
**UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE**

**Uniscout: Aplicație mobilă pentru
organizarea Zilelor Porților
Deschise
PROIECT DE DIPLOMĂ**

Coordonator științific:

Conf. dr. Antal Margit

Absolvent:

Gergely Zsolt

2023

UNIVERSITATEA "SAPIENTIA" din CLUJ-NAPOCA

Viza facultății:

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Specializarea: Calculatoare

LUCRARE DE DIPLOMĂ

Coordonator științific:

Conf. dr. Antal Margit

Candidat: Gergely Zsolt

Anul absolvirii: 2023

a) Tema lucrării de licență:

Aplicație mobilă pentru organizarea Zilelor Porților Deschise

b) Problemele principale tratate:

- Studiu bibliografic privind sistemele de REST API și aplicații multiplatforme
- Realizarea unei aplicații pentru organizarea Zilelor Porților Deschise

c) Desene obligatorii:

- Schema bloc a aplicației
- Diagrame de proiectare pentru aplicația software realizată

d) Softuri obligatorii:

- Aplicație mobilă pentru organizarea Zilelor Porților Deschise

e) Bibliografia recomandată:

- [1] Douglas E Comer, „The Internet Book: Everything You Need to Know About Computer Networking and How the Internet Works”, 5-ik kiadás, Chapman and Hall/CRC, 2018.
- [2] Revda Uluşık, „Introduction to Spring Boot”, 2021-06-26. [Online]
<https://medium.com/adessoturkey/introduction-to-spring-boot-458cb814ec14>.
- [3] Stanislav Termosa, „An Introduction to Flutter: The Basics”, 2018-12-07. [Online]
<https://www.freecodecamp.org/news/an-introduction-to-flutter-the-basics-9fe541fd39e2/>.

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: 28.06.2022

Termen de predare: 21.06.2023

Semnătura Director Departament

Semnătura responsabilului
programului de studiu

Semnătura coordonatorului

Semnătura candidatului


Declarație

Subsemnata/ul Gergely Zsolt, absolvent(ă) al/a specializării Calculatoare, promoția 2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu Mureș

Absolvent

Data: 21.06.2023

Semnătura 

Uniscout: Aplicație mobilă pentru organizarea Zilelor Porților Deschise

Extras

Tema lucrării mele este proiectarea și realizarea unui sistem care asistă atât organizatorii cât și participanții zilelor porților deschise din cadrul universității în desfășurarea mai ușoară și transparentă pentru toți a acestei serii de evenimente.

Zilele porților deschise la universitate este o serie de evenimente organizată anual cu scopul de a promova universitatea în rândul liceenilor, și de a oferi celor interesați o imagine asupra funcționării universității. Acesta este un proces absolut esențial, deoarece este singurul prilej cu care cei neinițiați pot să se familiarizeze cu detaliile vieții de zi cu zi a universității înainte de a decide să se înscrie aici. Există numeroase sarcini de îndeplinit în cadrul organizării, cum ar fi gestionarea informațiilor legate de evenimente și participanți, a căror soluție devine lentă și greoaie, determinând nevoia pentru un sistem care să faciliteze întreaga procedură.

Scopul lucrării mele este realizarea unui sistem care oferă o soluție la problemele sus-menționate prin utilizarea avantajelor oferite de două kituri de dezvoltare software moderne. Sistemul pe care l-am dezvoltat facilitează transferul informațiilor legate de evenimente, înregistrarea și memorarea datelor celor prezenți și prepararea unor statistici din setul de date existent.

Sistemul este de fapt alcătuit din 3 părți. Am dezvoltat aplicația mobilă cu ajutorul kitului de dezvoltare software Flutter; aceasta trebuie instalată pe telefoanele utilizatorilor, independent de platformă datorită kitului de dezvoltare. Aici utilizatorii au posibilitatea, în funcție de rolul deținut, să se înregistreze, să creeze evenimente noi, să le modifice pe cele existente, să se înscrie la evenimente și să genereze statistici. În fundal rulează o aplicație Spring Boot, care oferă acces la informațiile stocate în baza de date, procesând, respectiv salvând datele generate de către aplicația mobilă. Iar a treia parte este serviciul de stocare în cloud oferit de Firebase, unde se stochează imaginile, precum și baza de date relațională MySQL, unde se păstrează toate celelalte tipuri de date.

Cuvinte cheie: cross-platform, aplicație mobilă, Flutter, Spring Boot, evenimente.

**SAPIENTIA ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK**

**Uniscout: Mobilalkalmazás nyílt napok
szervezésére**

DIPLOMADOLGOZAT

Témavezető:

Dr. Antal Margit
egyetemi docens

Végzős hallgató:

Gergely Zsolt

2023

Kivonat

A dolgozatom témája egy olyan rendszer megtervezése és kivitelezése, amely egyaránt segít az egyetemi nyílt napok szervezőinek és résztvevőinek abban, hogy az eseménysorozat lebonyolítása könnyebb és átláthatóbb legyen mindenki számára.

Az egyetemi nyílt napok egy évente megrendezett eseménysorozat, melynek célja, hogy népszerűsítse az egyetemet a középiskolás diákok körében, és betekintést engedjen az egyetem működésébe az érdeklődők számára. Ez egy rendkívül fontos folyamat, mert egy kívülálló igazán csak itt pillanthat be mélyebben az egyetem mindennapjaiba mielőtt úgy döntene, hogy ide jelentkezik. A szervezés során több olyan teendő van, mint az eseményekkel és résztvevőkkel kapcsolatos információk menedzselése, amelyek megoldása hagyományos módszerekkel lassú és körülményes, ezért létrejön az igény egy olyan rendszerre, amely mindezt megkönnyíti.

A dolgozatom célja egy olyan rendszer elkészítése, amely az előbb említett problémákra kínál megoldást két modern keretrendszer előnyeit kihasználva. Az rendszerem megkönnyíti az eseményekkel kapcsolatos információk átadását, a jelenlévők adatainak elmentését és statisztikák készítését a meglévő adathalmazból.

Az rendszer tulajdonképpen 3 részből áll. A mobilalkalmazást a Flutter keretrendszer segítségével valósítottam meg, amelyet a felhasználók telefonjaira kell telepíteni és a keretrendszernek köszönhetően platformfüggetlen. Itt a felhasználóknak szerepkörüktől függően lehetőségük van regisztrálni, új eseményeket létrehozni, meglévő eseményeket módosítani, eseményekre jelentkezni, statisztikákat generálni. A háttérben egy Spring Boot alkalmazás fut, amely hozzáférést biztosít az adatbázisban tárolt adatokhoz és feldolgozza, valamint elmenti a mobilalkalmazás által generált adatokat. A harmadik rész pedig a Firebase által biztosított felhő alapú tárhely, ahol a képeket tárolom, valamint a MySQL relációs adatbázis, ahol minden egyéb adatot tárolok.

Kulcsszavak: platformfüggetlen, mobilalkalmazás, Flutter, Spring Boot, esemény.

Abstract

The topic of my thesis is the design and implementation of a system that makes the management of events of university open days easier and more transparent for both the organizers and participants.

University open days are an annual event aimed at promoting the university among high school students and providing insight into its operation for those interested. This is an extremely important process because an outsider can really only get a deeper look into university life here before deciding to apply. There are several tasks involved in organizing an event, such as managing information related to the event and participants, which are slow and cumbersome to solve with traditional methods, hence the need for a system that makes the whole process more effortless and faster.

The aim of my thesis is to create a system that offers solutions to the aforementioned problems, using the advantages of two modern frameworks. My system makes it more accessible to transfer information related to events, save participant related data, and create statistics from the existing dataset for the organizers.

The system consists of three parts. I implemented the mobile application using the Flutter framework, which must be installed on users' phones and is platform-independent. Here, depending on their role, users can register, create new events, modify existing events, sign up for events, and generate statistics. In the background, a Spring Boot application runs, which provides access to the data stored in the database and processes and saves the data generated by the mobile application. The third part is the cloud-based storage provided by Firebase, where I store the images, and the MySQL relational database, where I store all other data.

Keywords: platform-independent, mobile application, Flutter, Spring Boot, event.

