



Határidőnapló webes és androidos platformon

Készítette

Kertész Zoltán

Programtervező informatikus BSc

Témavezető

Dr. Tajti Tibor Gábor

Adjunktus

EGER, 2022

Tartalomjegyzék

Bevezetés	5
1. Az alkalmazás bemutatása	6
1.1. Webes alkalmazás	6
1.2. Androidos alkalmazás	9
2. Fejlesztői környezet	14
2.1. Webszerver	14
2.2. Kódszerkesztő	14
2.3. Android alkalmazáshoz Android Studio	14
2.4. Postman a REST API-hoz	15
3. Az alkalmazás felépítése	16
3.1. Miért PHP és Laravel?	16
3.2. Sass és CSS	16
3.3. Bootstrap	17
3.4. MariaDB, mint adatbázis	17
3.5. Java	17
4. Webes alkalmazás	18
4.1. Migráció	18
4.1.1. Általános információk	18
4.1.2. Migráció a felhasználóhoz	18
4.1.3. Migráció az eseményhez	19
4.1.4. Létrehozott táblák	20
4.1.5. Migrációs hiba lehetőség	20
4.2. Modellek	20
4.2.1. Felhasználói modell	20
4.2.2. Esemény modell	21
4.2.3. Eloquent ORM	21
4.3. Nézetek	21
4.3.1. Felhasználói nézetek	22

4.3.2.	E-mailhez tartozó nézetek	22
4.4.	Kontrollerek	23
4.4.1.	Regisztráció controller	23
4.4.2.	Bejelentkezéshez használt controller	25
4.4.3.	Kijelentkeztetést végző controller	25
4.4.4.	A felhasználóhoz tartozó CRUD controller	26
4.4.5.	Esemény controller és CRUD műveletek	28
4.4.6.	API kontrollerek	30
4.4.7.	CSRF token	31
4.4.8.	Felmerülő hibák	31
4.5.	Levezetés	32
4.6.	API hitelesítéshez Sanctum	32
4.7.	Útvonalak	33
4.7.1.	Webes felülethez tartozó útvonalak	33
4.7.2.	API-hoz tartozó útvonalak	33
4.7.3.	Felmerülő hibák	34
4.8.	Frontendhez használt eszközök	34
4.8.1.	Elképzelés	34
4.8.2.	Bootstrap szerepe a kivitelezésben	35
4.8.3.	CSS helyett SCSS	35
5.	API tesztelése Postmannel	36
5.1.	Kérés írása	36
5.2.	Válasz értelmezés	36
5.3.	Kollekciók a kérésekből	37
5.4.	Teszt írás a kéréshez	37
6.	Androidos alkalmazás	39
6.1.	Miért 21-es API szint?	39
6.2.	AndroidManifest fájl	39
6.3.	Activity indítás	40
6.4.	Activityk meghatározása	40
6.5.	Erőforrások hozzárendelése az alkalmazáshoz	41
6.6.	Modellek	41
6.7.	Külső könyvtárak és Gradle	41
6.8.	API kapcsolat kialakítása	42
6.9.	Végpontok a Retrofithez	42
6.10.	Válaszok kezelése	43
6.11.	Kérések használata	43
6.12.	SharedPreferences az alkalmazásban	43

6.13. Felmerülő hibák	44
7. Továbbfejlesztési lehetőségek	45
7.1. Webes továbbfejlesztési lehetőségek	45
7.2. Mobil alkalmazás bővítési és fejleszthetőségi lehetőségek	45
8. Összegzés	47
9. Köszönetnyilvánítás	48
Irodalomjegyzék	49

Bevezetés

A tanulmányaim alatt lehetőségem volt többféle programozási nyelv megismerésére. A szerver és a kliens oldali nyelvek között is találtam olyat, ami elnyerte a tetszésemet.

A mobil applikáció fejlesztéssel először egy beadandó feladatban találkoztam, majd később az ehhez tartozó órákat is elvégeztem, hogy elmélyítsem a tudásomat. A Java programozási nyelv elsajátítása nem okozott nagy kihívást, köszönhetően a tanáraimnak.

A webprogramozási ismeretek megléte úgy gondolom, hogy manapság alapvető elvárás, az itt töltött éveim során több erre szolgáló programozási nyelvvel is megismerkedtem. De számomra a PHP [14] (PHP:Hypertext Pre-processor) vált a legkedveltebb programozási nyelvvé. A számos PHP alapú keretrendszerek közül számomra a Laravel [16] volt ideális, alkalmas a terv megvalósítására, valamint az eddigi tanulmányok során átfogó tudásra tettem szert vele kapcsolatban.

Ezért az lett a célom, hogy szakdolgozatomban ezeket a technológiákat használjam. Ezek felhasználására rengeteg terv volt a fejemben, mire kialakult a jelenlegi projekt.

Véleményem szerint az emberek manapság inkább különböző applikációkat használnak, hogy a teendőiket nyilvántartsák, mint a hagyományos papír alapú noteszt. Viszont csak a mobilon tárolni a fontos feljegyzéseinket veszélyes, mivel az egy sérülékeny eszköz. A dolgozatom lényege, hogy egy olyan alkalmazást hozzak létre, amiben a felhasználók el tudják tárolni az eseményeiket és el is ériék azokat távolról is, platformfüggetlenül. Mivel az okostelefonok rendelkeznek beépített böngészővel, így elég lenne, a probléma megoldásához csak a webes alkalmazás is, de szívesebben használják a felhasználók a kliens programokat.

1. fejezet

Az alkalmazás bemutatása

A szakdolgozatom projektje két részből tevődik össze. Az első rész egy webes alkalmazás, a második pedig egy mobil applikáció. A fő összetevője a webes felület, mivel Androidon a projekt nem minden funkciója érhető el.

A webes és az androidos alkalmazás tekintetében is törekedtem egy minimalista, mégis modern dizájn kialakítására, ami illeszkedik a világos és sötét témához is.

1.1. Webes alkalmazás

A projekt fő eleme a böngészőből elérhető alkalmazás. Itt került kialakításra a regisztráció, a bejelentkezési felület, és egy rövid ismertető a programról, hogy milyen feladatok, problémák megoldására terveztem. Ezek az oldalak elérhetőek a látogatók számára regisztráció nélkül.

A felhasználónak regisztrációkor meg kell adnia a nevét, e-mail-címét és jelszavát, mint ahogyan az látható is a 1.1-es képen. Ha a regisztráció sikeres, egy levelet küld ki az alkalmazás a megadott címre ahol a regisztrációkor kitöltött név jelenik meg és egy gomb, ami átirányítja a felhasználót a webes felületre. Bejelentkezés után a felhasználó korlátozás nélkül tudja használni az alkalmazás nyújtotta lehetőségeket.

A navigációs részben, a legördülő menüben, az események alatt, lehetőség van új esemény hozzáadására, és a meglévők kilistázására annak megfelelően, hogy melyik csoportba tartozókat szeretnénk megjeleníteni. Ilyen kategóriák az aktív, a teljesített, és a lejárt események.

Ahhoz, hogy a használó új eseményt tudjon hozzáadni, kötelező annak nevet, kezdési és befejezési dátumot adni. Ennek az oldalnak a felépítése a 1.2-es ábrán látható. Mentés közben ezeken az adatokon ellenőrzés fut le, hogy a kitöltés a szabályoknak megfelelő legyen. Ilyen előírás, hogy a név és dátumok ki legyenek töltve. A dátumokra másik szabály is vonatkozik, mégpedig, hogy a kezdési időpont ne lehessen korábban, mint az aktuális dátum, és a befejezési idő nem lehet korábban, mint a kezdési. Ha ezeknek a bevitt értékek megfelelnek, akkor megtörténik az eseménynek a mentése, és

Név*

pl. Minta Elek

Email cím*

Jelszó

Jelszó megerősítés

☐ Elfogadod az oldalra vonatkozó [ASZF](#)-et? *

Regisztráció

1.1. ábra. Regisztrációs form

közben az alkalmazás levélben értesíti a felhasználót, amiben leírja neki az eseményhez tartozó elnevezést, leírást, kezdési és befejezési dátumot.

Új esemény hozzáadása

Esemény neve: pl. Találkozó

Esemény leírása

Esemény kezdete: éééé. hh. nn. --:--

Esemény vége: éééé. hh. nn. --:--

Rögzítés

1.2. ábra. Esemény hozzáadásához tartozó form

Az feljegyzések közötti kilistázás első eleme az „Aktív események”. Az alkalmazásban az számít futónak, ahol a befejezési időpont nem korábbi, mint az éppen aktuális dátum és az adatbázisban a „complete” attribútumban 0 szerepel.

A menüben a következő választható lehetőség a „Lejárt események” listája. Itt van lehetősége a felhasználónak megtekintenie azokat az eseményeket, amiket nem teljesített, de már nem is aktívak. Lejárt eseménynek a rendszerben az számít, ahol a

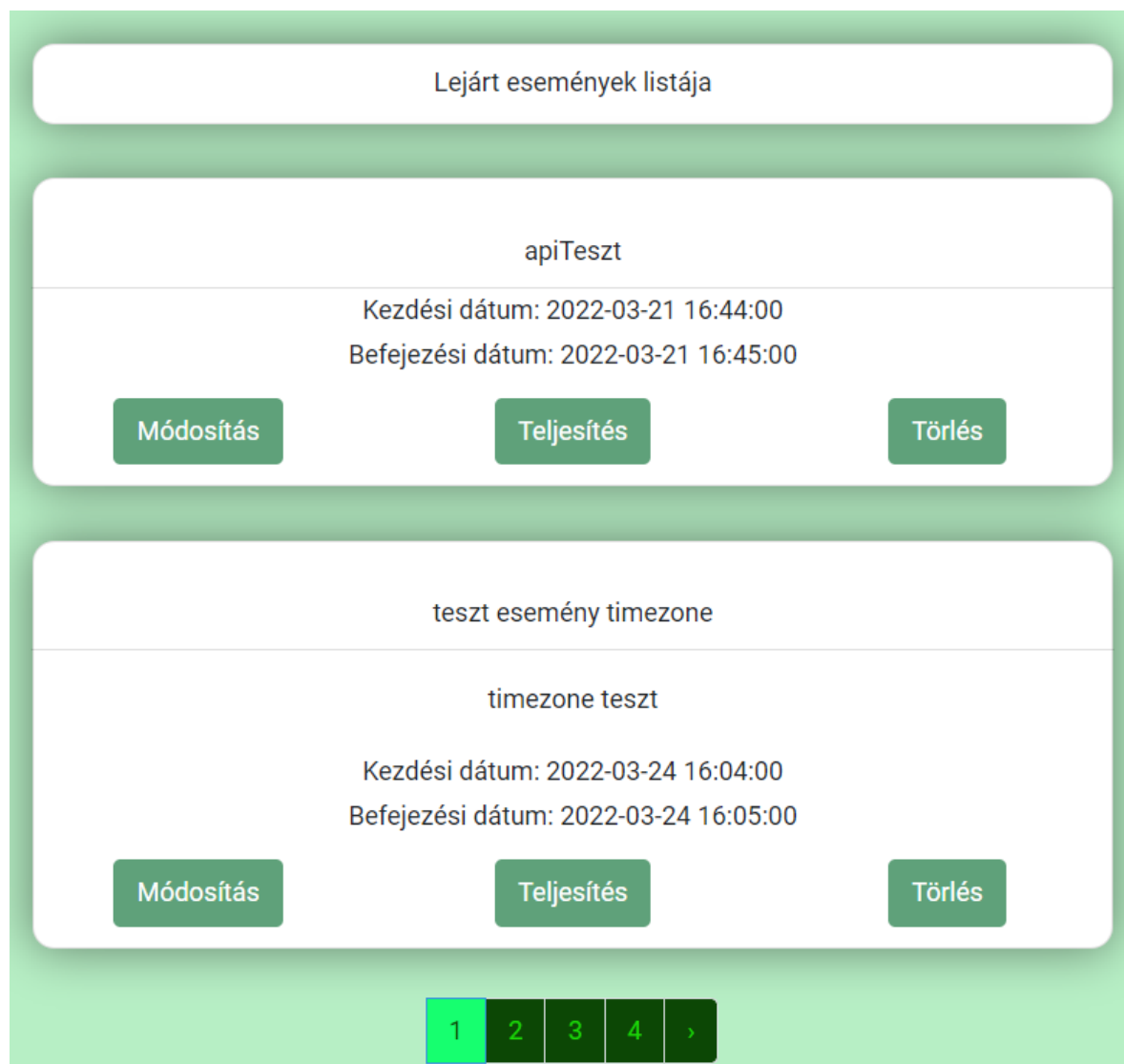
befejezési dátum korábbi, mint az aktuális dátum és a „complete” oszlop értéke szintén 0.

A fenti két eseményben az a közös, hogy a kliensnek lehetősége van a módosításra, teljesítésre és a törlésre is. Módosításkor a hozzáadási szabályok érvényesek az adatokra.

A legördülő menü utolsó eleme pedig a „Teljesített események”. Itt lehet megtalálni azokat az eseményeket, ahol a „complete” oszlopnak 1 az értéke. A megjelenő rekordok a módosítási dátumuk alapján csökkenő sorrendben kerülnek megjelenítésre. A teljesített események fontos különbsége, hogy a módosításra már nincsen lehetősége a felhasználónak, csak a törlésre.

