmaz kártyákat” szöveg jelenik meg.

Ellenkező esetben a leckék megjelenítésért felelős kártyákhoz hasonló szerkezetben jelennek meg a leckéhez tartozó kártyák is. Ezen szerkezeti elemek fejlécében egy törlés gomb található, mely törli az adott kártyát, mind az adatbázisból, mind az oldal szerkezetéből.

A plusz gombra kattintva egy üres lecke sablont kér le a program a szerverről, melyet a kártyákat tartalmazó <div> címke elejére fűz a prepend() függvény segítségével.

A mentés gomb a módosított, illetve hozzáadott leckeiket veti össze az adatbázissal és indítja meg a véglegesítést.

Kártya törlése esetén hívódik a törlésért felelős eseménykezelő. Itt ismételten egy trükkhöz kellett folyamodnom, hogy azonosítani lehessen a törlendő kártyát. Minden kártya azonosítója tartalmazza a kártya adatbázisban található azonosítóját. Ezt a slice() függvény választja le a html címke azonosítójáról. Fejlesztés közben a törlésnél hibába ütköztem. A jQuery \$(this) kiválasztója nem akart működni a kártyákat törölő gombok esetében. Ugyanis külön elemként érzékelt a bennük található ikont és magukat a gombokat. Ezért egy if vezérlési szerkezet dönti el, hogy a felhasználó a gombra vagy az ikonra kattintott-e. Ezt az event.target.tagName változó segítségével teszi. Ha a kapott érték „BUTTON”, akkor az adott címke azonosítóját, ha pedig „I”, akkor az adott címke ős-elemének azonosítóját tárolja el a program. Az azonosító „kinyerése” után hívásra kerül az ajaxCall(), mely elindítja a kártya adatbázisból való törlését. Sikeres törlés esetén hívódnak a fadeOut() és a remove() függvények, animálva a törlési folyamatot.

```

//Kártya törlése
$('body').on("click", "button[id^='delete-card-btn-']", (e) => {
    let cardId = undefined;
    if (e.target.tagName == 'BUTTON') {
        cardId = e.target.id.slice(16);
        ajaxCall(toArray("delete_card", cardId), function (response) {
            console.log(response);
            if (response['state'] === "success") {
                $(e.target).parent().parent().parent().parent().parent()
                    .fadeOut("slow", function () { $(this).remove(); });
            }
        });
    } else if (e.target.tagName == 'I') {
        cardId = $(e.target).parent().attr('id').slice(16);
        ajaxCall(toArray("delete_card", cardId), function (response) {
            console.log(response);
            if (response['state'] === "success") {
                $(e.target).parent().parent().parent().parent().parent()
                    .fadeOut("slow", function () { $(this).remove(); });
            }
        });
    }
});
}

```

28. ábra Kártyát törölő függvény

practice.js

Ebben a fájlban is található egy Cards osztály, mely a kártyák adatait tárolja, illetve a felhasználó oldalon végzendő műveletek metódusait tartalmazza. Az oldal elemeinek betöltése során lekérdezésre kerülnek a szerver \$SESSION['selectedSet'] kulcsa alatt tárolt leckéhez tartozó kártyák adatai.

Az oldal elemeinek betöltése egymásba ágyazott ajaxCall() függvényekkel történik. Erre azért van szükség, mert különálló függvények esetén, tekintve, hogy aszinkronos lekérdezésekről van szó, az oldal szerkezete nem feltétlen lenne az elvárttal megegyező. Az egymásba ágyazásnak köszönhetően viszont úgymond rá vannak kényszerítve a függvények, hogy várják meg az előző lefutását. A legutolsó függvény callback paramétere a Practice() metódus, melyben a gyakorláshoz található eseménykezelők és műveletek hajtódnak végre.

Helyes válasz esetén hívódik a Sets increaseGoodAnswer() metódusa, mely növeli a helyes leckék számát, valamint az adott kártya state változója true értéket vesz fel.

Hibás válasz esetén hívódik a `Sets increaseWrongAnswer()` metódusa, mely növeli a helytelen leckék számát, valamint az adott kártya `state` változója `false` értéket vesz fel.

Statisztika kérése esetén hívódik a `#statistics` azonosítójú gombhoz rendelt eseménykezelő, mely aszinkronos lekérdezést indít a szerver felé, elküldve az eltalált és a nem eltalált szavakat. Ezen szavakból a szerveren egy táblázat generálódik és kerül visszaküldésre a kliens oldal felé. A válaszban kapott statisztika betöltődik az oldalra az említett eseménykezelő segítségével.

[style.css](#)

A fájl tartalmaz minden stílusbeállítást, melyre az alkalmazás felhasználóbaráttá tételéhez szükség van. Osztály jelölők segítségével írtam felül a Bootstrap beállításait, személyre szabva azokat, a háttérszíntől kezdődően, egészen a bemeneti mezők és gombok megjelenéséig.

[Bemeneti mezők](#)

A bemeneti mezők a szokásos, négyszög alakú megjelenés helyett egyetlen sor látszatát keltik, ezt a kiemelés (outline) és szegély (border) beállítások kikapcsolásával, valamint az árnyék (box-shadow) beállításával értem el.