Tartalomjegyzék

Jelölések	10
Ábrák jegyzéke	12
Kódrészletek jegyzéke.....	13
1. Bevezető	14
1.1. Témaválasztás indoklása	14
1.2. Elméleti háttér	16
1.3. Technológiai kitekintő	19
1.3.1. Webhely	19
1.3.2. Progresszív webalkalmazás	22
1.3.3. Mobilalkalmazás	24
1.4. Célkitűzések	26
2. Technológiák bemutatása	28
2.1. Szerveralkalmazás	28
2.1.1. MySQL	28
2.1.2. Spring Boot	29
2.1.3. JPA	29
2.1.4. Hibernate	30
2.1.5. API	30
2.1.6. REST	31
2.1.7. RESTful API	31
2.1.8. RESTful API kérések	32
2.2. Kliensalkalmazás	33
2.2.1. Flutter	33
2.2.2. Dart	34
2.2.3. Riverpod	34
3. Rendszerspecifikáció	36
3.1. Felhasználói követelmények (szerver)	36
3.2. Rendszerkövetelmények (szerver)	36
3.3. Felhasználói követelmények (kliens)	37
3.3.1. Fontosabb felhasználói esetek	39
3.4. Rendszerkövetelmények (kliens)	43
4. Tervezés	45
4.1. Adatbázis	45

4.1. Szerver.....	46
4.2. Kliensalkalmazás	48
5. Kivitelezés	50
5.1. Szerveralkalmazás.....	50
5.1.1. Előfeltételek	50
5.1.2. A szerver elindítása	50
5.1.3. Entitás osztályok	52
5.1.4. Vezérlő osztályok.....	54
5.1.5. DTO osztályok.....	55
5.1.6. Szolgáltatás osztályok.....	55
5.1.7. Repository osztályok	56
5.1.8. Biztonság	56
5.2. Kliensalkalmazás	58
5.2.1. Felhasználói felület	58
5.2.2. Vezérlők	58
5.2.3. Repository	59
5.2.4. API hívások.....	59
5.2.5. Az alkalmazás bemutatása.....	61
6. Eszközök	69
6.1. Integrált fejlesztői környezet	69
6.2. Verziókövetés.....	69
6.3. Tesztelés	70
6.3.1. API végpontok tesztelés.....	70
6.3.2. Egységtesztelés	70
6.4. Deploy	71
7. Összefoglaló.....	72
7.1. Következtetések.....	72
7.2. Továbbfejlesztési lehetőségek	72
8. Irodalomjegyzék	74

Jelölések

Rövidítés	Angol név	Magyar fordítás
SPC	Simon Personal Communicator	–
CRUD	Create-Read-Update-Delete	Létrehoz-olvas-frissít-töröl
REST	Representational State Transfer	–
API	Application Programming Interface	Alkalmazásprogramozási felület
WWW	World Wide Web	Világháló
URL	Uniform Resource Locator	Egységes erőforrás helymeghatározó
URI	Uniform Resource Identifier	Egységes erőforrás azonosító
IRI	Internationalized Resource Identifier	Internacionalizált erőforrás azonosító
HTTP	Hypertext Transfer Protocol	Hiperszöveg átviteli protokoll
HTTPS	Hypertext Transfer Protocol, Secure	Hiperszöveg átviteli protokoll, biztonságos
SSL	Secure Socket Layer	Biztonsági réteg
IP	Internet Protocol	Internet protokoll
HTML	Hypertext Markup Language	Hiperszöveges jelölőnyelv
SVG	Scalable Vector Graphics	Skálázható vektorgrafika
PWA	Progressive Web App	Progresszív webalkalmazás
CSS	Cascading Style Sheets	Egymásba ágyazható stíluslapok
QR code	Quick Response Code	Gyors válasz kód
JDK	Java Development Kit	Java fejlesztési környezet

LTS	Long-term Support	Hosszútávú támogatás
WAR	Web Application Resource	Webalkalmazás erőforrás
MVC	Model-View-Controller	Modell-nézet-vezérlés
ORM	Object Relational Mapping	Objektumrelációs leképezés
POJO	Plain Old Java Object	–
XML	Extensible Markup Language	Bővíthető jelölőnyelv
JPA	Java Persistence API	–
REST	Representational State Transfer	–
JSON	JavaScript Object Notation	–
DTO	Data Transfer Object	Adatátviteli objektum
SDK	Software Development Kit	Szoftverfejlesztő készlet
JIT Compilation	Just In Time Compilation	Futásidejű fordítás

Ábrák jegyzéke

1. ábra: Az okostelefon felhasználók száma évekre lebontva	16
2. ábra: Esemény készítése Bitrix24-el	18
3. ábra: Egy URL link felépítése.....	20
4. ábra: Egy reszponzív weboldal illusztrációja különböző méretű eszközökön	21
5. ábra: A Garten-und-Freizeit webalkalmazás böngészőben és telefonon	24
6. ábra: A letöltött alkalmazások növekedése Android és iOS eszközökre évekre bontva	25
7. ábra: A GET kérés Postman-ben	33
8. ábra: Használati eset diagram	38
9. ábra: Aktivitás diagram.....	39
10. ábra: A bejelentkezés szekvenciadiagramja.....	42
11. ábra: Egy esemény elkészítésének szekvenciadiagramja.....	43
12. ábra: A teljes rendszer architektúrája	45
13. ábra: Egyed kapcsolat diagram.....	46
14. ábra: A szerver architektúrája	47
15. ábra: A kliens architektúrája	48
16. ábra: A kliens drótváza.....	49
17. ábra: A WAR fájl elkészítése, parancsok	52
18. ábra: A WAR fájl bevetése a Tomcat webes felületén	52
19. ábra: Kezdőoldal	61
20. ábra: Regisztrációs oldal	62
21. ábra: Az ellenőrző (OTP) kód megadása	62
22. ábra: Bejelentkezési oldal	63
23. ábra: Főmenü, vendég nézet.....	64
24. ábra: Főmenü, felhasználó nézet.....	64
25. ábra: Főmenü, szervező nézet	64
26. ábra: Esemény részletei, vendég nézet.....	65
27. ábra: Esemény részletei, felhasználó nézet.....	65
28. ábra: Esemény részletei, szervező nézet	65
29. ábra: Szervező által generált QR kód	66
30. ábra: QR kód beolvasása	66
31. ábra: Esemény készítése	67
32. ábra: Esemény módosítása.....	67
33. ábra: Profil	68
34. Projektmenedzsment GitHub Project-ben	70

Kódrészletek jegyzéke

1. kódrészlet: Az adatbáziskapcsolat létrehozása az application.properties fájlban	50
2. kódrészlet: A szerver „main” függvénye	51
3. kódrészlet: A WAR fájl automatikus elkészítéséhez szükséges osztály	51
4. kódrészlet: Egy entitás osztály attribútuma.....	53
5. kódrészlet: Az egy a többhöz kapcsolat	53
6. kódrészlet: A több az egyhez kapcsolat	54
7. kódrészlet: Egy @PostMapping végpont elkészítése.....	54
8. kódrészlet: Egy „@DeleteMapping” végpont elkészítése.....	54
9. kódrészlet: A befecskendezés megvalósítása Spring Boot-ban	55
10. kódrészlet: Egy repository osztály saját e-mail alapján kereső függvénnyel.....	56
11. kódrészlet: Egy végpont publikussá tétele	56
12. kódrészlet: A hitelesítéshez szükséges token kigenerálása	57
13. kódrészlet: Egy végpont korlátozása a szerepkör alapján	57
14. kódrészlet: Az EventScanner osztály, amelyik kibővíti a „ConsumerStatefulWidget” osztályt	58
15. kódrészlet: Egy szolgáltató elérése a „ref” segítségével	59
16. kódrészlet: Szolgáltató készítése egy repository osztályból	59
17. kódrészlet: Egy HTTP művelet eredményének kezelése	60

1. Bevezető

1.1. Témaválasztás indoklása

A telefonos alkalmazás egy viszonylag új technológiának számít, de ma már szinte el sem tudnánk képzelni az életünket nélküle. A telefonos alkalmazásokról legelőször az IBM által készített, 1994-ben megjelent SPC, az első okostelefon, kapcsán beszélhetünk, mint azok elődei. Sokan azonban az 1997-ben megjelent Nokia 6110-el érkező, eredetileg árkád játék, Snake-et tekintik az első valós alkalmazásnak, de arra a formára, ahogy a legtöbben ma gondolunk rájuk, még ennél is többet kellett várni. Habár a telefonokba már 2000-ben kamerát építettek be, majd 2001-ben már 3G Internet elérést is, az első olyan okostelefonra, amely még mai standardok szerint az avatatlan szem számára is pusztán ránézésre beazonosítható, várni kellett egészen 2007-ig, amikor az Apple forgalomba hozta az első iPhone-t. Alig egy évvel később, 2008-ban, megjelent az App Store, Steve Jobs egy korábbi álma is, egy olyan alkalmazás, ahonnan a felhasználók bármikor alkalmazásokat tölthetnek le saját készülékeikre. Ettől kezdve a felhasználók harmadik féltől származó alkalmazások ezreit tölthették le, nem korlátozva magukat az eredeti fejlesztők által biztosított néhány alkalmazásra. Mára már nehéz elképzelni bármit, amire ne lenne egy alkalmazás, főbb kategóriák között lehet említeni a játékokat, közösségi médiát, produktivitást, életstílust, információáramlást megcélzó alkalmazásokat, amelyek napi szinten meghatározzák a legtöbbször életét. Az első iPhone megjelenése óta is sokat fejlődtek az alkalmazások és okostelefonok. Ma már elérhető technológia a vezeték nélküli töltés, a hajlítható telefonok, a kiterjesztett valóság vagy a valós idejű fordítás is. Az okostelefon felhasználók rohamos növekedése jól megfigyelhető a [1. ábrán](#), ahol 2016-tól láthatóak az adatok kiegészítve a 2023 és 2027 közötti várható növekedés becslésével [1] [2].