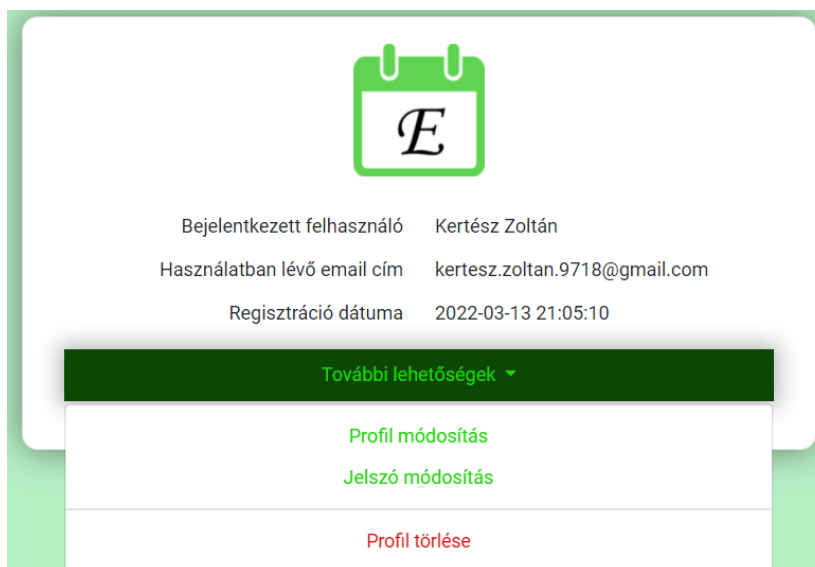
Amennyiben a csoport nem tartalmaz eseményeket a felhasználónak gyors elérést biztosítunk, hogy ne csak a menüből legyenek számára elérhetőek a további esemény csoportok.

Az események megjelenítése és a hozzájuk tartozó funkciók az 1.3-as ábrán láthatóak.



1.3. ábra. Események megjelenítésének stílusa és az azokhoz tartozó funkció gombok

A következő választható menü elem a „Profil”. Itt egy olyan oldal jelenik meg, ahol a bejelentkezett felhasználó adatai kerülnek kiíratásra. Az információk alatt egy legördülő menü található, amiben a profilon elvégezhető műveleteket érhetjük el az 1.4-es ábrán látható módon.



1.4. ábra. Profilhoz tartozó funkciók listája

A „Profil módosítás”-t választva megjelennek a felhasználó adatai. Amennyiben egy adatot nem szeretnénk módosítani úgy azt nem kell átírni. A mentésre kattintva az adatokat ellenőrizzük, ahol a következő szabályoknak meg kell felelni: a név mező nem lehet üres, az e-mail cím nem lehet üres és tartalmaznia kell „@” írásjelet.

Lehetőség van jelszómódosításra is, amit a következő menüpont biztosít. Ezen elem módosításakor egyértelmű, hogy az üres mezők esetén hibával térünk vissza. Ezen kívül az új jelszót kétszer kell megadni és vizsgáljuk, hogy ugyanazok-e a jelszavak. Ezt követően az új jelszónak beírt karakterláncot kódolva eltároljuk, kijelentkeztetésre kerül a felhasználó és át lesz irányítva a bejelentkező oldalra, ahol már az új jelszóval kell bejelentkeznie.

Az utolsó funkció, amit a felhasználó elér ebben a menüben az a profiljának a törlése. Hogy ez sikeres legyen, meg kell adni kétszer a jelenlegi jelszót. Ha a törlés sikeres, a felhasználó az applikáció főoldalára lesz irányítva. Amennyiben sikertelen, egy hibaüzenettel küldjük vissza a törlési oldalra.

A navigációs fül utolsó lehetősége a „Kijelentkezés”. A bejelentkezett felhasználó itt ki tud lépni az alkalmazásból és a kezdőoldalra kerül.

1.2. Androidos alkalmazás

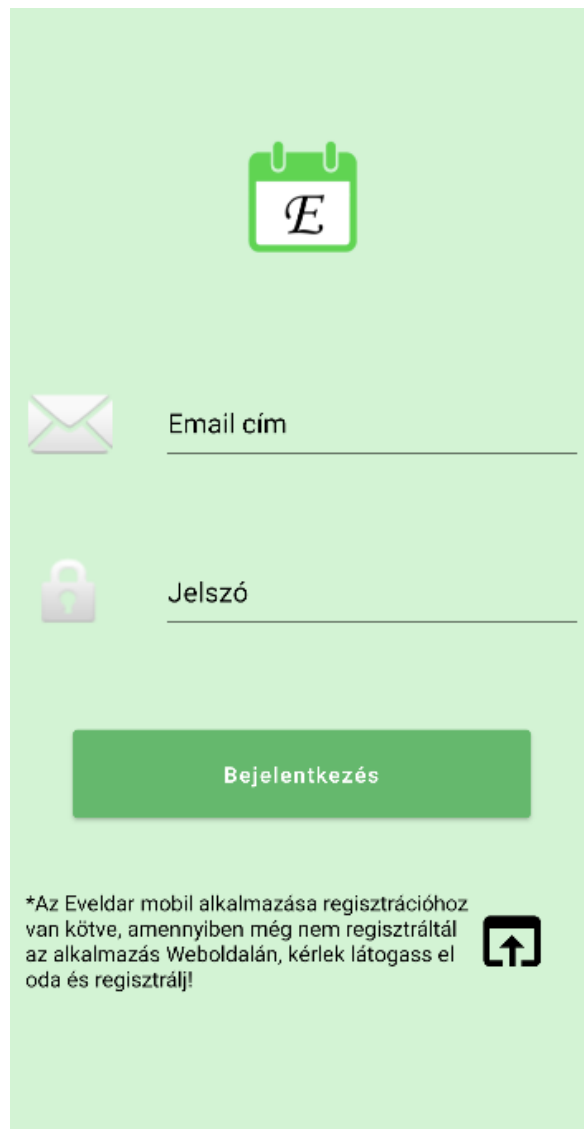
A kliens program Androidos telefonokra készült alkalmazás, ami API [2] kapcsolaton keresztül kommunikál a webes szoftverrel. Az alkalmazás nem tárol lokálisan adatot,

ezzel került megoldásra az a probléma, hogy esetleges meghibásodás esetén adatvesztés történne.

Az alkalmazás első megnyitása után, egy bejelentkezési felület fogadja a felhasználót, ahol meg kell adni az e-mail-címét és jelszavát. Ezen két adat ellenőrzése több rétegben valósul meg. Először az alkalmazás megvizsgálja, hogy nem-e üresek a mezők, amennyiben nem írtunk bele adatot, hibaüzenetet kapunk. Ha ki vannak töltve a mezők, egy REST kérést küldünk. Amennyiben a válasz nem a felhasználó adatai, egy szövegbuborék jelenik meg, hogy hibás a bevitt adat. Máskülönben sikeres a bejelentkezés és a kezdőképernyő jelenik meg.

A még nem regisztrált felhasználók számára egy ikon segítségével gyors elérést biztosítunk az alkalmazásunk főoldalára, hogy a regisztrációt végre tudják hajtani.

Ezen oldal kinézete és elrendezése az 1.5-ös ábrán látható.



1.5. ábra. Androidos alkalmazás belépéshez tartozó felülete

A már bejelentkezett felhasználó a webes felülethez hasonlóan, az aktív események

megjelenítésének oldalára kerül. Ezen kívül található még egy felugró menü, ahol minden funkciót megtalálunk, amit elérhetünk az alkalmazásban. A menüben olyan pontokat találhatunk, mint az esemény hozzáadás, listázás típusonként, feljegyzés kezelése, profil szerkesztése és a kijelentkezés.

Az esemény hozzáadása több helyről is elérhető, mivel ez egy alap funkció, ezért kézközelben kell lennie. Amikor megnyílik a hozzáadásért felelős nézet, amit az 1.6-os ábrán szemléltetünk, látható, hogy a webes felületen megszokott adatok bevitelére van lehetőség. Dátumot egy felugró ablak segítségével lehet beállítani, majd megjelenítjük a kiválasztott időpontot.

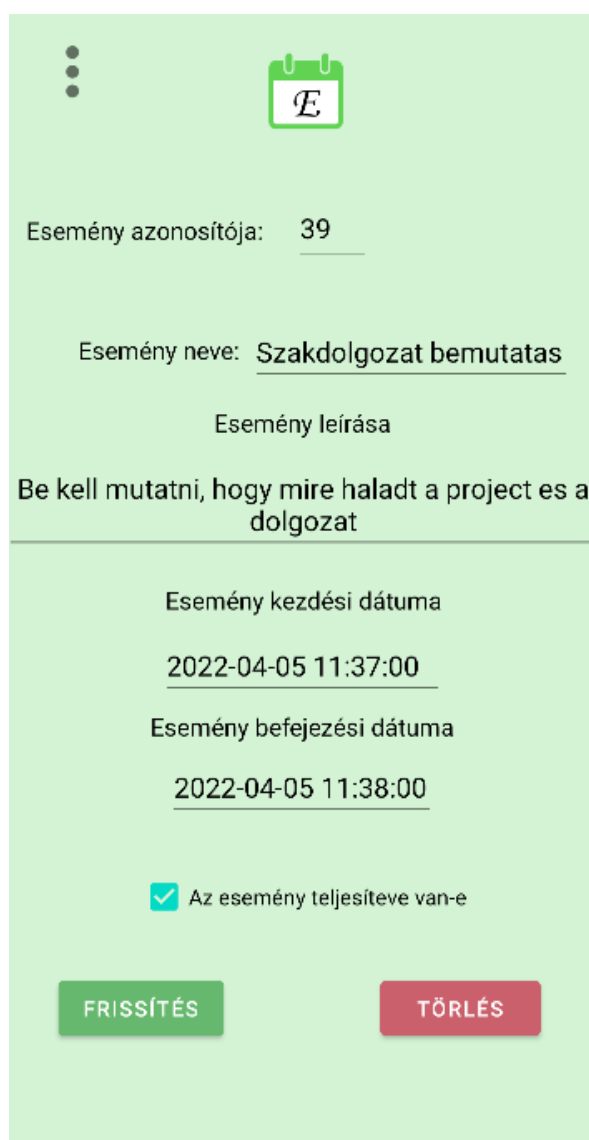


1.6. ábra. Esemény hozzáadása mobilos alkalmazásban

A csoportonként megjelenítendő eseményeknél a felhasználó a következő adatokat látja: az esemény azonosítója, neve, leírása, kezdeti és befejezési dátum. A webes felülethez hasonlóan, itt is kártyaként jelennek meg az adatok. Ugyan úgy, mint a böngészős alkalmazás esetén, ha nem tartalmaz elemeket valamelyik lista, gombokat biztosítunk,

hogy a további esemény csoportok gyorsan elérhetőek legyenek.

Az „Esemény kezelése” menüpontban van lehetőség módosítani, késznek jelölni és törölni az adott eseményt. Először a felhasználónak le kell ellenőriznie, hogy az esemény létezik-e, ehhez meg kell adnia az azonosítóját. Amennyiben nem létezik, egy felugró szövegbuborék jelenik meg, hogy helytelen az azonosító. Ha sikeres volt az ellenőrzés, megjelennek a kiválasztott azonosítóhoz tartozó adatok. Mentés esetén a már webes felületen ismertetett szabályok kerülnek ellenőrzésre. Amennyiben üresen hagyja a felhasználó a nevet vagy a dátumok közül valamelyiket, egy hibaüzenet kerül kiírásra, ahol megjelenítésre kerül, hogy mely adatokat kell leellenőrizni vagy pótolni. Az 1.7-es ábrán látható a lekérdezett esemény azonosítója, ahhoz tartozó értékek, és elvégezhető műveletek.



Esemény azonosítója: 39

Esemény neve: Szakdolgozat bemutatás

Esemény leírása

Be kell mutatni, hogy mire haladt a project es a dolgozat

Esemény kezdési dátuma

2022-04-05 11:37:00

Esemény befejezési dátuma

2022-04-05 11:38:00

☒ Az esemény teljesítve van-e

FRISSÍTÉS TÖRLÉS

1.7. ábra. Kiválasztott azonosítóhoz tartozó adatok és műveletek

A „Profil” menüpontot választva, megjelennek az aktuálisan bejelentkezett felhasználó adatai, amiket módosítani lehet. A sikeres módosítás után az alkalmazás egy

szövegbuborékban tájékoztatja a felhasználót a sikerességről.

Ezen felül lehetőséget biztosít az alkalmazás a felhasználó számára, hogy a jelszavát tudja módosítani. Sikeres jelszó változtatás esetén a rendszer kijelentkezteti a felhasználót. Amennyiben a kérés sikertelen, egy hibaüzenet jelenik meg a felhasználó számára. Ahhoz, hogy a kérés lefusson az alkalmazás megvizsgálja, hogy a felhasználó által megadott jelszó megfelel-e a következő szabályoknak: nem lehet üres egyik mező sem, minimum 6 karakterből kell állnia a jelszónak, nem lehet a jelenlegi és új jelszó ugyan az.

A felhasználói adatokat bejelentkezés után megjegyzi a kliens program, így nem kell minden megnyitáskor belépni csak, ha a menüben kijelentkezünk az alkalmazásból.

2. fejezet

Fejlesztői környezet

2.1. Webszerver

Az alkalmazások működéséhez szükséges egy webszerver. Ehhez több lehetőség is van a fejlesztők részére. Virtualizálhatunk egy Linux alapú szervert, ahol kialakítjuk a LAMP [1] (Linux, Apache, PHP, Myadmin) környezetet. Létrehozhatjuk a programunkat Dockerben [3], ami egy virtuális szervernek felel meg, vagy használhatunk olyan segédprogramot, mint a XAMPP [4] vagy a WAMPP [5].

Személy szerint nekem sokkal jobban bevált a XAMPP, így dolgozatom fejlesztése alatt is azt használtam, mivel egyszerű a telepítése és a kezelhetősége. Mindemellett lehetőség van konfigurálni azt is, hogy milyen adatbázist szeretnénk használni.

2.2. Kódszerkesztő

Ha már van környezetünk ahol majd a webes projektünk futtathatóvá válik, keresnünk kell egy olyan szövegszerkesztőt amit használni fogunk. Programfejlesztésre több ilyen eszköz is létezik, ami segíti a programozó munkáját szövegkiemeléssel, kód kiegészítéssel és szintaxis ellenőrzéssel. Számomra a legjobban bevált ilyen program a „Visual Studio Code” [6], ami ingyenes szövegszerkesztő és különböző bővítményekkel lehet kényelmesebbé tenni a használatát.