[Gombok](#)

A gombokra több stílust is készítettem, melyek azok funkció szerinti megkülönböztetésére szolgálnak. A Bootstrap alapbeállítása miatt kattintásra a gomb kék árnyékot kap, ami nem illett az alkalmazás stílusához ezért ezt kikapcsoltam, valamint, ha fókuszba kerül egy gomb, (`:focus, :active, :hover`), akkor mérete megnő, köszönhetően a `transform-scale` beállításnak, továbbá színe fehérre változik, jelezve az aktív gombot a felhasználónak. Azért, hogy a változás ne legyen kellemetlen a szemnek, a `transition` beállítást 0.5 másodpercre emeltem.

[Kártyák](#)

A kártyák sarkait 25 pixellel lekerekítettem, a `border-radius` segítségével, valamint a fókuszban lévő kártya mérete is megnő, segítve a felhasználót a tájékozódásban.

[Összegzés és tapasztalatok](#)

A program készítése előtt már rendelkeztem minimális webfejlesztési ismerettel, de az koránt sem lett volna elég egy ilyen volumenű projekt elkészítéséhez. Ebből kifolyólag meg kellett tanulnom új technikákat, trükköket. A jQuery-t, az AJAX-ot és a PDO-t is ennek a

feladatnak köszönhetően sajátítottam el. Továbbá a számtalan kódolással és tervezéssel eltöltött óra alatt jöttem rá, hogy én webfejlesztéssel akarok foglalkozni. Ezért kiemelten hálás vagyok a projektnek.

Említésre méltó hibák és megoldásaik, tesztesetek

A fejlesztés során felfigyeltem rá, hogy bizonyos html elemeket nem talál a kód, holott ezek a böngészőben megjelennek. Kisebb kiatómunka után rájöttem, hogy ez azért van, mert az elemek dinamikusan kerülnek létrehozásra, azaz az oldal letöltése után, így a sima jQuery szelektorral nem lehetett őket azonosítani. Ezért át kell vizsgálni a teljes DOM-ot, hogy tartalmazza-e az adott elemeket. Gombra kattintás esetén például nem az adott azonosítóval rendelkező gombot, hanem a body címkében található gombot választjuk ki. Ez az aprónak tűnő különbség mondja azt a kódnak, hogy nézze át a teljes dokumentumot.

Nagyon tanulságos volt az az eset, mikor a szerver oldalon próbáltam meg egy osztályt tárolni a \$SESSION-ben és a kód folyton hibát eredményezett. A megoldás az volt, hogy a session-t csak az osztályokat tartalmazó fájlok csatolása után indítottam el. Ez azért kritikus fontosságú, mivel a Model OOP szemléletben tárolja az adatbázisból lekérdezett adatokat. Tehát mind a felhasználói adatok, mind a leckék, kártyák, témák attribútumai egy osztály példányban tárolódnak. Ezen osztálypéldányok pedig, mivel több, különböző oldalon is szükségesek, egy a \$_SESSION változóban, adott kulcs érték párjaként tárolódnak. Ha az osztályokat tartalmazó szkriptek a session elindítása után kerülnének meghívásra, akkor a \$_SESSION nem rendelkezne referenciával az osztályok szerkezetére vonatkozóan és hibát eredményezne.

A Bootstrap használatából kifolyólag sok beállítást felül kellett írnom, ám azok nem működtek minden esetben. Ezért szükséges volt az „!important” beállítása használata, mivel az érintett helyzetekre a Bootstrap fejlesztői is ezt használták, ebből kifolyólag csak így tudtam az adott stílusbeállítást felülírni.

Továbbfejlesztési lehetőségek

Mindenképp szeretnék egy adminisztrációs felületet készíteni az alkalmazáshoz és további gyakorlómódokat is tervezek hozzáadni. Esetleg egy, a felhasználók közötti kapcsolatrendszert implementálni, melynek köszönhetően a leckék megoszthatók lennének egymás között.

Felhasznált irodalom

A felhasznált technológiák elsajátításához azok hivatalos dokumentációit tanulmányoztam. Ezen felül nagy segítségemre volt két oktató weboldal, a [w3schools](https://www.w3schools.com/) és a [Mozilla Developer Network](https://developer.mozilla.org/).

JavaScripthez és jQueryhez:

- w3schools JavaScript Tutorial: <https://www.w3schools.com/js/default.asp>
- MDN JavaScript Oktatóanyagok: <https://developer.mozilla.org/hu/docs/Web/JavaScript>
- jQuery dokumentáció: <https://api.jquery.com/>

CSS-hez és Bootstraphez:

- Bootstrap dokumentáció: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>
- w3schools CSS Tutorial: <https://www.w3schools.com/css/default.asp>

Html-hez:

- w3schools HTML Tutorial: <https://www.w3schools.com/html/default.asp>

PHP-hoz:

- PHP dokumentáció: <https://www.php.net/manual/en/>
- w3schools PHP Tutorial: <https://www.w3schools.com/php/default.asp>

SQL-hez:

- w3schools SQL Tutorial: <https://www.w3schools.com/sql/default.asp>