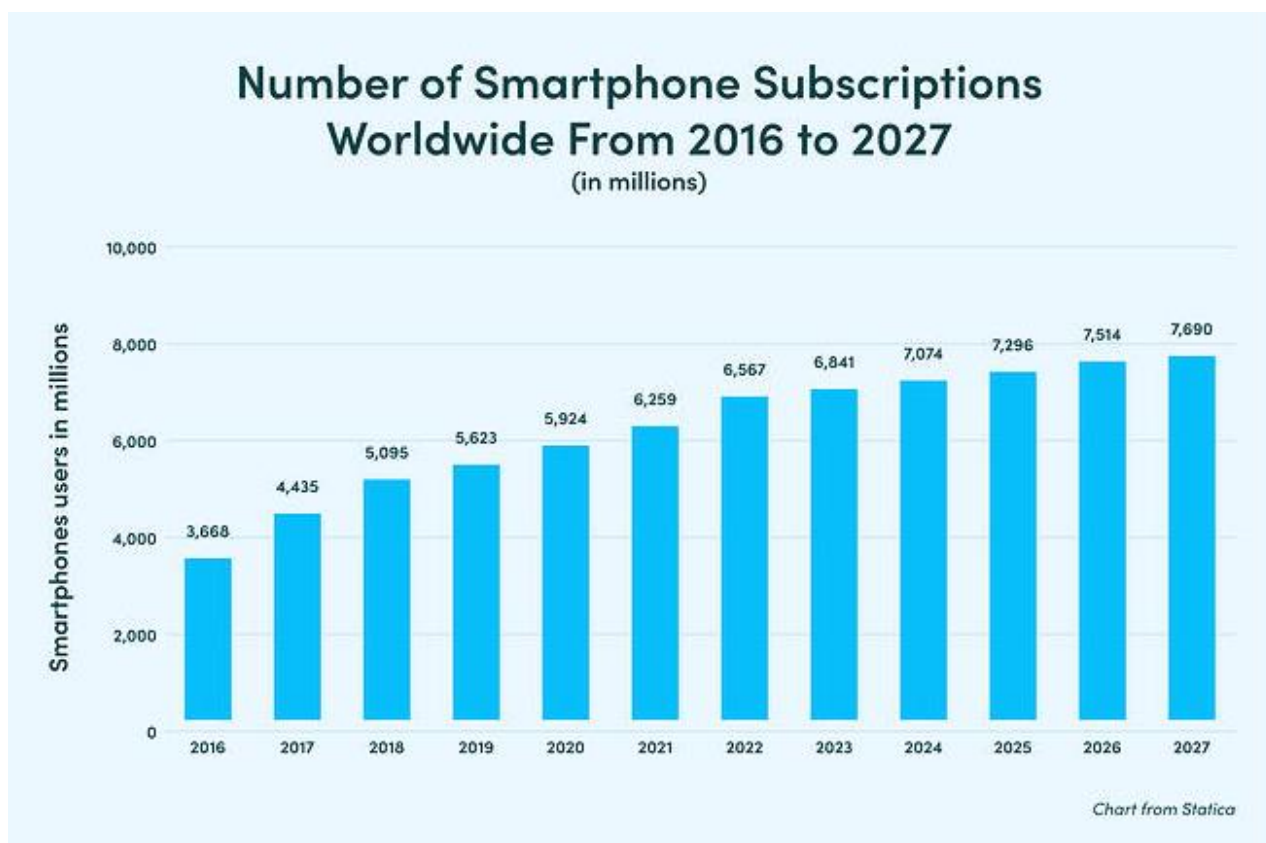
Az okostelefonok és alkalmazások meghatározzák az életünket, akár szórakozásról, munkáról, mindennapi életről vagy oktatásról legyen szó. Ehhez kapcsolódik az én témám. A középiskolás diákoknak, akik tovább szeretnének tanulni, egy fontos és nehéz döntést kell meghozniuk, hogy melyik intézményt válasszák továbbtanulásuk helyszínéül. Ennek a döntésnek a meghozatalában segíthetnek az egyetemeken megszervezett nyílt napok, ahol az egyetemek tanárai és diákjai bemutatják intézményük szakjait, felszerelését, termeit, eseményeit. Egy általános betekintést adhatnak az egyetem mindennapjaiba, így könnyítve meg a választást a középiskolás diákok számára. Ilyenkor sok eseményt szerveznek a szervezők, ezeken pedig sok diák megfordul, szóval

magától értetődő egy olyan alkalmazásnak az elkészítése, amely ennek az egésznek a lebonyolítását megkönnyíti mindkét fél számára.

Az előbb leírt igények fényében a dolgozatom egy olyan Flutter-ben elkészített telefonos alkalmazás, amely a diákok számára megkönnyíti a tájékozódást a különböző események között, egyszerűsíti a jelentkezést és úgy általában összegyűjti ezeket az eseményeket, hogy senki se maradjon le semmiről, ami érdekli. A szervezők számára pedig megkönnyíti az információk megosztását, a jelentkezők számontartását, névsorok írását, valamint események létrehozását, módosítását és törlését.

Az kliensalkalmazást Flutterben készítettem el, mert ez tökéletes kisebb és közepes alkalmazások fejlesztésére Android és iOS rendszerekre egyaránt, teljesítményben pedig alig marad el a natív megoldásoktól, továbbá könnyen lehet vele modern felhasználói felületet készíteni. A szerver egy Java programozási nyelvben elkészített Spring Boot program. Itt a választásom azért esett erre, mert a Spring Boot jól dokumentált és arra törekszik, hogy minél egyszerűbb legyen a fejlesztőnek, megóvva a sokszor ismétlődő kódrészletek írásától. Az adatbázis egy MySQL adatbázis, ami hatékony több millió rekord CRUD műveleteinek elvégzése esetén is.

Ez a dolgozat a fent említett alkalmazás dokumentálására jött létre és több nagyobb fejezetet tartalmaz. A bevezető után tárgyalom az adatbázist, majd a szerveroldali alkalmazást, végül pedig a kliensalkalmazást részletesebben. Az utóbbi két fejezet esetében külön tárgyalom a használt technológiákat, a funkcionális és nem funkcionális követelményeket, a fejezet második felében pedig a megvalósítás részleteit először fejlesztői majd felhasználói szemszögből.



1. ábra: Az okostelefon felhasználók száma évekre lebontva

Képforrás: [2]

1.2. Elméleti háttér

Sok olyan telefonos alkalmazás és webhely készült és készül a mai napig, melyek az eseményszervezéssel kapcsolatos teendőket segítik, mint a helyfoglalás, a résztvevők informálása, értesítések küldése a résztvevőknek, események népszerűsítése a közösségi oldalakon, jegyek eladása, eseménnyel kapcsolatos számlák kiállítása. Ebből jónéhány kifejezetten a felsőoktatási intézményeket célozza meg, azok igényeit próbálják kielégíteni a lehető legspecifikusabban. Kutatásomat egy körképpel kezdtem, hogy milyen hasonló alkalmazások és weboldalak vannak a piacon, ennek során többet is kipróbáltam, valamint több formátumot is számításba vettem és tanulmányoztam, hogy betekintést nyerjek a különböző ötletekbe és megvalósítási módszerekbe. Ezzel a betekintéssel egyszerre próbáltam felhasználói és programozói szemszögből is vizsgálni az adott alkalmazásokat és weboldalakat, hogy tudjam milyen elvárásoknak kell megfelelnem és mik a lehetőségeim.

A célom az volt, hogy egy kinézetre modern, intuitív és megkapó, funkcionalitásait tekintve egy lényegretörő és hasznos alkalmazást készítsek, amelyet a középiskolás diákok és egyetemi tanárok könnyen megtanulhatnak használni és szívesen is használnak, mert megkönnyít egy fontos feladatot az életükben. Ezen célok elérésével biztosíthatom, hogy a dolgozatom végeredménye megfeleljen a modern felhasználói elvárásoknak és egy hasznos alkalmazást hozzak létre.