2.3. Android alkalmazáshoz Android Studio

Androidos alkalmazás fejlesztésére többféle eszköz áll a rendelkezésünkre. Ilyen lehetőség lehet az „Eclipse” [7] vagy esetleg a „RAD Studio” [8]. Számomra a leginkább preferált fejlesztői környezet az „Android Studio” [9]. Azért tartom jobbnak a többinél, mivel nem csak mobil eszközre lehet vele alkalmazást írni, hanem Wear OS-es órára és TV-re is. A másik előny, hogy a felhasználói kezelőfelület kialakításához biztosít egy

tervezőt, ahol az elemeket fogd és vidd (drag and drop) módszerrel lehet elhelyezni a megjelenítő felületre.

Az „Android Studio”-ban olyan fontos funkciók vannak beépítve, mint az AVD [10] menedzser (manager), ami egy emulátor, ahol virtuális eszközöket tudunk létrehozni, hogy tesztelni tudjuk az alkalmazásunkat. A másik fontos eszköz, ami implementálva van az ADB [11] (Android Debug Bridge), ami a virtuális vagy fizikai eszközünk között kommunikál a számítógéppel így figyeli az alkalmazásunk működését és hiba esetén megkönnyíti a hibakeresést.

2.4. Postman a REST API-hoz

A REST API kapcsolat teszteléséhez szükség volt egy kliens programra, ahol le lehet ellenőrizni, hogy a kérés sikeres-e és a válasz megfelelő-e. Ehhez a „Postman” [12] programot választottam, mivel kliens program, egyszerű a használata, ezért nem csak egy személy vehet részt a tesztelésben. Mindezek mellett, lehetőség van a teszteket kollekcióba elmenteni, amiből későbbiekben automatizált teszt is futtatható. A szoftver nagyon sok API szabványt és formátumot támogat köztük a „JSON”-t [13] is.

3. fejezet

Az alkalmazás felépítése

3.1. Miért PHP és Laravel?

A szerver oldali programozási nyelvek között még mindig vezető helyen szerepel a PHP, köszönhetően annak, hogy széles körben használt, egyszerű és könnyen tanulható nyelv. A másik vezető nyelv a JavaScript [38], amit leginkább „SPA” (Single Page Application / Egyoldalas alkalmazások) fejlesztésére használnak.

Számomra a PHP azért lett kedvesebb, mivel széleskörűen használható, gyors, rugalmas és jól dokumentált.

Mivel tisztán PHP kódot már kevés helyen használnak, leginkább kis projektek esetén, így érdemes keresni egy keretrendszert. Nagyon sok lehetőség van az interneten, amit használhatunk, de talán a legelterjedtebb ilyen a Laravel [39].

A Laravel több szempontból is ideális választás webes alkalmazásokhoz, mivel megkönnyíti az olyan általános feladatokat, mint az útvonalválasztás (routing) és a hitelesítés. A keretrendszer követi az MVC [41] (Model View Controller / Modell Nézet Vezérlő) tervezési mintát, ezért hatékonyabb, biztonságosabb, és a projekt átlátható lesz.

Szinte minden webes alkalmazás kommunikál valamilyen adatbázissal. A Laravel ezt a kommunikációt teszi egyszerűbbé a Query Builder [18] (lekérdezéskészítő) és az ORM (Object Relational Mapping / Objektum-relációs leképezés) használatával.

3.2. Sass és CSS

A CSS [32] állományok gondoskodnak a frontend dizájnjáról. Ez sajnos sok hiba lehetőséggel jár, mivel egyetlen nagy fájlban tároljuk a megjelenésre vonatkozó információkat. Ennek eredménye, hogy nagy projektben strukturálatlanná válik az állomány, aminek rendszerezése igen nagy feladat lenne. Erre jó mód lenne külön állományban tárolni az egyes oldalakra vonatkozó CSS információkat, de ez több kérés lenne a szerver felé.

Ennek a problémának a megoldására léteznek olyan előfeldolgozók (pre-processor), mint a Sass [33], ahol lehetőség van több rész állományt létrehozni, ezzel javítva a struktúrát. Az előfeldolgozó végül pedig egyetlen fájlba egyesíti a részeket és nem növeli a kérések számát.

3.3. Bootstrap

A Bootstrap [34] egy olyan nyílt forráskódú CSS keretrendszer, ami megkönnyíti a frontend programozást az előre definiált komponensekkel. Ez a technológia minimális erőfeszítést igényel a fejlesztő részéről, hogy implementálja saját projektjében. Az előre definiált komponensek tartalmaznak HTML, JavaScript és CSS elemeket, melyeket a későbbiekben könnyű módosítani.

A webes alkalmazásom tervezését ezzel végeztem és a későbbiekben saját ízlésemnek megfelelően írtam felül a szükséges elemeket.

3.4. MariaDB, mint adatbázis

A piacon nagyon sok SQL [40] alapú relációsadatbázis-kezelő szoftver létezik, ezek egyike a MariaDB [35], ami egy nyílt forráskódú, többfelhasználós adatbázisszerver. A legismertebb szerverek közé tartozik a MySQL [36], ami az Oracle tulajdona. A MariaDB és a MySQL között nagyfokú a kompatibilitás, mivel a MariaDB a MySQL-nek a nyílt forráskódú részeiből alakult ki. Másik fontos előny a kompatibilitás tekintetében, hogy egymást ki tudják váltani.

3.5. Java

Manapság az Androidos operációs rendszerre történő fejlesztéskor inkább a Kotlin [37] nyelv van feltörekvőben, mivel a Google ezt támogatja, de nem elhanyagolható a Java programozási nyelv jelenléte sem, hiszen más területeken még mindig fontos szerepe van.

Számomra a Java programozási nyelv a kedveltebb, mivel viszonylag független az operációs rendszertől és olyan könyvtárakat tartalmaz, amik elősegítik a programozó munkáját. Mindezek mellett, a Java idősebb nyelv, mint a Kotlin, így sokkal több ismeretanyag érhető el hozzá.

4. fejezet

Webes alkalmazás

4.1. Migráció

4.1.1. Általános információk

A migráció (migration) [17] segít az adatbázis tábláinak létrehozásában, továbbá, ha az adatbázisban valamilyen módosítást kell végrehajtani, akkor ezen fájlok lefuttatásával újra tudjuk generálni a táblákat. Ahhoz, hogy új migrációt adjunk a projektünkhöz, egy terminálra van szükségünk, ahol a projektünk mappájában tartózkodunk. Ekkor a következő parancsot kell lefuttatni: „php artisan make:migration tabla_neve”.

A migráció létrehozásakor két metódust találunk: „up” és „down”. Az „up” metódus új táblák, oszlopok, indexek hozzáadására szolgál, míg a „down” pedig a módosítások visszaállítására szolgál. Ahogy megfigyelhető a 4.1-es kódokban is, ha a „down” metódus kerül meghívásra, azok a táblákat, ahol a tábla név megegyezik, eldobásra kerülnek.

4.1.2. Migráció a felhasználóhoz

A felhasználóhoz tartozó tábla migrációjában kétféle adat szerepel: felhasználó által kitöltött és generált. A bevitt adatok közé tartozik a név, az e-mail cím, és a jelszó. Az elektronikus levelezési címhez tartozik egy megkötés, ami biztosítja, hogy egyedi (unique) minden felhasználónál.

A generált adatok a következők: id, remember_token, created_at, updated_at. Az emlékezz token (remember token) akkor kerül kitöltésre, ha a felhasználó szeretné, hogy a weboldal megjegyezze a bejelentkezést és bejelentkezve tartsa. Ez a token akkor frissül, mikor a bejelentkezett kliens kijelentkezik. Ez egy biztonsági funkció, arra az esetre, ha eltérítenék a sütit (cookie), akkor egy egyszerű kijelentkezéssel érvénytelenítődjön.

A created_at és updated_at egy dátum típusú mezők, amik szintén legenerálódnak, amikor a felhasználó regisztrál, vagy módosítja az adatait.

Az id attribútum automatikusan növekvő számsorozat amivel egy egyed beazono-

sítható, tehát ez az elsődleges kulcs az adattáblában. A kulcs minden esetben előjel nélküli szám.

4.1.3. Migráció az eseményhez

Az eseményben is szerepelnek generált és felvitt adatok. A generáltról nem beszélnek, mivel nagyban megegyezik a felhasználó migrációban szerepelt generált adatokkal. A különbség az eseményre vonatkozó, felhasználó által beírt adatokban van.

Generálódó adatok közül, amit érdemes megemlíteni, az esemény táblánál a `user_id`, ami egy külső kulcs. Itt az aktuálisan bejelentkezett felhasználó id-ja kerül eltárolásra.

A felhasználónak a következő mezők kitöltésére van lehetősége: `topic`, `description`, `start`, `end` és `complete`.

A `topic` az esemény neve, ahol karakterláncot (string) tárolunk el, amelynek maximális mérete 255 bájt lehet. A következő az esemény leírása. Ez egy text formátumú attribútum, melynek a maximális mérete 65.535 karakter hosszúságú.

Az kezdési és befejezési dátumnál események tárolása révén fontos kritérium volt, hogy nem csak dátumot, hanem időt is tárolni kell. Erre a legjobb megoldás a `datetime` típus, ahol a legnagyobb megadható dátum a „9999-12-31 23:59:59”.

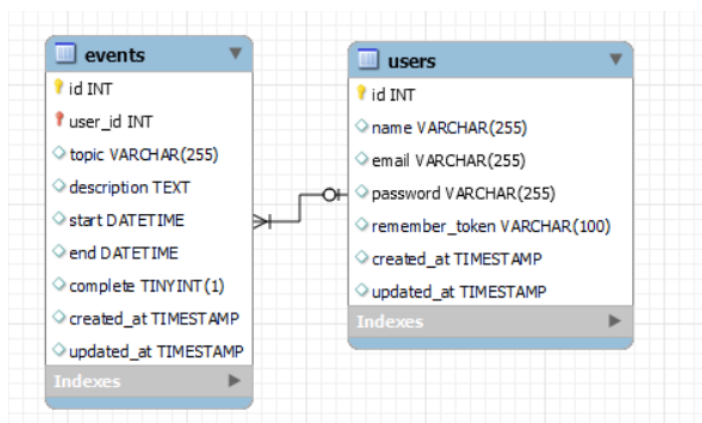
Az eseményhez tartozó migrációt a 4.1-es kódrészlet tartalmazza, ahol megfigyelhető a kitöltési szintaxis is.

```
1 public function up(){
2     Schema::create('events', function (Blueprint
3         $table) {
4         $table->id();
5         $table->foreignId('user_id')->constrained()
6             ->onDelete('cascade');
7         $table->string('topic');
8         $table->text('description')->nullable();
9         $table->dateTime('start');
10        $table->dateTime('end');
11        $table->boolean('complete')->default('0');
12        $table->timestamps();
13    });
14 }
15 public function down(){
16     Schema::dropIfExists('events');
17 }
```

4.1. kód. Eseményhez tartozó migrációs kód

4.1.4. Létrehozott táblák

A migráció lefuttatásával a definiált táblák létrejönnek az adatbázisban. A Laravel keretrendszerben vannak olyan előre definiált táblák, amelyek a projekt legenerálásakor létrejönnek. Ilyen tábla például a `personal_access_token` nevezetű ahol a felhasználói API tokenek találhatók. Ez egy különálló tábla, ahol eltárolásra kerülnek a tokenek. Nincs összekötve a felhasználóval, de id alapján azonosítható. A létrehozott felhasználó és esemény tábla között „egy a többhöz” kapcsolat kerül kialakításra. A 4.1-es ábrán láthatjuk az adatbázishoz tartozó relációs modell kapcsolatát.



4.1. ábra. EER diagram az adatbázis tábláihoz

4.1.5. Migrációs hiba lehetőség

Ahhoz, hogy a migráció sikeresen fusson le, figyelni kell a lefuttatás sorrendjére. Ugyanis, ha előbb kerül létrehozásra az események tábla, mint a felhasználóhoz tartozó, hiba generálódik a külső kulcsra való hivatkozás miatt. Ez a hiba könnyen orvosolható a migrációs fájlok átnevezésével.

4.2. Modellek

A modellek a MVC [43] tervezési mintában az adatbázis és a vezérlő közötti kapcsolat. Itt definiálhatjuk az adatok láthatósági szintjét és, hogy az adott szinten mik érhetőek el.

4.2.1. Felhasználói modell

A modell definiálása a kitölthető adatokkal kezdődik, melyek jelen esetben a név, az e-mail-cím, és a jelszó. A következő attribútumok a rejtettek, amelyek a jelszó és a tokenek. Majd, a végén található egy olyan metódus ami, a kapcsolatot képezi a felhasználó

és az esemény modell között. Itt jelen esetben azt adjuk meg, hogy egy felhasználóhoz több esemény is tartozhat.

A modellhez tartozó implementációt a 4.2-as kódrészlet tartalmazza.

```
1 class User extends Authenticatable{
2     use HasApiTokens, HasFactory, Notifiable;
3     protected $fillable = [
4         'name',
5         'email',
6         'password',
7     ];
8     protected $hidden = [
9         'password',
10        'remember_token',
11        'api_token',
12    ];
13     public function events(){
14         return $this->hasMany(Event::class);
15     }
16 }
```

4.2. kód. Felhasználóhoz tartozó modell

4.2.2. Esemény modell

Az esemény lényegében sokkal egyszerűbb, mint a felhasználó modellje, mivel itt csak kitölthető mezőkkel találkozunk. Ezek a mezők a következők: topic, description, start és end.

4.2.3. Eloquent ORM

Ugyan ez nem modell, hanem az adatbázissal való kommunikációt biztosítja, azért tartom fontosnak megemlíteni a modellek között, mivel ez teszi lehetővé az adatbázisba való rekord beszúrását, lekérdezést és törlést. Amennyiben az adatbázisunkban a tábla neve nem azonos a modellével, akkor meg kell adni, hogy mi a tábla neve. Az eloquent orm [19] és query builder előnyeiről a kontroller osztályban fogok még írni.

4.3. Nézetek

A nézetek (view) felelnek a megjelenítésért. Jelen projektben a nézeteket két csoportba osztanám, melyek a következők: felhasználói nézet, és e-mailhez tartozó nézet. A Laravel a nézetekhez a Blade [21] sablonozó motort használja, ami egyszerű és ugyanúgy lehet benne PHP kódot futtatni. A blade sablonok meghívhatóak útvonalról és vezérlőből is, ennek köszönhetően teljesül a MVC tervezési minta.

A Blade sablonban lévő PHP kódunkat dupla kapcsos zárójelek közé kell tenni. Ilyen PHP kód lehet egy útvonalhívás vagy olyan adat megjelenítés, ami a szerverről érkezik. Ezen kívül a Blade-ben lehetőség van az alapvető vezérlési szerkezetek implementálására, mint például az „if”, „foreach”.

4.3.1. Felhasználói nézetek

Egy felhasználói nézet két komponensből tevődik össze, az egyik az úgynevezett keret, a másik maga a tartalom. A keret tartalmazza a HTML [15] fejléct, ami a következőket foglalja magába: milyen néven található a CSS állomány, az alkalmazás ikonja és a JavaScript meghívása. Ezen kívül itt szerepel még a navigációs menü és egy @yield kulcsszó, ami egy szakasz tartalmának megjelenítésére szolgál. A navigációs menüben találhatunk egy felhasználó ellenőrzést, ugyanis, a látogató annak megfelelően láthatja a menü elemeket, hogy be van-e regisztrálva.

Mielőtt definiálnánk egy szakaszt, meg kell adni, hogy melyik elrendezést örökölje, illetve esetemben, hogy milyen keretet örököljön. Erre a @extends kulcsszóval kell hivatkozni. A szakaszokat @section és @endsection segítségével definiáljuk.

Az egyes eseménycsoportok megjelenítésénél az „if” vezérlési szerkezettel ellenőrzöm le, hogy milyen típusú eseményt kapok vissza. Ennek megfelelően lesznek az események alatt a módosításhoz vagy törléshez szükséges interakció gombok. Ezen kívül az események megjelenítésénél találkozhatunk még „foreach” ciklussal, ahol az egyes elemek és hozzájuk tartozó gombok külön kártyában jelenítődnek meg.

4.3.2. E-mailhez tartozó nézetek

A kiküldött e-mailekhez tartozó nézetek külön vannak definiálva, mivel eltérnek a felhasználóitól, ugyanis, itt a Blade sablonozómotor komponenseit, és a Markdown [30] szintaxisát használjuk. Ennek köszönhetően az e-mailünkben nem csak szöveget tudunk küldeni, de gombot is, mely átirányíthat minket az alkalmazásra, vagy a már meglévő útvonalak valamelyikére.

Az e-mailek-hez tartozó nézet felépítését, az új esemény esetén kiküldött e-mail nézetet szemlélteti, a 4.3-es kódrészlet, mivel itt kezelünk adatot is.

```
1 @component('mail::message')
2 # Új esemény
3
4 A következő eseményt taroltuk el:<br>
5 Az esemény neve: {{$event->topic}}<br>
6 @if ($event->description!='')
7 Az esemény leírása: {{$event->description}}<br>
8 @endif
9 Az esemény kezdeti dátuma: {{$event->start}}<br>
```

```

10 Az esemény befejezési dátuma: {{$event->end}}<br>
11
12 @component('mail::button', ['url' => route('
    modify_event', $event->id)])
13 Tekintsd meg az eseményt!
14 @endcomponent
15
16 Koszonettel,<br>
17 {{ config('app.name') }}
18 @endcomponent

```

4.3. kód. Új eseményhez tartozó e-mail felépítése

4.4. Kontrollerek

A kontrollerek [20] dolgozzák fel a felhasználói műveleteket és válaszol azokra. A kontrollerekben gyűjtjük össze azokat az adatokat, amik a felhasználói felületről érkeznek, és ha kell, meghívjuk a modellt, hogy tárolja el azokat az adatbázisban. Amennyiben az adatbázisból érkeznek az adatok, úgy azt tovább küldjük a megjelenítéshez. Mivel dolgozatomban a felhasználó kétféle szerepkört tölthet be, miszerint lehet látogató vagy bejelentkezett felhasználó, így ezt ellenőrizni kell, hogy a jogosultságának megfelelő oldalakat érje el. Ehhez definiálni kell egy konstruktort, ahol a köztes szoftver (middleware) [22] ellenőrzi, hogy a felhasználó be van-e jelentkezve vagy vendég.

A vezérlők általános működése a következőképpen írható le:

- A felhasználó valamilyen interakciót létesít a kezelő felülettel.
- A vezérlő átveszi a küldött adatokat a felhasználó felülettől.
- Kapcsolat létesítése a modellel, és ha szükséges az adatok módosítása.
- A nézet a válasz alapján létrehozza a megfelelő kezelési felületet. A nézet a modellből nyeri az adatokat, de a modellnek nincs közvetlen tudomása a nézetről.
- A felhasználói felület újabb interakcióra vár és kezdődik előről a kör. [42]

4.4.1. Regisztráció kontroller

A regisztráció csak a látogatók (guest) számára elérhető oldal. Első lépésben ezt ellenőrizni kell, hogy a felhasználó be van-e jelentkezve. Ezt a köztes szoftver (middleware) ellenőrzi. Amennyiben bejelentkezett felhasználó hívná meg a regisztrációért felelős kontrollert, a profilra kerül átirányításra. Ha ténylegesen a vendégtől érkezik a kérés, akkor meghívódik a regisztrációhoz tartozó kontroller.

Amikor meghívódik a következő adatokat vizsgáljuk, amik a felhasználótól érkeznek: név, e-mail cím, jelszó és az „aszf”. Sorban haladva, a névre vonatkozó szabályok, aminek meg kell felelni a bevitt adatnak: kötelező kitölteni, maximum 255 hosszú lehet. A következő az e-mail-cím, ahol a névhez képest annyi a kiegészítés, hogy elektronikus levelezési cím formátumnak kell lennie, tehát „@” jelet kell tartalmaznia. A jelszó is kötelező adat, aminek minimum 6 karakterből kell állnia, és meg kell erősíteni. Az utolsó az általános szerződési feltételekhez tartozó jelölőnégyzet (checkbox), amit be kell pipálni.

Ha a validáció sikeres, a kontroller megvizsgálja, hogy a beírt e-mail-cím már szerepel-e az adatbázisban. Itt elő is jön az eloquent és a query builder előnye, hogy nem kell kézzel lekérdezést írni, hanem egyszerűen meghívjuk a modellt és annak az ahol (where) metódusát. Itt megvizsgáljuk, hogy az e-mail-cím szerepel-e már az adatbázisban. Ehhez a „count” aggregátumot társítjuk, ami megszámolja, hogy a lekérdezés mennyi sort adott vissza. Amennyiben szerepel, tehát a visszatérési érték 1, visszatérünk a nézetre, ahol megjelenítjük, hogy hibás a cím, mert szerepel már az adatbázisban. Amennyiben az eredmény 0, eltároljuk az adatokat úgy, hogy az adatbázisban szereplő attribútumához a kérés tömb megegyező adatát társítjuk. Kivételt képez ez alól a jelszó, ahol előtte meg kell hívnunk a Hash osztály készítő (make) függvényét a kérésben szereplő jelszóra, hogy kódolva tároljuk el azt. Mentés után a felhasználót bejelentkeztetjük, amiben segítségünkre lesz az Auth [23] osztálynak az attempt metódusa, ami megvizsgálja, hogy a kérésben kapott e-mail cím és jelszó páros szerepel-e az adatbázisban. A metódus visszatérési értéke lehet igaz vagy hamis, annak megfelelően, hogy sikerült-e megtalálni a beírt adatokat. Ezt követően egy e-mailt küld ki a rendszer a megadott levelezési címre, amelyben köszönti a frissen regisztrált felhasználót. Végül pedig átirányítjuk a saját profil oldalára.

Ezen folyamat a 4.4-ös kódrészletben tekinthető meg.

```
1 public function store(Request $request){
2     $this->validate($request,[
3         'name'=>'required|max:255',
4         'email'=>'required|email|max:255',
5         'password'=>'required|confirmed|min:6',
6         'aszf'=>'required',
7     ]);
8     $email=$request->email;
9     $emailcheck=User::where('email','=',$email)
10         ->count();
11     $bad_email='';
12     if ($emailcheck>0) {
13         return view('auth.register',[
14             'bad_email' => $emailcheck,
15             'response' => $request,
```



```

15         });
16     }else{
17         User::create([
18             'name' => $request->name,
19             'username' => $request->username,
20             'email' => $request->email,
21             'password' => Hash::make($request->
                password),
22         ]);
23         auth()->attempt($request->only('email',
            'password'));
24         $user=auth()->user();
25         $token=$user->createToken('eveldartoken
            ')->plainTextToken;
26         Mail::to($user)->send(new
            NewRegisteredUser(auth()->user()));
27         return redirect()->route('profile');
28     }
29 }

```

4.4. kód. Regisztrációhoz használt adatok validálása és eltárolás

4.4.2. Bejelentkezéshez használt kontroller

A bejelentkezésért felelős kontroller meghívásakor is vizsgálni kell, hogy már be van-e jelentkezve a hívó fél. Ennek az ellenőrzéséért a kontrollerben szereplő konstruktorban lévő middleware a felelős. Ha egy bejelentkezett felhasználó próbálja elérni a bejelentkezést, a profiljára kerül átirányításra, amennyiben nincs bejelentkezve, úgy eléri a vezérlőt.

A bejelentkezéskor lefut egy ellenőrzés a felhasználó által beírt adatokra, ahol a következő szabályoknak kell megfelelni: az e-mail-cím és jelszó nem lehet üres, és az e-mail-címbe szerepelnie kell a „@” karakternek.

Az Auth osztály itt is nagy segítségünkre van, elvégzi az adatvizsgálatot és bejelentkeztetést. Amennyiben mégsem sikerülne neki, egy hiba üzenettel visszairányítja a felhasználót a bejelentkezési felületre. Ha pedig sikeres, az aktív események oldal jelenik meg.

4.4.3. Kijelentkeztetést végző kontroller

A kijelentkezéshez használt kontrollerben egyetlen egy metódust láthatunk, ami az Auth osztálynak a kijelentkezés (logout) metódusát hívja meg. Ez a metódus, amikor meghívásra kerül, eltávolítja a felhasználót hitelesítő információkat a munkatérből. Amint ez megtörtént, a már vendég felhasználó, átirányításra kerül a kezdőoldalra.

4.4.4. A felhasználóhoz tartozó CRUD kontroller

Ezt a kontrollert már csak az autentikált felhasználók érhetik el. Ennek ellenőrzése a már leírt módon, a kontroller konstruktorában történik. A CRUD műveletek jelentik azt, hogy egy adaton milyen alapvető műveleteket lehet elvégezni. Ezek a műveletek a következők: Create (létrehozás), Read (olvasás), Update (frissítés) és Delete (törlés).

A create műveletet itt nem említeném meg újra, mivel az nem a felhasználói kontrollerben található, hanem egy külön regisztrációért felelősben. Azért került különválasztásra a felhasználótól, mivel regisztrálni csak a vendég felhasználó tud.

Ugyan a kontrollerben nem szerepel a read művelet, azért mivel a bejelentkezett felhasználó adatai gyorsabban elérhetők, ha az Auth kontrollertől kérjük le a nézetben.

Térjünk is rá az Update műveletre, ami a kontrollerünk első fontos eleme. Szeretném megemlíteni, hogy a frissítésből több tartozik a felhasználóhoz. Az első frissítés, amikor a nevét vagy e-mail címét tudja frissíteni. A szabályok egyszerűek és nem térnek el a regisztráció szabályaitól, miszerint nem lehet üres a mező és az e-mailnek megfelelő formátumúnak kell lennie.

Ha ez megvolt, a következő fontos rész, hogy meg kell keresni az adatbázisban azt a rekordot, amit módosítani szeretnénk. Ezt a legegyszerűbben úgy tudjuk megtenni, ha meghívjuk a modell-t és hozzá a find metódust. Itt mutatkozik meg újra az eloquent erőssége, hogy nem kell lekérdezést írni. Ebben az esetben, a lekérdezés egyetlen modell példányt ad vissza. Ezt eltároljuk, hogy a következő lépésben már csak felül kelljen írni az adatokat a felhasználó által kitöltöttel, majd menteni. Ha ez sikeresen megtörtént, visszairányítjuk a felhasználót a profiljára, hogy ellenőrizhesse a megváltoztatott adatokat.