Egy ilyen alkalmazást több különböző formában is el lehet készíteni, mint például telefonos alkalmazásként, weboldalként vagy progresszív webalkalmazásként. Úgy gondolom azonban, hogy az én esetemben is fontos először megvizsgálni, hogy a különböző technológiáknak és formátumoknak milyen előnyeik és hátrányaik vannak. A következő alfejezetekben ezeket fogom taglalni, valamint adok mindegyikről egy átfogó képet. A szerver elkészítéséhez szükséges technológia kiválasztásával könnyebb dolgom volt hisz tudtam, hogy bármilyen kliensprogramot választok annak igényeit egy REST API szerverrel ki tudom elégíteni. Továbbá a szerverhez használt technológia nem befolyásolja a viszonyt a felhasználókkal, amíg az gyorsan és megbízhatóan működik. A választásom a Spring Boot-ra esett, mert az egy modern, átgondolt technológia, amely felgyorsítja a fejlesztés menetét.

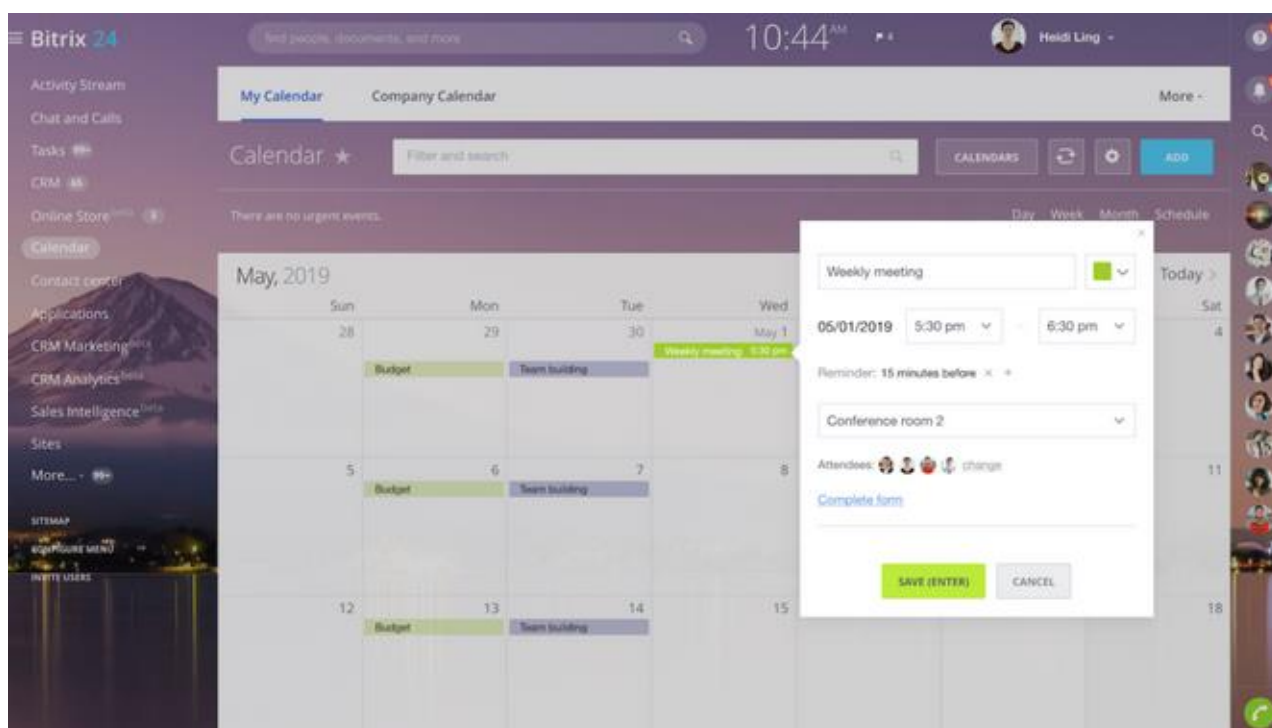
Az események szervezésére használt programokat annak mentén is meg lehet különböztetni, hogy elsősorban milyen típusú események szervezésében tudnak segíteni. Például a Steven Feingertz és Jeff Kear által készített *Planning Pod* olyan eszközökkel rendelkezik, amelyek elsősorban az oktatási intézményeknek lehetnek hasznosak az eseményszervezésben. Ezt csak böngészőn keresztül érhetjük el, nincs telefonos alkalmazás. Hozzáférhetünk olyan funkcionalitásokhoz, amelyek segítenek a résztvevők menedzselésében, a produktivitás növelésében, üzletkezelésben és személyre szabásban.

Kezelhetjük vele az ülésrendet, regisztrálhatunk vendégeket, kezelhetjük a vendégek listáját. Készíthetünk vele ütemtervet, költségvetési keretet, teendő listát vagy naptárakat [3] [4].

Egy másik csak weboldalként elérhető eseményeket kezelő rendszer, a Mykel Nahorniak és Nate Mook által létrehozott *Localist*, amely főként a turizmussal foglalkozó szervezeteknek lehet hasznos. A Localist egy egyszerű naptár, amelyben megtalálhatóak a különböző események. Célja, hogy növelje a résztvevők számát a tudatosság növelésével azáltal, hogy megkeresik a számukra érdekes eseményeket, majd elmentik, hogy ne felejtsek el és azzal, hogy többen szereznek tudomást az eseményekről. Ezt a webhelyet használva készíthetünk egy interaktív naptárat minden eseménynek, amelyet majd megoszthatunk különböző platformokon, így juttatva el az emberekhez. Hangsúlyt fektet arra, hogy a konverzió az itt létrehozott események és az eseményszervezők webhelye között növekedjen, vagyis minél többen azok közül, akik találkoznak az eseménnyel rákattintsanak a linkre,

ami az eseményszervező webhelyére navigálja őket. A Localist kapcsolódik a Facebookhoz, így a felhasználók bejelölhetik, hogy érdekli őket egy esemény, illetve láthatják, hogy mely ismerőseiket érdekli még az esemény [3] [5].

A [2. ábrán](#) egy több platformra is elérhető rendszer a *Bitrix24*, amelyet Sergey Rizhikov készített. Ez elérhető webhelyként, valamint Androidon, iOS-en és Mac-en is. Ez a rendszer elsősorban számlázásra lett létrehozva, de sok cég használja események szervezésére, mert rendelkezik az összes ehhez szükséges eszközzel. Kezelhetjük vele az ügyfeleket és az eladókat, kiállíthatunk számlákat. Küldhetünk a segítségével e-maileket. Megoszthatunk vele naptárakat, amelyben lefoglalunk helyszíneket bizonyos időpontokban. Van még olyan funkcionálisága is, amellyel csapatokat irányíthatunk, adhatunk nekik feladatokat a Bitrix24-en keresztül [3] [6].



2. ábra: Esemény készítése Bitrix24-el

Képforrás: [3]

1.3. Technológiai kitekintő

1.3.1. Webhely

A webhely weboldalak más néven weblapok összessége. A weboldalak olyan internetes dokumentumok, amelyek nyilvánosan elérhetők, és amelyeket a böngészők segítségével nyithatunk meg például számítógépeken, laptopokon, táblagépeken vagy telefonokon. A legnépszerűbb böngészők a Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge és az Opera. A weboldalak egyik fő komponense a hiperhivatkozás, amelyre rákattintva a böngésző elnavigálja a felhasználót egy másik weboldalra vagy webhelyre, így navigálva tulajdonképpen a különböző oldalak és webhelyek között. Egy webhely általában egy adott téma köré csoportosuló weboldalakból tevődik össze, amelyek tartalmazhatnak például szövegeket, képeket, videókat, hiperhivatkozásokat és egyéb médiatartalmakat. A néhány statikus weboldalt magába foglaló webhelyektől a párhuzamosan több webalkalmazást futtató webhelyekig terjedő skálán rengeteg webhely létezik, amelyek összessége alkotja a világhálót, angolul World Wide Web-et (www). A webhelyek általában követik a standardokat, így a legtöbb webhelynek van egy honlapja más néven kezdőlapja, amely a webhely betöltésekor elsőként jelenik meg, innen a hiperhivatkozásokkal lehet eljutni a többi oldalra vagy más webhelyekre [7] [8].