Ezt a folyamatot a 4.5-ös kódrészlet szemlélteti.

```
1 public function update(Request $request){
2     $this-> validate($request,[
3         'name' => "required ",
4         'email' => "required | email",
5     ]);
6     $id=$request->input('id');
7     $profile = User::find($id);
8     $profile->name = $request->input('name');
9     $profile->email = $request->input('email');
10    $profile->save();
11    return redirect()->route('profile');
12 }
```

4.5. kód. Felhasználói adat frissítés

A következő frissítés a jelszóra vonatkozik. A szabály egyszerű és nem tér el a regisztrációs szabályoktól, miszerint ki kell tölteni, minimum 6 karakter hosszúnak kell lennie és meg kell erősíteni az új jelszót. A könnyebb hivatkozás kedvéért, egy-egy vál-

tozoba eltárolásra kerül a jelenlegi és az új jelszó is. Ezután, leellenőrzésre kerül, hogy a jelenlegi és az új jelszó megegyezik-e, mivel ha igen, egy hibaüzenettel térünk vissza a jelszótároláshoz, hogy a két érték ugyanaz. Következő lépésben, a bejelentkezett felhasználó jelszavát kell ellenőrizni, hogy megegyezik-e azzal, amit beírt a felhasználó. Mivel a rendszert használó jelszava kódoltan tárolódik, így a kódolt jelszót kell összehasonlítani a kódolatlanal. Ehhez nagy segítség a Hash osztályban definiált check metódus, ami a kódolatlan értéket hasonlítja össze a kódolt értékkel. Amennyiben az összehasonlítás igaznak bizonyul, tehát megegyezik, lekérjük és eltároljuk az aktuális felhasználó azonosítóját, majd a profilfrissítéshez hasonlóan, kimentjük az adatokat, hogy felül tudjuk írni a jelszót, a felhasználó által megadott, most már kódolt jelszóra. Amennyiben ezek sikeresen megtörténtek, kijelentkeztetjük a felhasználót és átirányítjuk a bejelentkezésre. Itt az átirányításhoz egy üzenetet fűzünk, hogy az új jelszóval kell már bejelentkezni. Ha sikertelen az összehasonlítás, egy hibaüzenettel térünk vissza, amelyben leírjuk, hogy a jelenlegi jelszó nem megfelelő.

Az utolsó művelet amit a felhasználóval lehet végezni, a törlés. Itt fontos kiemelni, hogy a migrationban szereplő kötés miatt, ha a felhasználó törli a profilját, úgy az eseményei is törlésre kerülnek. Ahhoz, hogy a törlés sikeresen lefusson, a felhasználónak meg kell erősítenie azt a jelszavával. Ezért a kontrollerben vizsgálni kell, hogy a jelszó ne lehessen üres és tartozik hozzá egy megerősítés is. Ha ezek a szabályok teljesülnek, akkor összehasonlításra kerül a bejelentkezett felhasználó jelszava és a megadott. Ha nem egyezik meg, akkor visszairányítjuk a törlés oldalra egy hibaüzenettel, amiben az szerepel, hogy nem helyes a jelszó. Ha viszont az ellenőrzés sikeres, megkeressük a felhasználót a bejelentkezett felhasználó id-ja alapján és töröljük. A következő lépés az átirányítás a főoldalra.

A törlésre vonatkozó kódrészlet a 4.6-os.

```
1 public function delete(Request $request) {
2     $this-> validate($request,[
3         'password'=>'required|confirmed',
4     ]);
5     $id=$request->input('id');
6     $password=$request->input('password');
7     $user_password=auth()->user()->password;
8     if (Hash::check($password, $user_password)){
9         $user=User::find($id);
10        $user=User::where('id', '=', $id)
11            ->delete();
12        auth()->logout();
13        return view('about');
14    }else{
15        $error="Hibas jelszot adtal meg!";
16        return view('profile.confirm_delete')->with
            ('error_password',$error);
```

```

17     }
18 }

```

4.6. kód. Felhasználó törlése

4.4.5. Esemény kontroller és CRUD műveletek

Itt is ugyanúgy találkozhatunk egy konstruktorral, amiben vizsgáljuk, hogy a hívó fél be van-e jelentkezve. Amennyiben igen, a szokásos módon itt is eléri a kontroller metódusait, míg, ha nincs bejelentkezve, a bejelentkezési oldalra irányul át.

Betűrendben haladva, kezdjük a sort az eseménylétrehozással. Létrehozáskor a felhasználónak meg kell adni a következő adatokat: név, leírás, start és end. Ezekre ellenőrzés fut le, hogy megfelelők-e a szabályoknak, tehát, hogy a név, a kezdési, és a befejezési dátum nem üres és, hogy az esemény kezdete nem-e korábbi, mint a jelenlegi dátum, illetve a befejezési időpontnak későbbinek kell lennie, mint a kezdési idő. Amennyiben a fenti szabályok valamelyike nem teljesül, úgy a felhasználó felé hibát küldünk, amiben a helytelen adatokat megjelöljük. Ha sikeresen lefut az ellenőrzés, a kérésből létrehozuk az eseményt és az adatokat hozzáadjuk. Sikeres mentés esetén, hogy e-mailt tudjunk küldeni a felhasználónak, le kell kérdeznünk és egy változóba eltárolni az aktuális felhasználót. Ha ez megvan, a rendszerben meg kell keresni azt az eseményt, amit a felhasználó utoljára adott hozzá. Ehhez nagy segítség, megint, hogy nem kell lekérdezéseket kézzel készítenünk, így a where feltételben meg kell adni, hogy melyik felhasználó eseményét keressük, majd, hogy a legutolsót (latest), ezután nincs más dolgunk, mint, hogy az első (first) metódust használva, az első találatot kérjük le és tároljuk el. Az e-mail kiküldésre kerül a bejelentkezett felhasználó által megadott e-mail-címre az esemény adataival, ezután pedig az aktív események listájának nézetére ugrunk.

Ezen folyamathoz tartozó kódrészlet a 4.7-es.

```

1 public function store(Request $request){
2     $this->validate($request, [
3         'start'=> 'required | date | after_or_equal
4             :today',
5         'end'=> 'required | date | after:start',
6         'topic'=> 'required',
7     ]);
8     $request->user()->events()->create([
9         'topic' => $request->topic,
10        'description' => $request->description,
11        'start'=>$request->start,
12        'end'=>$request->end,
13    ]);
14    $user=auth()->user();

```

```

14     $event=Event::where('user_id', '=', auth()->
        user()->id)
15     ->latest('created_at')->first();
16     Mail::to($user)->send(new NewEventAdded($event,
        $user));
17     return redirect()->route('active_events');
18 }

```

4.7. kód. Esemény mentése

A következő, amit az adatokkal lehet csinálni a kiolvasás (read). Ez a projektemben több csoportra osztható, mivel az összes adatot egyben nem lehet kiolvasni. A kiolvasható adatok csoportjai a következők: aktív, lejárt és teljesített események. Nem is írnám le az összes működését, hiszen a különbség csak a szűrési feltételekben tapasztalható. Így csak általánosan írom le, és kitérek a különbségekre. Egy feltétel van, ami minden lekérdezés esetén megegyezik, az pedig a felhasználó vizsgálat, hogy az aktuálisan bejelentkezett felhasználó csak a saját eseményeihez férhessen hozzá. Itt az aktuális felhasználói azonosítót hasonlítjuk össze az adatbázisban tárolt user_id nevű külső kulccsal. Ami még megegyezik minden lekérdezés esetében, a megjelenítésre is vonatkozó paginate metódus, ahol megadjuk, hogy egy oldalon hány rekord jelenhet meg, így a nézeten létre jön egy lapozó az adatok megjelenítéséhez.

Amennyiben az aktív eseményekre vagyunk kíváncsiak, a következő feltételeket kell definiálnunk: a befejezési dátumnak később kell lenni-e, mint az aktuális dátum, és a complete mező értékének 0-nak kell lennie. Ezen feltételeket az ahol (where) metódussal tudjuk meghatározni. A lekérdezett eseményeket pedig befejezési dátum szerint emelkedő sorrendben rendezzük.

A fent leírt folyamat és az aktív esemény szűrési feltételei a 4.8-as kódrészletben láthatóak.

```

1 public function active_events(){
2     $events = Event::where('user_id', auth()->user
        ()->id)
3
4         ->where('end', '>', date('Y-m-d H:i:sa')) )
5         ->where('complete', '=', '0')
6         ->orderByRaw('end ASC')
7         ->paginate(2);
8     return view('events.event',[
9         'events' => $events,
10        'type'=>'active',
11    ]);

```

4.8. kód. Aktív események lekérdezése

Ha a felhasználó a lejárt eseményeket szeretné megjeleníteni, ki kell szűrni azokat a rekordokat, ahol a befejezési dátum hamarabb van, mint az aktuális és a complete mező értéke 0. Az utolsó pedig, hogy az adatokat rendezzük befejezési dátum szerint emelkedő sorrendbe.

A teljesített események kiírása a legegyszerűbb, ugyanis itt pluszban csak a complete mező értékét kell vizsgálni, ugyanis, ha az értéke 1, akkor az esemény teljesítve van. Itt a rendezési feltétel különbözik a többitől, ugyanis módosítási dátum szerint rendezzük csökkenő sorrendbe.

Ezek után, egy nézetet adunk vissza a felhasználónak a lekérdezett adatokkal és egy típus változóval, amiben az szerepel, hogy melyik csoport került lekérdezésre.

A következő műveleti csoport a frissítés (update). A webes felületen nincs lehetőség minden eseménymódosításra, ilyen csoport a késznek jelöltek csoportja. Ahhoz, hogy a frissítést megvalósítsuk két módszerre lesz szükségünk. Az egyik lekéri az adott azonosítónak megfelelő adatot, míg a másik a frissítést végzi el. Nem szeretnék nagyon kitérni, az egy elemet megjelenítőre, mivel csak az esemény modellhez tartozó find módszer található benne, ahol az azonosítóra szűrünk és azt visszaadjuk a nézetnek. A frissítésben található egy kérés ellenőrzés, amire ugyanazok a szabályok vonatkoznak, mint a létrehozásnál. Ezután megkeressük az adott azonosítóhoz tartozó eseményt, amit kimentünk változóba. Így már csak össze kell párosítani a kérés mezőit a kimentett esemény mezőivel és el kell menteni a változásokat a mentés (save) módszerrel. Ezután, meghívásra kerül az aktív eseményekhez tartozó nézet.

A végére maradt a törlés (delete) művelet az alapvető műveletek csoportjából. Törölni minden típusú eseményt lehet, ezért az átirányítás adott nézetre nem megfelelő, mivel nem tároljuk el, hogy honnan érkezik a kérés így a vissza (back) módszert fogjuk visszatérésre használni, ahol az előző oldalra ugrunk vissza. A kérésben itt csak egy azonosítót kapunk, tehát csak meg kell keresnünk a hozzá tartozó rekordot és meghívni hozzá a törlés (delete) módszert. Ezután pedig, a már említett módon, megtörténik a visszatérés a nézetre.

4.4.6. API kontrollerek

Az API kontrollerek felelősek azért, hogy a külső alkalmazás tudjon csatlakozni a projektünkhöz. A jobb kezelhetőség érdekében, külön vezérlőkben található a felhasználóira és az eseményekre vonatkozó módszerek. A válaszok könnyebb értelmezhetősége miatt, azokat JSON formátumban küldjük és fogadjuk. Ahhoz, hogy az eléréseket szűrjük a middlewaret kell használni, de itt nem a vezérlő osztályon belül, hanem az elérési útban vizsgáljuk a felhasználói státuszt.

Felhasználó API kontroller

A lényegi különbség aközött a kontroller között, amit a webes kliens, és amit az API használ nem látható, ugyanis ugyanazokat a validációs folyamatokat végzi el. Amiben mégis eltérnek, azok a válaszok és a szintaxis. A másik különbség, hogy itt nem nézetet adunk vissza, hanem választ (response). A válasz minden esetben egy kulcs-érték páros lista. Az API vezérlőben a felhasználó törlését nem definiáljuk, ezzel csökkentve az illetéktelen használó esetleges károkozását.

Esemény API kontroller

Az esemény kontrollernél ugyanaz a helyzet, mint a felhasználó által használt API kontrollerrel, hogy megegyezik a webes felület vezérlőjével, csak más szintaxist kell alkalmazni. Funkcionalitás tekintetében a webes és az API kontroller megegyezik, ezzel biztosítva a korlátozatlan működést a felhasználó számára.

4.4.7. CSRF token

Bár szorosan nem kapcsolódik a kontroller működéséhez, mégis itt tüntetném fel, mivel, ha a felhasználó nem rendelkezik CSRF tokennel [29], úgy az adatok változtatását sem tudja végrehajtani. Tehát ez is egy biztonsági intézkedés, mint a middleware által a felhasználónak az autentikációja. Ezért minden formot CSRF védelemmel kell ellátni a nézeten, hogy ne történjen illetéktelen átirányítás.

4.4.8. Felmerülő hibák

A kontrollerhez tartozó adat felvitel és menedzselés közben sok hiba történhet. Ilyen hibák lehetnek a tipikus hibák:

- Sikertelen bejelentkezés.
- Már létező felhasználó próbál regisztrálni.
- Hibásan kitöltött adatok.

A hibák kezelésekor minden esetben az aktuálisan futó eljárás felfüggesztésre kerül, és a felhasználót visszairányítjuk a kiindulási oldalra, ahol tájékoztatjuk, hogy mivel volt a hiba.

A másik módja a hibakezelésnek, hogy a felhasználó által kitöltendő adatok mennyiségét csökkentjük. Ezt jelenleg a dátumok kitöltésénél alkalmazom, ahol egy naptárban kell a felhasználónak dátumokat kiválasztania.

4.5. Levelezés

A projektben, véleményem szerint, fontos a levelezés, mivel így a felhasználó értesül a regisztrációról és az új esemény hozzáadásáról. Ehhez nagy segítséget nyújt a Laravelben implementált Mail [28] API. Ahhoz, hogy az alkalmazásunk leveleket küldjön, a .env fájlban be kell állítani a levelező szerverhez tartozó adatokat. Ezután, létre kell hozni egy e-mail osztályt és nevet adni neki, utána következik a formátum megadás, majd ezután oda kell írni, hogy a hozzá tartozó nézet hol jöjjön létre. A létrejövő osztály build metódusában, lehetőség van megadni, hogy milyen tárggyal küldje ki a levelet. A konstruktorban pedig megadásra kerül, hogy milyen modelleket vagy adatokat kívánunk elküldeni.

Ennek megvalósítását a 4.9-os ábrán látható, ahol az új esemény hozzáadásakor küldünk ki levelet a felhasználó számára.

```
1 class NewEventAdded extends Mailable
2 {
3     use Queueable, SerializesModels;
4     public $event;
5     public $user;
6     public function __construct(Event $event, User
7         $user)
8     {
9         $this->event=$event;
10        $this->user=$user;
11    }
12    public function build()
13    {
14        return $this->markdown('mails.events.
15            new_events')
16        ->subject('Új esemény került hozzáadásra!')
17        ;
18    }
19 }
```

4.9. kód. Eseményhez tartozó levélküldő kontroller

4.6. API hitelesítéshez Sanctum

A Sanctum [25] egy hitelesítési rendszer token alapú API-okhoz és mobil alkalmazásokhoz. A Sanctummal egyszerűen adhatók ki tokenek a felhasználó számára. Ezek a kiadott elemek nagyon hosszú lejáratú idejűek, általában több év, de igény szerint módosíthatók és a felhasználó visszavonhatja ezeket. Az API útvonalak eléréséhez a felhasználói státusz ellenőrzését a sanctumba épített köztes szoftver (middleware) végzi.

Először megpróbálja a munkamenet hitelesítő cookit ellenőrizni, amennyiben ez sikertelen, megvizsgálja a kérés fejlécét, hogy az authorizationban található token megfelel-e. A hitelesítéshez Bearer [24] Tokent használunk, ami egy átláthatatlan karakterlánc, ami nem tartalmaz jelentést.

4.7. Útvonalak

A projektünk során fontos megemlíteni az útvonalakat [26], mivel szoros összefüggés van köztük és a kontrollerek között. Majdnem minden útvonalhívás megfelel egy-egy vezérlőben lévő metódushívásnak.

4.7.1. Webes felülethez tartozó útvonalak

A webes felület esetén két fajta kérést különböztetünk meg. Vannak a szerverhez érkező GET és POST kérések. [31] GET kérés esetén adatot kapunk a szervertől, míg POST-kor küldünk a szervernek. A legtöbb kérés esetén vezérlőt hívunk meg, mivel szükséges az autentikáció, amik pedig kivételek, azok esetében egy nézettel térünk vissza. A POST metódusoknál megfigyelhető, hogy el vannak nevezve, mindegyik azért van szükség, hogy a nézetben könnyebb legyen a rájuk való hivatkozás.

4.7.2. API-hoz tartozó útvonalak

A különbség a két útvonal gyűjtemény között, hogy az API útvonalak esetében, a bejelentkezésen kívül, minden más elérési utat a sanctum köztes szoftver segítségével hitelesítünk, tehát csak a hitelesített felhasználók számára elérhető. Éppen ezen különbség miatt, a 4.10-es kódban szeretném bemutatni az itt használt útvonalakat és azok deklarációját.

```
1 Route::post('/login', [UserController::class, 'login']);
2 Route::post('/logout', [UserController::class, 'logout'])->middleware('auth:sanctum');
3 Route::get('/profile', [UserController::class, 'profile'])->middleware('auth:sanctum');
4 Route::post('/checkToken', [UserController::class, 'checkToken'])->middleware('auth:sanctum');
5 Route::get('/active', [EventsController::class, 'active'])->middleware('auth:sanctum');
6 Route::get('/complete', [EventsController::class, 'complete'])->middleware('auth:sanctum');
7 Route::get('/expired', [EventsController::class, 'expire'])->middleware('auth:sanctum');
```

```

8 Route::post('/specified', [EventsController::class,
  'specified'])->middleware('auth:sanctum');
9 Route::post('/new', [EventsController::class, '
  store'])->middleware('auth:sanctum');
10 Route::post('/update', [EventsController::class, '
  update'])->middleware('auth:sanctum');
11 Route::post('/update_profil', [UserController::
  class, 'update'])->middleware('auth:sanctum');
12 Route::post('/delete', [EventsController::class, '
  delete'])->middleware('auth:sanctum');

```

4.10. kód. API-hoz használt útvonalak

4.7.3. Felmerülő hibák

Az útvonal kérések esetén is merülhetnek fel hibák, amiket le kell kezelni úgy, hogy a felhasználó ne érzékelje azokat. Ilyen hibák lehetnek a következők:

- Nem létező útvonal hívása.
- Jogosultságnak nem megfelelő hívás

Amennyiben nem létező útvonalat hív meg a felhasználó, ekkor 404-es hiba oldal jelenik meg a felhasználónak.

Az útvonalak nem mindegyike érhető el a vendégek számára. Amennyiben bejelentkezés nélkül próbálunk meg meghívni egy létező útvonalat, úgy a felhasználó átírányításra kerül a bejelentkezési oldalra és így már meg tudja hívni a kívánt létező útvonalat.

4.8. Frontendhez használt eszközök

4.8.1. Elképzelés

Mikor a projekt webes része elkezdett körvonalazódni a fejemben, egy fontos elhatározásra jutottam. Az alap elképzelésem az lett, hogy egy könnyen kezelhető felületet kell elkészíteni, ahol a fontos funkciók egyben elérhetőek. Színek tekintetében fontosnak találtam, hogy olyanokat használjak, amik kellemesek a szemnek. Az alapértelmezett betűkészletet mindenképpen olyanra akartam cserélni, ami nem túl különleges, de mégsem a megszokott, és illeszkedik az Androidos klienshez is.

Az események kiírásakor ki kellett küszöbölni az esetleges összemosódást. Ehhez a legjobb lehetőségnek az bizonyult, hogy kártyákon jelenjenek meg az adott eseményhez tartozó adatok. Mivel fontos szempont volt, hogy egységes legyen a dizájn, így a kártyás megoldás lett kiválasztva a profil és az alkalmazás leírása oldalakra is.

4.8.2. Bootstrap szerepe a kivitelezésben

A felhasználói felület kialakításában nagy segítséget jelentett a Bootstrap, mivel sok komponens található a dokumentációjában, ahol vizuálisan is megtekinthető a kinézet. Mindemellett nagy előnye, hogy a rendszerbe könnyen implementálható és, ha szükséges, csak a felüldefiniálás igényel munkát.

4.8.3. CSS helyett SCSS

Egyre jobban elterjed manapság a CSS pre-processorok használata a webes fejlesztésekkor. A Sass azért lett kedvelt, mivel hatékony és barátságos a szintaxisa. Ezen pre-processor kétféle fájl kiterjesztést tud kezelni, melyek a .sass és .scss. A kettő közti különbség, hogy míg a .sass a saját szintaxisát követi, addig a .scss kiterjesztésű megőrzi a CSS szintaxist, de többet ad a környezet lehetőségeihez. Mivel a CSS-el kapcsolatban már voltak ismereteim, ezért jobbnak láttam azzal dolgozni.

Mivel az Sass-el megoldódik a CSS állományok rendezésének problémája, így már csak le kell fordítani őket. A Mix [27] lehetővé teszi, hogy a CSS fájlokat összeállítsa.

5. fejezet

API tesztelése Postmannal

Miután megírásra kerültek az API kapcsolathoz szükséges útvonalak és vezérlők, elkezdődhetett a tesztelés. A Postman nagy segítséget nyújt a kérések megírásához, mivel útvonalat lehet benne eltárolni, így nem kell mindig a teljes elérési utat beírni. Mindezek mellett, az API kérésen történt bejelentkezés után, lehetőség van eltárolni a token-t, ezzel segítve a további munkát.

5.1. Kérés írása

A kérések megírásában nagy segítséget nyújt, hogy lehetőségünk van a token és a szerver címének kimentésére, mivel így csak hivatkozni kell az eltárolt címre és az útvonalat megadni. A kérések definiálásakor meg kell adni, hogy milyen típusú a kérés. Ezt a Postmanban egy legördülő menüből lehet kiválasztani.

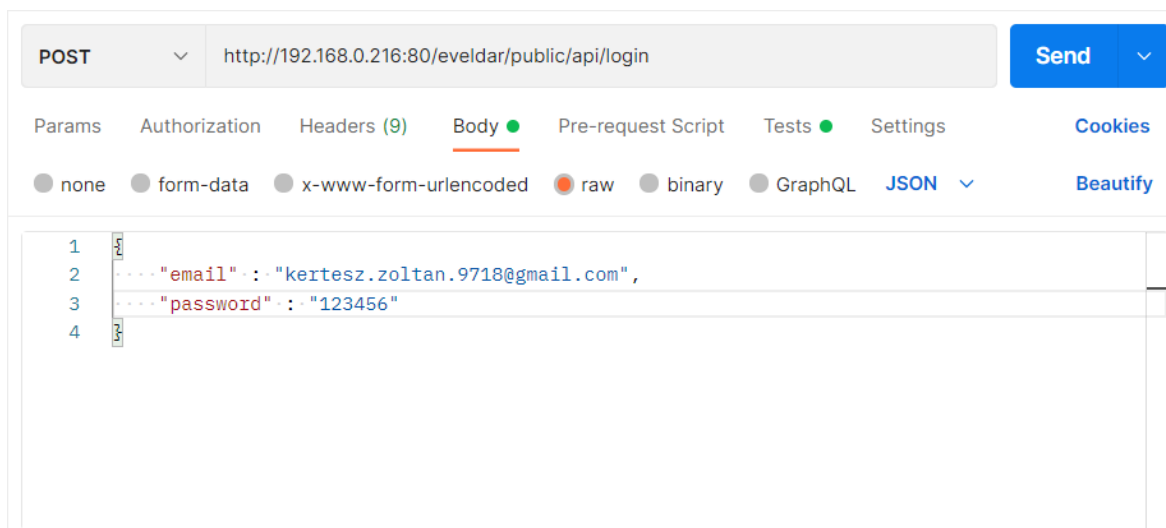
Amennyiben a bejelentkezési token is eltároljuk, és a hívásunkban fel kellene használni, azt könnyen megtehetjük az Authorization fülön úgy, hogy kiválasztjuk a token típusát, ami esetünkben a Bearer token és ahol a token meg kellene adni, ott csak a kimentett adatra kell hivatkozni.

Sok kérés esetében, a fejlécen kívül tartozik adat is hozzá, amit a kérés testében (body) küldünk. Ennek megadására a Postman sok lehetőséget kínál. Számomra a legideálisabb ezek közül a JSON formátum.

A bejelentkezéshez használt kérést és annak formáját a 5.1-es képen szeretném szemléltetni.

5.2. Válasz értelmezés

A kérésre érkező válasz megjelenítésére is több formátum érhető el, hasonlóan a kérés íráshoz. A program lehetőséget kínál a felhasználónak arra is, hogy a választ lementse.



5.1. ábra. Bejelentkezéshez használt POST kérés

5.3. Kollekciónak a kérésekből

Lehetőség van kollekciónak létrehozni, így az egyes kéréseket csoportosítani tudjuk, majd egyben lefuttatni. Erre azért van szükség, hogy lássuk milyen státusz kóddal térnek vissza az esetleges kérések. A kollekciónak elemeit lehetőség van egyesével is futtatni és az egész kollekciónak is megadott iterációnak számban elküldeni a szerver felé. A státusz kód mellett olyan adatokat is megkapunk, hogy mennyi idő alatt kaptunk választ és, hogy mekkora volt az adatforgalom.

5.4. Teszt írás a kéréshez

A Postman lehetőséget biztosít teszt írásra a kéréshez. A fejlesztőnek lehetősége van saját tesztek írására is, de ezek mellett a program, kód töredékeket (snippet) biztosít a hatékony teszteléshez. Az általános tesztek esetén a státusz és a válasz kerül vizsgálatra.

Mivel a tesztek lefutnak amikor elküldésre kerül a kérés, lehetőség van olyan metódusok futtatására ami futtatási környezet változtatására alkalmas. Ilyen változtatás lehet az autentikációs token eltárolása.

A státuszkód vizsgálatra és token eltárolásra az 5.1-es kódrészletben látható példa.

Csoportos kérés futtatása esetén, a tesztek is lefutnak és az eredmények között szerepel, hogy mely tesztek futottak le és milyen eredménnyel.

```
1 pm.test("Successful POST request", function () {
2     pm.expect(pm.response.code).to.be.oneOf([200]);
3 });
4 if(pm.response.to.have.status(200)){
5     var jsonData=JSON.parse(responseBody);
6     pm.environment.set("token", jsonData.token);
```

7 }

5.1. kód. Token mentése és státuszkód vizsgálata

6. fejezet

Androidos alkalmazás

A mobil platformra írt kliensprogram nem pótolja a webes alkalmazást, mivel regisztrálni csak a böngészőn keresztül lehet.

6.1. Miért 21-es API szint?

Ahhoz, hogy androidos eszközre fejlesszünk, meg kell határozni az API [44] szintet. Ehhez figyelembe kell venni, hogy melyik androidos operációs rendszer verzió legyen a legkisebb, amin projektünk futhat. Itt számomra a minél alacsonyabb verzió szám volt az optimális, hogy széles körben használható legyen az alkalmazás. A következő fontos szempont, hogy az Android verzió API szintje támogassa azokat a funkciókat, melyeket meg szeretnék valósítani. Ezen feltételeknek számomra a legjobb a 21-es API szint, ami az Android 5.0-as verziója. Ez a manapság használt eszközök 98%-ával kompatibilis.

6.2. AndroidManifest fájl

Fejlesztés közben a Manifest [47] fájl dinamikusan fog növekedni, ugyanis az alkalmazásról tárolni kell azokat az információkat, amelyeket minden futás előtt tudnia kell a rendszernek az alkalmazásról.

Itt találhatók meg azok az engedélyek, melyek a program futásához szükségesek, valamint az alkalmazáshoz tartozó komponensek.

Lehetőség van még ezeken kívül olyan feltételeket definiálni, amik az alkalmazás egészére vonatkoznak. Ilyen feltétel lehet a projektünk orientációja, tehát a képernyő elforgatása esetén az alkalmazás is elforduljon-e.

6.3. Activity indítás

Az activity indítás történhet adattal és adat nélkül. Dolgozatomban a legtöbb helyen indításakor nem küldünk adatot, mivel azt a szervertől kapjuk. Kivételt képez ez alól az eseményhez tartozó módosítás. Ahhoz, hogy esemény módosítható legyen kattintásra, muszáj elküldenünk a hozzá tartozó azonosítót, hogy megkapjunk minden attribútumát.