A webhelyeket és weboldalakat az egységes erőforrás helymeghatározó (URL) hivatkozással érhetjük el, amelynek a [3. ábrán](#) látható részei vannak. A URL a URI, egységes erőforrás azonosító, egy sajátos formája. Akkor használjuk a URL-t, amikor egy webkliens egy kérést küld egy szervernek, hogy elérjen valamilyen erőforrást. Röviden összefoglalva, a URI bármilyen karakterlánc lehet, amely azonosít egy erőforrást, míg a URL egy olyan URI, amely a helye vagy elérési módja szerint azonosítja az adott erőforrást az erőforrás neve, azonosítója vagy egyéb tulajdonsága helyett. Egy az URI-nél újabb protokoll az IRI, amely sokkal több karakter használatát engedélyezi az azonosítóban [9].

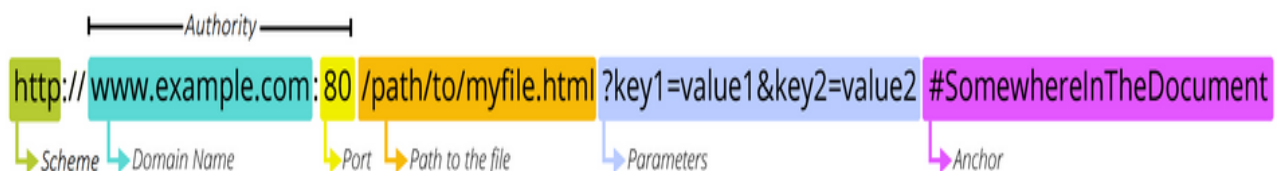
Séma: a séma azonosítja a protokollt, amelyet a kliens kell használjon erőforrás eléréséhez. Ez a webhelyek esetében általában a HTTP vagy HTTPS protokoll. A HTTP tulajdonképpen a HTTPS nem biztonságos formája. A HTTPS rövidítésben a S az SSL biztonsági réteget jelenti, amely azért felelős, hogy a szerver és a kliens között egy titkosított, biztonságos kapcsolat jöhessen létre. Mindkét protokoll tulajdonképpen metódusok halmaza, amely meghatározza az adatok cseréjének és küldésének mikéntjét. Léteznek természetesen egyéb protokollok is, mint a mailto, amely egy email kliens megnyitására szolgál [9].

Hatóság/authority: ez két részből tevődik össze, az első a domén név, a második a port szám. Minden webhelyhez tartozik egy domén név, amely azonosítja azt, hogy melyik szerverhez intézzük a kérést. Ilyen domén név például a `www.google.com`, `www.facebook.com` vagy a `www.wikipedia.org`, ahol a második pont utáni betűk arra utalnak, hogy milyen szervezetek birtokolják azokat. Például `.com` a kereskedelmi webhelyekre utal, a `.edu` tanintézményekre, a `.org` pedig nonprofit szervezetekre. Ezek mellett népszerűek még a országokra utaló végződések, mint a `.ro`, a `.de` vagy `.it`. A domén név kiváltható egy IP címmel is, de ez a kényelmetlensége miatt elég ritka. A port szám felfogható a „kapunak” a számaként, amelyet a szerver használ a kommunikációra. Ez nem muszáj explicit a URL részét képezze, ha a szerver a standard portokat használja, amely a 80-as port HTTP protokoll esetén és a 443-as a HTTPS protokoll esetén [9].

Az útvonal: ez az erőforrás útvonala a webszerveren belül, amely szükséges, hogy a webszerver tudja, hogy mely fájlokat kell visszaküldje válaszként a kérésre [9].

Paraméterek: a kérés URL-jében plusz paramétereket adhatunk meg, amelyet a webszerver arra használhat, hogy pontosítsa a kérés eredményét vagy plusz műveleteket végezzen el mielőtt bármit visszatérítene. A különböző paramétereket az & szimbólummal választjuk el egymástól. Megadásuk csak ritkán kötelező, arról pedig, hogy egy webhely URL-jében használhatunk paramétereket vagy sem, a webszerver tulajdonosát kell megkérdeznünk, vagy az adott dokumentációt kell tanulmányoznunk, ha rendelkezésünkre áll az [9].

Horgony: a horgony úgy működik, mint egy könyvjelző, amely egy adott helyre navigál az erőforráson belül. Ez egy HTML dokumentumban azt jelenti, hogy betöltéskor a böngésző az adott helyre görget, míg egy videó az adott időbélyegtől indul, ha lehetséges.



3. ábra: Egy URL link felépítése

Képforrás: [9]

Kiindulva a tényből, hogy 2022-ben körülbelül 400 millió aktív webhely létezik, és megközelítőleg 4.95 milliárd ember, aki internet hozzáféréssel rendelkezik, kijelenthető, hogy ez egy óriási piac. Hogy maximalizáljuk az weboldalunk elérését ma már általános, hogy reszponzív weboldalakat készítünk [9].

Reszponzív weboldal: a részponzív tervezés, melynek vázlata megtekinthető a [4. ábrán](#), a webfejlesztés egy olyan megközelítése, amelyben az adott oldal alkalmazkodik a képernyő méretéhez, amelyen a felhasználó megnyitja azt. Ezzel tulajdonképpen javítja az oldal elérhetőségét, használhatóságát és a felhasználó élményt. Az elemek a weboldalon relatív egységekben vannak méretezve (%). A 2010-es évek elején felvetődött a probléma, hogy egyre többen próbálnak meg webhelyeket telefonokon és táblagépeken elérni. Akkor még a webhelyek többsége nem támogatta az ennyire kis képernyőket. Erre válaszként a fejlesztőknek két megoldásuk volt. Az egyik, hogy az adaptív megközelítést választják és a különböző méretű képernyőkre elkészítik a webhely különböző változatait rögzített méretű elemekkel. A másik opció, hogy a webhelyet a részponzív tervezési mintát követve implementálják. Hosszútávon a második megoldás bizonyult jobbnak, amellyel egyszerre érte el a következőket: nagyobb felhasználóbázis elérését tette lehetővé, időt takarított meg, mert a fejlesztők csak egy változat elkészítésére kellett koncentrálniuk, javították vele a keresőmotor optimalizációját, mert azok a részponzív weboldalakat preferálják, biztosították vele a márka és dizájn egységességét. Méret tekintetében 3 részre osztják a képernyőket a tervezők. Vannak a 320 és 767 pixel közötti eszközök, a 768 és 1023 pixel közöttiek és a harmadik kategória, az 1024 és annál nagyobb eszközök. Ugyanakkor alapszabály, hogy részponzív webhelyek tervezésénél mindig a mobileszközöket kell elsősorban szembe előtt tartani, mert azok a legkisebbek majd a különböző részeket felfele skálázni. Ebben segít például az SVG képformátum, ami minőségvesztés nélkül nagyítható [\[10\]](#).



4. ábra: Egy részponzív weboldal illusztrációja különböző méretű eszközökön

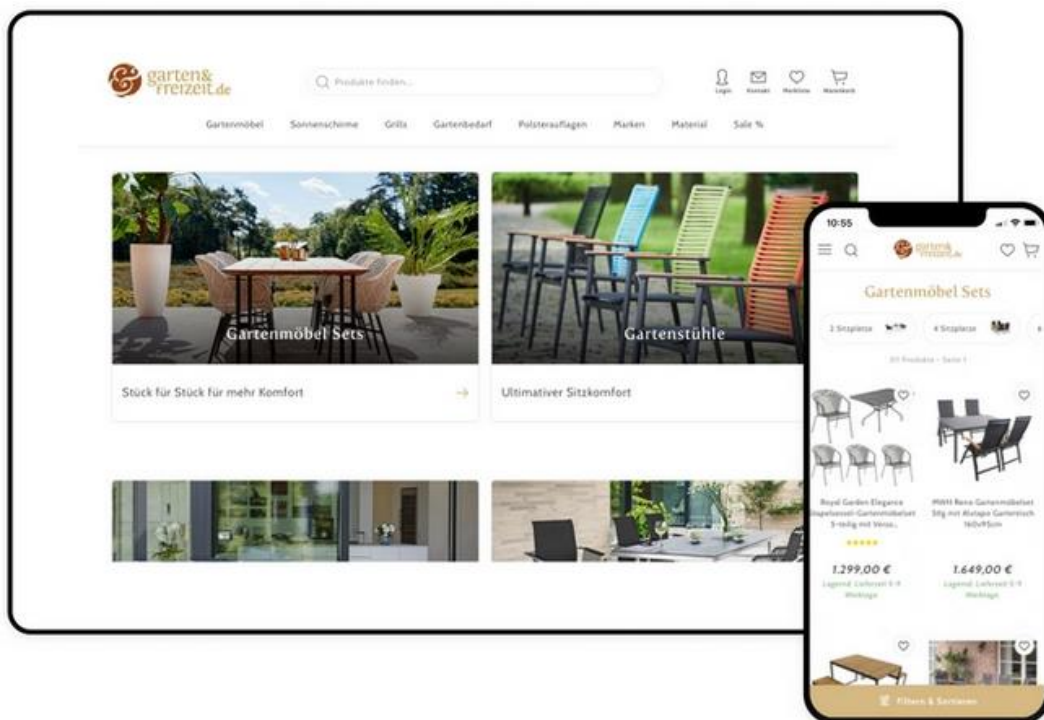
Képforrás: [\[10\]](#)