Az intenteket [48] lehet implicit és explicit módon használni. Amennyiben explicit módon szeretnénk felhasználni, meg kell határozni az összetevőket, hogy honnan indítjuk és mit hívunk meg. Projektem során ezt részesítem előnyben, mivel minden híváshoz tartozik egy egyértelműen megállapítható osztály.

A fent leírtakra példa a 6.1 kódrészletben látható, ahol az aktív eseményeknek az oldaláról explicit módon definiáljuk az indítani kívánt activity-t, majd hozzá adjuk az adott esemény azonosítóját.

```
1 Intent uniqItem = new Intent(context,
    UpdateEventActivity.class);
2 uniqItem.putExtra("id", sendID);
3 uniqItem.addCategory(Intent.CATEGORY_LAUNCHER);
4 context.startActivity(uniqItem);
```

6.1. kód. Intent indítás explicit módon adattal

6.4. Activityk meghatározása

Az androidos alkalmazás fejlesztésekor a megjelenített felületet Activitynek hívjuk. Minden nézetnek van egy életciklusa, ami a létrehozástól a megszüntetéséig tart.

A nézetek kialakításakor, a színek összeállításában törekedtem arra, hogy a weboldalon használtakhoz hasonló legyen, így egységesítve a két nézetet.

A nézetre különböző elemeket többféle módon tudunk felhelyezni. Lehetőség van a beépített grafikus tervezőt használni, ahol fogd és vidd (drag and drop) módszerrel tudjuk a felületre felhelyezni az elemeket vagy XML kódként is meg tudjuk írni. Amennyiben a kijelző méretének változásakor is dinamikus megjelenítést szeretnénk biztosítani, az elemekre olyan megkötéseket kell definiálni, hogy a szülőhöz képest hol helyezkednek el. A tartalom megjelenítő elemek esetén, azok méretét meg tudjuk határozni úgy, hogy a tartalomhoz igazodjon (wrap content).

Az események kiíratásához egy gyűjtő nézetet kellett biztosítani, hogy az adatok elkülönítve jelenjenek meg. Mivel a megjeleníteni kívánt elemek API kérés alapján kerülnek a felhasználóhoz, így egy dinamikus tartalom megjelenítőre volt szükség. Ehhez a recycler view [49] volt a legjobban megfelelő. Az itt megjelenő adatoknak egy különálló xml-ben lett definiálva a megjelenítőben használt elem. Az egyedi megjelenítés végül

pedig egy adapter segítségével lett hozzáillesztve a viewhoz. Ehhez a megjelenítéshez is a webes felület dizájnja volt segítségül hívva és egy ahhoz hasonló kialakítás volt a cél. Mivel az eszközök között képernyőben, és az alkalmazás által használt kijelző felületben vannak eltérések, így egy kisebb megjelenítőre volt célszerű definiálni a méretet.

6.5. Erőforrások hozzárendelése az alkalmazáshoz

A kliens programhoz többféle erőforrást kell hozzárendelni. A legtöbb ilyen hozzárendelést a képek implementálása jelenti a projektben. A webes felülethez használt alkalmazás ikon az androidos felületen is több helyen megjelenik. Ezen kívül, több kép is lett a projektbe importálva, ezzel is egyedivé téve a programot. Képek tekintetében, a beépített ikonok közül is kerültek felhasználásra, amiket vektor eszközként (vector asset) tudunk a projektünkhöz adni.

Mivel minden activity elrendezés xml fájlban található, így ezek is az erőforrások között tárolódnak el. Ilyen xml fájlok tartoznak a beimportált képekhez is. Ezen kívül, az erőforrások mappában találhatunk egy értékek (values) almappát. Az itt található strings.xml fájlban lehetőségünk van az alkalmazás használata során használt szövegek eltárolására. Amennyiben alkalmazásunkban figyelni szeretnénk a rendszer nyelvét és ahhoz igazítanánk a programunkon megjelenő szövegeket, segítségünkre van a Translations Editor [50].

6.6. Modellek

A mobil alkalmazásban is modelleket használunk az adatok kezeléséhez. Itt a modellek kialakítása az API válasz alapján történt, tehát a válaszban szereplő mezők szerepelnek a modellben. A felhasználók és az események számára is definiálni kellett egyet, mivel ezekhez tartozó adatokat kezelünk. A modellekhez tartozik egy konstruktor, ahol a kapott értékeket állítjuk be. Ezen mezőkhöz tartoznak külön lekérő és beállító metódusok.

6.7. Külső könyvtárak és Gradle

Ahhoz, hogy az alkalmazásunk képes legyen REST API kapcsolatra, külső könyvtárat kell használni. Ilyen könyvtár a Retrofit [45]. Ahhoz, hogy alkalmazásunk kommunikációja gördülékenyen működjön, még szükséges egy konvertert alkalmazni a kérések értelmezéséhez, ehhez a Gson konverter könyvtár a legegyszerűbb.

Az adatok megjelenítése szempontjából is szükséges egy külső könyvtár használata, ami implementálja a recycler view nézetet.

A projektünkben használt könyvtárakat a gradle fájlban kell megadni. Miután ezt az állományt módosítottuk, minden esetben le kell szinkronizálni. Ez a fájl tartalmaz az alkalmazásunkról információkat. Ilyenek azok, hogy melyik a minimum használható API szint, és mi a cél szint, ezen kívül, hogy az alkalmazásunk milyen verzió számmal rendelkezik. Illetve fordításra vonatkozó információk is találhatóak, hogy az alkalmazást a Java mely verziójára szeretnénk fordítani. A gradle segít összeállítani az alkalmazásunkat.

6.8. API kapcsolat kialakítása

Ahhoz, hogy a kéréseket tudjuk kezelni androidos alkalmazás esetén egy segéd könyvtárra van szükségünk, ami egyszerűvé teszi a JSON adatok letöltését. Ehhez a legjobb választás számomra a Retrofit.

Ahhoz, hogy a projektünk képes legyen a Retrofit kapcsolat kezelésére, a manifest fájlban engedélyt kell adni az internet használatához. Ezután, létre kell hozni egy Retrofit klienst, ahol felépítjük a kapcsolatot és megadjuk az alap elérhetőség címét. A végpontok definiálásakor ez a cím kerül kiegészítésre.

6.9. Végpontok a Retrofithez

Ha létrehoztuk a Retrofithez tartozó klienst, definiálni kell a végpontokat. A végponthoz tartozik egy paraméterezett Call objektum.

A végpontoknál definiálni kell a kérés típusát, ami lehet POST vagy GET. Az autentikáció miatt, ki kell egészíteni a kérés fejlécét egy Header annotációval aminek a kulcsa az „Authorization”, ahol egy karakterláncot kell megadni. Ha űrlap adatokat küldünk, szükség van a FormUrlEncoded megjegyzésre a HTTP metódus megadása előtt. A küldött adatokat a Filed annotációban tároljuk el, ahol először megadjuk a kulcsot, majd a hozzá tartozó érték típusát és nevét.

A 6.2-os ábrán látható példa a POST és GET típusú végpontok definiálásra.

```
1  @FormUrlEncoded
2  @POST("login")
3  Call<LoginResponse> login(
4      @Field("email") String email,
5      @Field("password") String password
6  );
7  @GET("profile")
8  Call<ProfileResponse> profilData(
9      @Header("Authorization") String authHeader
10 );
```

6.2. kód. POST és GET típusú végpont definiálás

6.10. Válaszok kezelése

Mivel a legtöbb végpont esetén típus specifikus választ várunk, így ezeket osztályokba kellett rendezni. Az osztályokban található lekérő és beállító metódusok, ahol a típushoz felhasználjuk a modelleket.

Ezek az osztályok annak megfelelően lettek kialakítva, hogy a webes applikációban milyen adatot küldünk vissza.

6.11. Kérések használata

A kérések aszinkron módon hívódnak meg az alkalmazásunkban, ami azt jelenti, hogy a kérés egy háttér szálon fut és elemzés után visszaküldi az eredményt a felhasználói felület szálára. Amikor meghívjuk a kéréseket, két metódus tartozik hozzá. Az egyik arra vonatkozik, hogy mi történjen, ha a kérés sikeres volt, a másik metódus, a sikertelen kérés után futtatható kód sorokat tartalmazza, ami legtöbb esetben egy felugró hibaüzenet a kérés sikertelenségéről.

6.12. SharedPreferences az alkalmazásban

Mivel az adatok megjelenítése és kérése valós időben történik, így nem tárolunk semmilyen adatot a lokális eszközön. Ez azért jó, mivel így az esetleges eszköz károsodás esetén, nem történik adatvesztés. Viszont ennek hátránya, hogy minden alkalmazás indításkor, a felhasználónak be kell jelentkeznie, ami rontja a felhasználói élményt. Ezen hiba kiküszöbölésére a SharedPreferences [46] nyújt megoldást, mivel így egy kulcs-érték pár kerül elmentésre egy fájlba, melyet egyszerűen lehet írni és olvasni, tehát a felhasználóhoz tartozó adatokat célszerű ezen a módon elmenteni. Ezen adatokhoz a mentésen kívül tartozik három metódus. Az első megvizsgálja, hogy a felhasználó be van-e jelentkezve. Ezt a metódust minden alkalmazás indításkor meghívjuk, mivel a már bejelentkezett felhasználó kezdő oldala az aktív eseményekhez tartozó oldal, máskülönben a bejelentkezési felület kerül elindításra.

A következő metódusok, a már bejelentkezett felhasználókra vonatkoznak. Az első ilyen, hogy lekérjük a felhasználó adatait. Ezt felhasználjuk a profil oldalon, és minden kérés indításkor, mivel a felhasználó tokenjét szükséges elküldeni a szerver számára, hogy authenticálja a kérést.

Az utolsó metódus felel azért, hogy a felhasználó ki legyen jelentkeztetve az alkalmazásból. Itt nincs más dolgunk, mint a SharedPreferences-hez tartozó editor-ban meghívni a kiürítés metódust és érvényesíteni a módosításokat.

6.13. Felmerülő hibák

A fejlesztés során a webes felülethez hasonlóan több hiba is felmerült. Ilyen hibák a következők:

- Hibás autentikáció.
- Megjelenésre vonatkozó hibák.
- Adatkitöltésben keletkező hiba.
- Hibás kérés és válasz.
- Nem kattintható esemény.

A hibák kezelésekor úgy láttam a legjobbnak, hogy a felhasználót értesítjük és segítjük a hiba megoldásában. A hibák keletkezhetnek az alkalmazásban, amely esetben a kérést el sem küldjük a szerver felé, és keletkezhetnek a szerverről érkező válasz miatt. Ilyen hiba létrejöhet a bejelentkezés folyamán, ahol a felhasználó nem tölt ki valamilyen adatot. Ebben az esetben nem küldjük el a kérést a szerver felé, hanem hibaüzenetben tájékoztatjuk és megjelöljük a felhasználónak, hogy melyik mező értékében szerepel a hiba. Ezt a megoldást több helyen is alkalmazhatjuk, például a profil módosítás esetében, ahol a nevet és e-mail-címet kötelező kitölteni, vagy éppen az új esemény hozzáadásnál, ahol nem lehet üres a név és a dátumok.

A következő hiba, ami előfordulhat, hogy a felhasználó rossz dátum formátumot használna. Ezt kiküszöböltem azzal, hogy egy felugró ablakot biztosítok neki, ahol a kívánt dátumot kiválasztja, és ez kerül eltárolásra.

A felhasználó generálhat olyan kérést, ahol a visszatérés nem elfogadó választ küld. Ilyen kérés lehet az, ha rossz adatot adunk meg bejelentkezéskor. Ebben az esetben, a válasz sikertelen ágra fut, és felugró üzenetben kerül értesítésre a felhasználó.

Ahhoz, hogy az eseményt kattintás hatására lehessen módosítani, az adaptert kell módosítanunk. Az adapter felelős azért, hogy a kérésben szereplő adatok megjelenítsenek a kijelzőn. Ezt az osztályt kellett annak megfelelően kialakítani, hogy egy kattintás figyelő vizsgálja az elemeket, és felhasználói interakció hatására az adott eseményhez tartozó nézet jelenjen meg. Ennek a megvalósításához egy repository nyújtott segítséget. [53]

7. fejezet

Továbbfejlesztési lehetőségek

7.1. Webes továbbfejlesztési lehetőségek

A webes felület még bőven tartogat magában továbbfejlesztési lehetőséget kód és funkcionalitás szinten is.

Amit kód szinten tovább lehet fejleszteni, hogy a HTML elemekből komponenseket lehetne létrehozni, így könnyebb lenne egységesíteni a dizájnt is. Amit még ezen felül érdemes lenne megcsinálni, az a kontroller osztályok újra definiálása az egyetlen felelősség elve alapján, tehát külön vezérlők létrehozása a következő műveletekhez: létrehozás, adat kiolvasás, frissítés és törlés.

Funkcionalitás szintjén több lehetséges ötletem is van. Az legelső ilyen, hogy a felhasználó ki tudja jelentkeztetni a bejelentkezett eszközöket. A másik továbbfejlesztés, ami funkcionálisan hasznos lenne, egy olyan adminisztrátori felület, ahol lehetőség van a felhasználók és azok adatainak kezelésére. Ezen felül még a felhasználóknak lehetne egy fiók szüneteltetési funkciót készíteni, hogy az adatai ne törlődjenek, amennyiben már nem kívánja használni a rendszerünket. Mindezek mellett, az e-mail funkciót lehetne még bővíteni, hogy a felhasználónak az esemény módosításakor is küldjön értesítést a rendszer, és a lejáratási idő előtt is, egy megadott idővel.

Megjelenítés szempontjából a dizájnnon még lehetne javítani, esetleg minimális animációt a felülethez rendelni, ami még nem rontja a felhasználói élményt, de különlegessé teszi az oldalunkat. Mivel események tárolására szolgál a rendszer, lehetne az oldalhoz egyedi kurzort is rendelni, ami még vonzóbbá teszi a felhasználó számára az oldalunkat.

7.2. Mobil alkalmazás bővítési és fejleszthetőségi lehetőségek

Lehetőség még itt is van bőven a továbbfejlesztésre. Többek között a nézetek átdolgozása fragmentekre [52], hogy ne mindig activity-t kelljen indítani. Külső megjelenésre

jobban össze lehetne igazítani a kinézetet annak megfelelően, hogy az adott telefon sötét módban van-e vagy simán használják. Mivel az alkalmazásunk jelenleg nem támogatja az elforgatott képernyőt, így ahhoz is el lehetne készíteni a nézetet. További lehetőség, ami a nézethez kapcsolódna, hogy amikor a felhasználó egy adott eseményt szeretne módosítani, azt jelenleg úgy tudja megtenni, hogy adott azonosítóra kell szűrnie. Ezt meg lehetne úgy oldani, hogy a kilistázott eseményeknél kattinthatóvá lehetne tenni az adott elemet, és így meg lehessen nyitni a módosítást.

Ezen felül, a megjelenített szövegeket érdemes lenne kimenteni a szövegekhez tartozó erőforrások közé. Ez azért indokolt, hogy a továbbfejlesztéskor az alkalmazásunk már ne csak magyar nyelven legyen elérhető.

Amit még érdemes lenne fejleszteni, hogy a felhasználó által lekért adatokat a készüléken eltárolni egy noSQL adatbázisban, vagy Room-ban [51], így nem kell mindig kéressel fordulni a szerver felé, ha az eseményeket szeretnénk látni, és internet elérés nélkül is használható lenne a szoftverünk.

Funkciók tekintetében, amivel érdemes lenne kibővíteni még a szoftvert, hogy ne csak egyesével lehessen törölni az eseményeket. Ehhez nézet tekintetében megfelelő lenne egy olyan megoldás, ahol a törölni kívánt eseményeket be kellene pipálni és így futna le a törlés.

Mivel eseményeket kezelünk, így hasznos lenne az alkalmazáshoz több widget-et készíteni, ami az új esemény felvételét megkönnyítené a felhasználó számára. Mindezek mellett, gyors elérhetőséget lehetne biztosítani az alkalmazás használójának, hogy az esemény csoportokat közvetlenül a képernyőjén láthassa.

8. fejezet

Összegzés

Dolgozatom zárásaként szeretném összefoglalni a munkát, a vele járó tapasztalatot, és mindezek után még egyetemi éveimet is.

A webes applikáció elkészítéséhez előzetesen meg kellett tervezni az adatbázist, ami ugyan elég egyszerű lett, de van lehetőség a továbbfejlesztésre. Szükséges volt meghatározni, hogy milyen funkciókat kell ellátni a projektemnek és ehhez tartva magam kellett kivitelezni a projektet.

Az androidos alkalmazás elkészítése már nem igényelt annyi előkészületet, mivel funkcionalitás tekintetében a weboldal megadta a hozzá tartozó követelményeket. Az alkalmazás dizájnja tekintetében is támaszkodhattam a már kész webes felületre, de ennek ellenére a legnagyobb kihívást az okozta, hogy egy olyan felhasználói felületet hozzak létre aminek használata egyszerű és egyértelmű. Tudásomat gyarapítottam az alkalmazás létrehozása közben, mivel külső könyvtárat kellett használni és implementálni a rendszerbe.

Úgy gondolom, hogy nagyon sokat fejlődtem az alkalmazások fejlesztése közben és különböző technológiákkal ismerkedhettem meg, amiket implementálni kellett az alkalmazásban.

Dolgozatom megírása alatt igen erős elhatározás alakult ki bennem, hogy tanulmányaim után web vagy Android fejlesztéssel szeretnék foglalkozni, mivel ezek az alkalmazások egyre nagyobb teret hódítanak.

Egyetemi éveim alatt nagyon sok szép emléket szereztem, annak ellenére is, hogy volt egy-egy vizsga, ami igen nehéz volt, nehézség ellenére ezeket az emlékeket is megízéstitették a szaktársaim és barátaim. Véleményem szerint az egyetemen barátkozni igen erősen ajánlott, mivel sok mindenben lehetnek egymás segítségére.

9. fejezet

Köszönetnyilvánítás

Szeretnék köszönetet mondani egyetemi tanárainknak a fáradhatatlan munkájukért, és az évek alatt átadott tudásukért, és tapasztalataikért.

Külön szeretném kiemelni Dr. Tajti Tibor Gábor tanár urat, aki szakmai tapasztalatával és útmutatásával segítette szakdolgozatom létrejöttét. Rengeteg tudást és tapasztalatot átadott számomra és hallgató társaim számára, arról, hogy milyen csapatban dolgozni és egy projekt életét végig kísérni a tervezéstől a kivitelezésig. Ezen kívül, szeretnék köszönetet mondani Neki, hogy felügyeli, és kezeli a szakmai gyakorlatok végrehajtását.

Köszönetet szeretnék mondani Balla Tamásnak, aki a webes technológiákkal kapcsolatos órákkal felkeltette érdeklődésem és segítette fejlődésem a web programozásban.

Szeretnék köszönetet mondani Dr. Kuser Gábornak, aki egy beadandó feladattal bevezetett az Android programozás világába.

Meg szeretném köszönni Fazekas Csaba munkáját, aki a Mobilprogramozás című tárgyakat tartotta, és segített elmélyíteni tudásomat ezen a területen.

Köszönetet szeretnék mondani szaktársaimnak, barátaimnak, akikkel együtt haladtunk végig ezen az úton és támogattuk, segítettük egymást mindenben. Bízom benne, hogy az egyetemen szerzett barátokkal megmarad a kapcsolatunk.

Végül szeretnék köszönetet mondani nagyszüleimnek, szüleimnek, nővéremnek, sógoromnak, barátnőmnek és lakótársaimnak. Köszönöm nekik, hogy támogattak és biztattak egyetemi éveim alatt és köszönöm nekik a türelmet és nyugodt környezetet, hogy sikeresen létrejöhessen a szakdolgozatom.

Irodalomjegyzék

- [1] SETTING UP LAMP : GETTING LINUX, APACHE, MYSQL, AND PHP WORKING TOGETHER
Eric Filson, Erick Rosebrock 2006 ,Sybex, ISBN:0782143377
- [2] API-HOZ TARTOZÓ LEÍRÁS:
<https://www.ibm.com/cloud/learn/api>
- [3] A DOCKER HIVATALOS DOKUMENTÁCIÓJA:
<https://docs.docker.com/get-started/overview/>
- [4] INSTALLING, CONFIGURING, AND DEVELOPING WITH XAMPP
Dvorski, Dalibor D Skills Canada, 2007
- [5] A WAMP HIVATALOS DOKUMENTÁCIÓJA:
<https://www.wamp.net/docs>
- [6] A VISUAL STUDIO CODE HIVATALOS DOKUMENTÁCIÓJA:
<https://code.visualstudio.com/docs>
- [7] A ECLIPSE HIVATALOS DOKUMENTÁCIÓJA:
<https://help.eclipse.org/latest/index.jsp>
- [8] A RAD STUDIO HIVATALOS DOKUMENTÁCIÓJA:
<https://docwiki.embarcadero.com/RADStudio/Alexandria/en/>
- [9] AZ ANDROID STUDIO HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/studio/intro>
- [10] AZ ANDROID STUDIO AVD MANAGER HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/studio/run/managing-avds>
- [11] AZ ANDROID STUDIO ADB HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/studio/run/managing-avds>
- [12] A POSTMAN ALKALMAZÁS HIVATALOS DOKUMENTÁCIÓJA:
<https://learning.postman.com/docs>

- [13] A JSON HÍVATALOS DOKUMENTÁCIÓJA :
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- [14] A PHP HÍVATALOS DOKUMENTÁCIÓJA :
<https://www.php.net/docs.php>
- [15] A HTML HÍVATALOS DOKUMENTÁCIÓJA :
<https://developer.mozilla.org/en-US/docs/Web/HTML>
- [16] LARAVEL HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/>
- [17] LARAVELHEZ TARTOZÓ MIGRÁCIÓ HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/migrations>
- [18] LARAVELHEZ TARTOZÓ QUERY BUILDER HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/queries>
- [19] LARAVELHEZ TARTOZÓ ELOQUENT HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/eloquent>
- [20] LARAVELHEZ TARTOZÓ KONTROLLER HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/controllers>
- [21] LARAVELHEZ TARTOZÓ BLADE SABLONIZÁLÓ HÍVATALOS DOKUMENTÁCIÓJA :
<https://laravel.com/docs/9.x/blade>
- [22] LARAVELHEZ TARTOZÓ MIDDLEWEAR HÍVATALOS DOKUMENTÁCIÓ OLDALA :
<https://laravel.com/docs/9.x/middleware>
- [23] LARAVELHEZ TARTOZÓ AUTHENTIKÁCIÓ HÍVATALOS DOKUMENTÁCIÓJA :
<https://laravel.com/docs/7.x/authentication>
- [24] BEARER TOKEN DOKUMENTÁCIÓJA :
<https://oauth.net/2/bearer-tokens/>
- [25] LARAVELHEZ TARTOZÓ SANCTUM HÍVATALOS DOKUMENTÁCIÓJA :
<https://laravel.com/docs/9.x/sanctum>
- [26] LARAVEL ROOTINGHOZ TARTOZÓ HÍVATALOS DOKUMENTÁCIÓ :
<https://laravel.com/docs/9.x/routing>
- [27] LARAVELHEZ TARTOZÓ MIX HÍVATALOS DOKUMENTÁCIÓJA :
<https://laravel.com/docs/9.x/mix>

- [28] LARAVELHEZ TARTOZÓ MAIL HIVATALOS DOKUMENTÁCIÓJA:
<https://laravel.com/docs/9.x/mail>
- [29] LARAVELHEZ TARTOZÓ CSRF TOKEN HIVATALOS DOKUMENTÁCIÓJA:
<https://laravel.com/docs/9.x/csrf>
- [30] A MARKDOWN HIVATALOS WEBOLDALA:
<https://www.markdownguide.org/>
- [31] KÜLÖNBSEÉG GET ÉS POST KÉRÉSEK KÖZÖTT:
<https://jkorpela.fi/forms/methods.html>
- [32] CSS-HEZ TARTOZÓ HIVATALOS DOKUMENTÁCIÓ OLDALA:
<https://developer.mozilla.org/en-US/docs/Web/CSS>
- [33] SASS-HEZ TARTOZÓ HIVATALOS DOKUMENTÁCIÓ OLDALA:
<https://sass-lang.com/documentation>
- [34] BOOTSTRAP-HEZ TARTOZÓ HIVATALOS DOKUMENTÁCIÓ OLDALA:
<https://getbootstrap.com/docs/5.1>
- [35] MARIADB-HEZ TARTOZÓ HIVATALOS DOKUMENTÁCIÓ OLDALA:
<https://mariadb.com/kb/en/documentation/>
- [36] LEARNING MYSQL: GET A HANDLE ON YOUR DATA 2ND EDITION
Vinicius M. Grippa, Sergey Kuzmichev, 2021 O'Reilly Media, Inc., ISBN: 1492085928
- [37] PROGRAMMING ANDROID WITH KOTLIN
Pierre-Olivier Laurence, Amanda Hinchman-Dominguez, Mike Dunn, G. Blake Meike, 2021 O'Reilly Media, Inc., ISBN: 9781492063001
- [38] JAVASCRIPT: THE GOOD PARTS
Douglas Crockford, 2008 Yahoo Press, ISBN: 0596517742
- [39] LARAVEL: UP & RUNNING, 2ND EDITION
Matt Stauffer, 2019 O'Reilly Media, Inc., ISBN: 9781492041214
- [40] LEARNING SQL
Alan Beaulieu, 2005 O'Reilly Media, Inc., ISBN: 9780596007270
- [41] WEB-APPLICATION DEVELOPMENT USING THE MODEL/VIEW/CONTROLLER DESIGN PATTERN
Leff, A. and Rayfield, J.T., 2001 DOI:10.1109/EDOC.2001.950428

- [42] PATTERN-ORIENTED SOFTWARE ARCHITECTURE VOLUME 1: A SYSTEM OF PATTERNS VOLUME 1 EDITION
Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, 1996 John Wiley & Sons, ISBN: 0471958697
- [43] A MOZILLA ÁLTAL KÉSZÍTETT LEÍRÁS AZ MVC-HEZ:
<https://developer.mozilla.org/en-US/docs/Glossary/MVC>
- [44] TÁBLÁZAT AZ ANDROID VERZIÓKHOZ ÉS API SZINTHEZ:
<https://apilevels.com>
- [45] A RETROFITHEZ TARTOZÓ DOKUMENTÁCIÓ:
<https://square.github.io/retrofit/>
- [46] A SHARED PREFERENCES HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/training/data-storage/shared-preferences>
- [47] AZ ANDROIDMANIFEST HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/guide/topics/manifest/manifest-intro>
- [48] AZ INTENT HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/reference/android/content/Intent>
- [49] A RECYCLER VIEW HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/reference/kotlin/androidx/recyclerview/widget/Rec>
- [50] A TRANSLATIONS EDITOR HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/studio/write/translations-editor>
- [51] AZ ANDROID ROOM HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/training/data-storage/room>
- [52] AZ ANDROIDHOZ TARTÓZÓ FREGMENTEK HIVATALOS DOKUMENTÁCIÓJA:
<https://developer.android.com/reference/android/app/Fragment>
- [53] AZ ADAPTER OSZTÁLYBAN MEGVALÓSÍTOTT KATTINTÁS FIGYELÉS:
<https://github.com/mitchtabian/Recyclerview>