1.3.2. Progresszív webalkalmazás

A progresszív webalkalmazások (PWA) JavaScript, CSS és HTML segítségével készülnek úgy, hogy egyszerre rendelkezzenek a webhelyek és natív alkalmazások pozitív tulajdonságaival. Hasonlóan működnek, mint a webhelyek, ami azt jelenti, hogy elérhetőek egy böngésző segítségével, ugyanakkor olyan tulajdonságaik is vannak, amelyek a telefonos alkalmazásokra jellemzőek. Telepíthetők, gyorsan és megbízhatóan működnek még internetkapcsolat nélkül is, jobban együtt képesek működni az operációs rendszerekkel, mint a hagyományos webhelyek, képesek push üzenetek küldésére. Elérhetik a wifi hálózatot és használhatnak olyan szolgáltatásokat, mint a GPS, kamera és Bluetooth. További pozitívum, hogy a felhasználók szeretik, ha kedvenc alkalmazásaikhoz külön parancsikon tartozik telefonjaikon és nem kell a böngészőben megkeressék. De miért is van erre szükség? Először 2014-ben történt meg, hogy több ember használt telefont, mint amennyi számítógépet, 80%-a az üzletben vásárlóknak mobil eszközön keresi meg a releváns információkat, hasonlítja össze az árakat és keres esetleg más üzletet, ahol megvásárolhatja az adott terméket olcsóbban. Továbbá becslések szerint 2025-re az online vásárlások 44.2%-a fog telefonon történni. Már korábban a reszponzív webhelyek készítésénél is a telefonok volt elsőként fókuszban, ez a trend csak tovább erősödött a PWA-k megjelenésével. Azután, hogy 2015-ben Alex Russel bejelentette a PWA létrejöttét fontos mérföldkövet ért el 2018-ban, amikor a Google a keresési találatok megjelenítésénél számításba vette, hogy mely weboldalak támogatják jobban a telefonokat és, hogy mennyire gyors egy adott webhely, továbbá a Microsoft és az Apple is támogatta a technológiát. 2020-ban a Play Store, az App Store és a Microsoft Store is lehetővé tette, hogy PWA alkalmazásokat töltsenek fel, még ebben az évben olyan népszerű márkák, mint a Starbucks, Spotify, Tiktok és az Uber éltek is a lehetőséggel. Első látásra nem teljesen egyértelmű, hogy egy webhely PWA vagy sem. Ahhoz, hogy PWA-nek tekinthető legyen szükséges, hogy rendelkezzen néhány tulajdonsággal, amelyek alapkövetelmények. Telepíthető: a webalkalmazásnak telepíthetőnek kell lennie. Ez viszonylag könnyen elérhető és plusz előny, hogy Android és iOS eszközökkel egyaránt kompatibilis anélkül, hogy két különböző alkalmazást kellene fejleszteni mindkét eszközre. Egy webalkalmazás általában telepíthető az adott platform üzletéből, de az is megoldható, hogy a böngészőből az adott linkre kattintva azonnal telepítse a felhasználó az eszközére. Ez is növeli annak az esélyét, hogy egy felhasználó a böngészőről az alkalmazásra vált, ami végsősoron fontos. Linkelhető: fontos, hogy egy webalkalmazás linkelhető legyen, ami azt jelenti, hogy egy URL link

segítségével megoszthatják egymás között a felhasználók a webalkalmazást. A URL használatával a webalkalmazás egyszerűen megnyitható egy böngészőben. Ez lehetővé teszi, hogy ne legyen kötelező a telepítés annak, aki nem igényli a gyors elérést telefonról. Kapcsolatfüggetlenség: egy webalkalmazás, ha telepítve van egy felhasználó eszközére, rossz vagy nem létező internet kapcsolat esetén is működőképes kell legyen. Ehhez persze szükség van arra, hogy adatokat és korábbi műveletek eredményeit tárolja el az eszközön. Push üzenetek: egy webalkalmazásnak képesnek kell lennie, hogy push üzenetek küldjön, így életben tartva a kapcsolatot maga és a felhasználó között akkor is, ha az épp nem használja aktívan. Ilyen üzenet lehet egy sporteredmény, egy leárazás vagy kupon akció is. Reszponzivitás: fontos, hogy egy webalkalmazás rezponzív legyen, ami azt jelenti, hogy jól néz ki és használható a legkisebb okostelefonokon és legnagyobb monitorokon egyaránt. Biztonság: azt jelenti, hogy a webalkalmazást használva a kapcsolat a felhasználó, az alkalmazás és a szerver között biztosítva van. Harmadik fél nem fér hozzá az adatokhoz. Visszafele kompatibilitás: egy webalkalmazás alapvető szinten használható kell legyen a régebbi böngészőkben is még ha az összes funkcionalitás csak az modern böngészőkben működőképes is [11] [12] [13].

Van néhány hátránya is a webalkalmazásoknak. Sajnos egy telefon lehetőségeit a natív alkalmazások még mindig jobban ki tudják használni. Például a telefonban lévő telefonszámok nem elérhetőek egy webalkalmazás számára főként biztonsági okok miatt. Az is megemlítenő, hogy a teljesítményük folyamatosan növekszik, de még mindig lemaradnak a natív alkalmazások teljesítményétől, ami egy versenykőzegben komoly hátrányt jelenthet. További korlátozásokat szenvednek el az iOS eszközöket használó felhasználók ugyanis ott a push üzenetek küldése nem támogatott és ezen a platformon a böngészők közül is csak a Safari támogatja a webalkalmazásokat. Példa a progresszív webalkalmazásra a [5. ábrán](#) látható Garten-und-Freizeit [11].

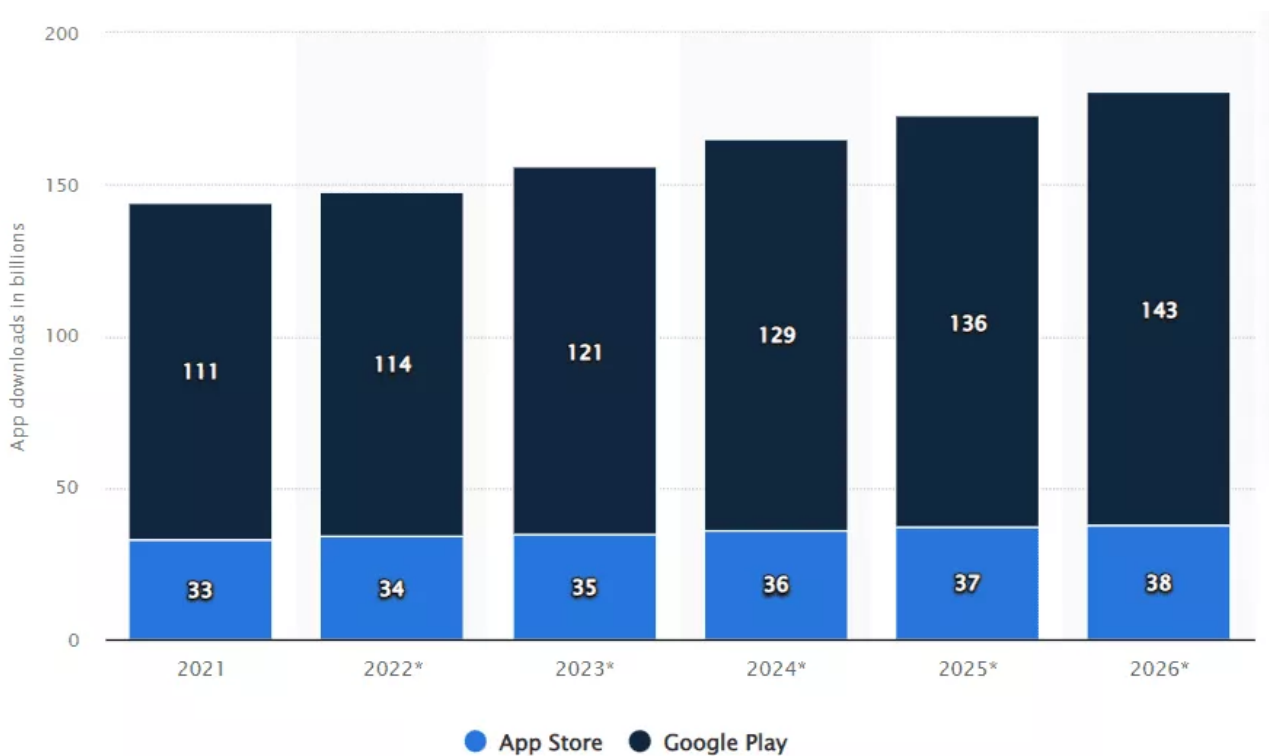


5. ábra: A Garten-und-Freizeit webalkalmazás böngészőben és telefonon

Képforrás: [13]

1.3.3. Mobilalkalmazás

Az alkalmazás egy olyan program, amely futtatható egy adott operációs rendszeren, legyen az Windows, Android, iOS vagy bármi egyéb. Az én esetemben most a telefonos alkalmazásoknak van nagyobb jelentősége, ezért ezekkel foglalkozom. Elsőként a natív alkalmazások jelentek meg, amelyek egy adott operációs rendszerrel kompatibilis technológiával készültek. Ezeket az alkalmazásokat mindig el kellett készíteni külön mindegyik operációs rendszerre, mert egy natív androidos alkalmazás nem telepíthető és futtatható iOS rendszeren. Példaként hozhatok olyan nagy neveket, mint a WhatsApp, Spotify, Waze vagy Pokemon Go, amelyek mindegyike natív alkalmazásként készült el külön a két vezető telefonos operációs rendszerre. Később aztán megjelent egyre több keresztplatformos technológia, mint a Flutter, React Native vagy az Ionic, amelyek lehetővé tették, hogy ugyanazzal a kódbázissal egyszerre készítsenek a fejlesztők alkalmazásokat a különböző rendszerekre. Itt olyan népszerű alkalmazásokat említhetek meg, mint a Google Pay, Instagram, Airbnb vagy Walmart. A mobilalkalmazások letöltésének száma várhatóan a következő években is nőni fog. A [6. ábrán](#) ennek a növekedésnek a becslése látható [14].



6. ábra: A letöltött alkalmazások növekedése Android és iOS eszközökre évekre bontva

Képforrás: [15]

Mindkét megközelítésnek vannak előnyei és hátrányai, amit fontos számításba venni mielőtt egy alkalmazás fejlesztésébe kezdenénk. A natív alkalmazások előnyeként tekinthető, hogy jobb teljesítményt nyújtanak, egyszerűen gyorsabban működnek, mint a progresszív webalkalmazások vagy a keresztplatformos társaik. Ez lehet, hogy nem észrevehető egy egyszerű alkalmazásnál, de azoknál, amelyek sok műveletet végeznek, vagy sok animációt mutatnak a felhasználóknak, mint egy játék, ott a különbség jelentős. Hozzáférés a platform lehetőségeihez: a natív alkalmazások hozzáférnek, és jobban együttműködnek a platform nyújtotta lehetőségekkel, mint a kamera Bluetooth, névjegyek. Biztonság: a natív technológiákkal biztonságosabb alkalmazásokat lehet készíteni, mert azok hozzáférnek a beépített platform specifikus biztonsági rendszerekhez. Nem minden esetben lesz egy keresztplatformos alkalmazás eredendően sérülékeny, de egy olyan szempont, amelyet számításba kell venni a fejlesztés előtt a felhasználók adatai érdekében.

Kevesebb hiba: külön kódbázist tartani a különböző platformoknak végül kevesebb hibát eredményez, mert nem kell egyszerre több platform technológiai sajátosságára figyelni.

Skálázhatóság: egy natív alkalmazás a nagyobb teljesítménynek köszönhetően jobban viseli, ha a terhelése megnő idővel. Kisebb az esélye, hogy a nagy terhelés alatt összeomlik. Könnyebb piacra dobni: mind a Google Playnek, mind az App Store-nak megvannak a maga standardjai, amelyeket egy alkalmazásnak teljesíteni kell, hogy fel lehessen tölteni oda. Ezeket könnyebb teljesíteni a natív technológiákkal, mint a keresztplatformosakkal. Frissíthetőség: a natív technológiákat használva könnyebb egy új Android vagy iOS fejlesztéseket az alkalmazásba építeni, mert ezekben a fejlesztők közvetlen hozzáférnek az SDK frissítésekhez. A gyors frissítés javítja a felhasználók termékkel kapcsolatos élményeit. Jobb felhasználói élmény: a felhasználók hozzá vannak szokva, hogy egy adott alkalmazás hogy néz ki a különböző platformokon. Követve ezeket a mintákat, formákat egy olyan alkalmazást fejleszthetünk, amely sokkal ismerősebb és szerethetőbb számukra. A natív technológiákkal sokkal könnyebb a külön platformokra jellemző kinézetet megvalósítani [15] [16].

Most, hogy már felsoroltam a natív technológiák legfontosabb előnyeit, megteszem ugyanezt a keresztplatformos technológiákkal is, hogy a mérleg nyelve legyen kiegyenlítve.

Alacsonyabb költségek: egy ilyen alkalmazás fejlesztéséhez elég egyetlen csapat fejlesztő, amely elkészíti, majd karbantartja azt. Ez jelentősen csökkenti a szükséges költségeket. A kód újrahasználatossága: mivel csak egyetlen fejlesztőcsapat és kódbázis van, ezért nincs probléma azzal, hogy az üzleti logikában különbségek vannak. Konzisztensebbek a különböző platformokon futó alkalmazások, mert lényegében ugyanaz a kód fut mindenhol.

Gyors fejlesztés: az egyetlen kódbázisnak és a kód újrahasználatosságnak köszönhetően gyorsabb megy a fejlesztés és ezáltal hamarabb a piacra lehet dobni. Emellett még ezek a technológiák jobban optimalizálva vannak az tesztelésre, ez is gyorsítja a folyamatot. Könnyebb karbantartás: egy adott frissítést vagy javítást könnyebb elvégezni, ha csak egyetlen kódbázis van. Ennek hála ezek is konzisztensebbek a különböző platformokon. [15] [16].

1.4. Célkritikázések

A kutatómunkából kiindulva eldöntöttem, hogy a projektem keretein belül a keresztplatformos technológiával megvalósított telefonos alkalmazásra fogok koncentrálni. A projektem célja így tehát egy olyan telefonos alkalmazás létrehozása, amelyben az egyetemi oktatók eseményeket készíthetnek, annak érdekében, hogy az egyetemre látogató középiskolás diákok könnyen

tájékozódhassanak a különböző események között. Célom, hogy modern technológiákat és megoldásokat használjak a projekt megvalósítására. Továbbá az is célom, hogy a létrehozott szerverként szolgáló REST API megbízhatóan, gyorsan és biztonságosan működjön, valamint a kliens alkalmazás intuitív és a modern felhasználói elvárásoknak megfelelő kinézetet és felhasználói felületet kapja a használat szempontjából. Miután túljutottam az első fontos kérdésen, hogy miben valósítom meg az egész projektet, további célokat fogalmaztam meg, amelyek a következők:

- Megvalósítani egy skálázható architektúrát mind szerver és kliens oldalon.
- Kialakítani a megfelelő relációt az adatok között.
- Kiválasztani a megfelelő relációs adatbázist.
- Implementálni a szükséges funkcionalitásokat szerver és kliens oldalon.
- Kiválasztani egy megbízható szolgáltatót e-mailek küldésére.
- Biztosítani az adatok kezelésének biztonságos módját.
- Megvalósítani a különböző szerepkörökkel rendelkező felhasználók kezelését.

A projektem két nagyobb részből tevődik össze, ezek a szerver és a kliensalkalmazás. Szeretném megvalósítani azt, hogy a felhasználók tudjanak először regisztrálni, majd bejelentkezni. Szeretném, ha a bejelentkezett felhasználóhoz tartozna egy szerepkör, amely meghatározza, hogy milyen lehetőségei vannak az alkalmazáson belül. A szervező szerepkörrel rendelkező felhasználók eseményeket hozhatnak létre és törölhetnek. Módosíthatják az eseményekhez tartozó adatokat, láthatják az eseményre jelentkezettek névsorát, valamint létrehozhatják a névsor írására szükséges QR kódot. A normál felhasználók megtekinthetik az események listáját, jelezhetik, hogy részt szeretnének venni, és a helyszínen a QR kód beolvasásával segíthetnek a névsor létrehozásában a szervezőket.

Célom az is, hogy a Flutter és Spring Boot keretrendszereket teszteljem. Ezek még új technológiáknak számítanak, kíváncsi vagyok, hogy mennyire segítenek a fejlesztésben a régebbi illetve a natív technológiákkal szemben, valamint hogy ütközm-e olyan akadályba, amelybe nem ütköznék az előbb említettekkel.

2. Technológiák bemutatása

2.1. Szerveralkalmazás

2.1.1. MySQL

Az adatbázis kiválasztásánál két szempontot kell figyelembe venni. Először is, hogy relációs vagy nem relációs adatbázisra van szükség, másodszor, hogy melyik adatbázis a legmegfelelőbb az elérhető opciók közül. Én egy relációs adatbázist választottam, mert azok strukturáltabbak, ez a strukturáltság rugalmasabbá teszi bonyolult sémák létrehozását. Továbbá, többek között a kulcsoknak köszönhetően az adathalmaz jobban megőrzi az integritását. Például nem lehet véletlen törölni olyan adatot, amely hivatkozva van egy másik táblában. Végül pedig a relációs adatbázisok törekszenek az ACID tulajdonságokat megtartani, ezek az atomiság, konzisztencia, izoláció és tartósság. Ezeknek köszönhetően a relációs adatbázisokon végzett műveletek megbízhatóbbak és az adat konzisztens marad egy esetleg hiba esetén is.

A relációs adatbázisok közül a két legszélesebb körben használt a MySQL és PostgreSQL. Én jobbnak láttam az előbbi választani, mert az egyszerűbb telepíteni és karbantartani, kifejezetten arra van optimalizálva, hogy az egyszerű CRUD műveleteket hatékonyan futtassa és nagyobb mértékben kompatibilis a különböző operációs rendszerekkel. Nem véletlen, hogy messze ez a legnépszerűbb relációs adatbázis.

A MySQL-t egy svéd cég a MySQL AB alkotott meg. Az egyszerű struktúráktól a komplex objektumokig bármit el lehet tárolni benne. A MySQL a legtöbb operációs rendszerrel kompatibilis. Az SQL nyelv maga 4 különböző nyelvből tevődik össze, amelyek a következők. A DQL (Data Query Language)-t arra használjuk, hogy lekérdezéseket futtassunk az adatbázison, mint lekérni egy oszlop maximumát. A DDL (Data Definition Language) strukturális parancsok futtatására való, mint egy tábla elkészítése vagy egy adattípus definiálása. A DCL (Data Control Language) a különböző engedélyek kiosztására, adminisztrátori jogok jóváhagyására vonatkozó parancsokat képes futtatni. Nem konkrétan az adatbázissal foglalkozik, hanem a felhasználók jogosultságaival. A DML (Data Manipulation Language) pedig a táblák módosítását teszi lehetővé, mint új adatok beszúrása, meglévők módosítása és törlése.

2.1.2. Spring Boot

Egy REST API fejlesztésre használt keretrendszer kiválasztásánál fontos figyelembe venni olyan szempontokat, hogy az adott keretrendszer milyen nyelvet használ, mennyire könnyen kivitelezhetők a szükséges funkcionalitások, hogy mennyire jól dokumentált a technológia. Számos választási lehetőség van, mint a Spring, Spring Boot, Django REST és Express Js. Én a dolgozatom megvalósítására a Spring Boot-ot választottam

A Spring Boot egy nyílt forráskódú, Java keretrendszer, amely a Spring-re épül, rendelkezik annak legtöbb előnyével és tulajdonságával. Elsősorban web API-k fejlesztésére lett optimalizálva. A Spring Boot-tal készített alkalmazások annotáció alapúak, és megkönnyíti a függőségek kezelését. Automatikusan konfigurálja a Springet és a harmadik féltől származó könyvtárakat. A Spring és Spring Boot legnagyobb előnye a többi keretrendszerrel szemben, hogy Java nyelvre épülnek. Ez fontos számomra, mert jobban ismerem a Javat, mint bármely más nyelvet. A Spring Boot egyéb fontosabb előnyei a Springgel szemben [\[19\]](#) [\[20\]](#):

- Segít kezelni a függőségeket.
- Tovább csökkenti az ismétlődő kódrészleteket és ezáltal a fejlesztési időt.
- Előre konfigurált beállításokkal érkezik, nincs szükség a manuálisan megírt XML konfigurációkra.
- Rendelkezik beágyazott Tomcat, Jetty és Undertow tárolókkal, szóval nincs szükség a WAR fájlok generálására és bevetésére.
- Rendelkezik a H2-vel, ami egy alapértelmezett memória alapú (in-memory) adatbázis, amelyet használhat, ha nincs más adatbázis kapcsolat beállítva.

2.1.3. JPA

Elkészíteni egy adathozzáférési réteget egy nehéz és sok ismétlődő kóddal járó feladat még Javában is, ugyanakkor sokszor kell ezzel foglalkozni, ha APIkat fejlesztünk. Ebben a feladatban segíthet a JPA (Java Persistence API), ami szolgáltatásokat nyújt és standardokat határoz meg az ORM eszközöknek. A JPA arra lett kifejlesztve, hogy sokkal könnyebbé tegye egy Java alkalmazás kommunikációját a relációs adatbázisokkal (írás, olvasás), mint például a MySQL. Ennek segítségével egy programozó olyan interfészeket írhat, amelyek kereső függvényeket tartalmaznak.

Ezeket a függvényeket nem kell manuálisan implementálni, mert a JPA kigenerálja ezek implementációit. Tulajdonképpen specifikációk, szabályok és irányelvek összessége, nem tartalmaz konkrét implementációkat, mert azokat a hozzátartozó eszközök adják meg, mint a Hibernate [\[21\]](#).

2.1.4. Hibernate

A Hibernate egy ORM (Object Relational Mapping) keretrendszer, amely biztosít egy absztrakciós réteget a fejlesztők számára. Ez azt jelenti, hogy a fejlesztőknek nem kell az implementációval törődniük, mert azt a Hibernate kigenerálja. A Hibernate belsőleg implementál bizonyos modulokat, mint a kérések, amelyek a CRUD műveleteket valósítják meg, illetve létrehozza a kapcsolatot az adatbázissal. Segítségével „tartós” logikát írhatunk, vagyis olyat, amely hosszútávon képes tárolni az adatokat, illetve dolgozni azokkal. A Hibernate független a használt adatbázistól, ezért is kifejezetten használt a vállalati közegben. A JPA által meghatározott logikát használja, illetve implementálja. Gyakorlatban a Hibernate kezeli és kapcsolja össze az Entity osztályokat az adatbázis megfelelő tábláival, valamint ez végzi a Java adattípusok leképezését az adatbázis azonos adattípusaira [\[22\]](#) [\[23\]](#).

2.1.5. API

Az API egy angol mozaikszó, ami az Application Programming Interface-ből származik, magyarul alkalmazásprogramozási felület. Programozásban arra használjuk, hogy két program kommunikációját tegyük lehetővé úgy, hogy egyik sem ismeri a másik függvényeinek a konkrét implementációját, csak azokat a függvényeket vagy végpontokat hívhatják meg, amelyeket a másik publikussá tett. Egy mindennapi példával élve az API olyan, mint egy étlap, a vendég csak kiválasztja az ételt a neve alapján, nem tudja, hogy a háttérben hogyan készül az étel. Ez a hozzáállás leegyszerűsíti és felgyorsítja a különböző alkalmazások fejlesztését, hisz a fejlesztőknek nem kell megírni a logikát, csak meg kell hívniuk a megfelelő függvényt és megvárniuk az eredményt. Ugyanakkor egy API-t több alkalmazás is használhat, így csökkentve a fejlesztési időt, újrahasználva a kódot [\[24\]](#).

2.1.6. REST

A REST (REpresentational State Transfer) egy szoftver architektúra, amely standardokat biztosít a rendszerek számára, hogy azok zavartalanul kommunikálhassanak az interneten keresztül. Azokat a rendszereket, amelyek megfelelnek a REST szabályainak, RESTful-nak szokták nevezni. Az egyik előnye a REST standardok használatának, hogy kettéosztja a szoftverfejlesztést szerver és kliensprogramra. Ez azt jelenti, hogy sem a szerver sem a kliens nem kell ismerje a másik implementációját és bármelyik megváltoztatható anélkül, hogy a másikat befolyásolná. Csak azt kell tudják, hogy milyen formában kell küldeni és fogadni az üzeneteket. Ha két kliens egy REST interfész ugyanazon végpontját meghívják, akkor ugyanazt az eredményt fogják kapni. Egy másik előnye a REST rendszereknek, hogy „állapot nélküliek”, ami azt jelenti, hogy egyiknek sem kell tudnia semmit a másik aktuális állapotáról. Ennek köszönhetően minden üzenet önmagában értelmezhető a felek számára, nem szükséges kiegészítő információk megadása [25].

2.1.7. RESTful API

Egy API-t többféle architektúra segítségével is el lehet készíteni, de azt az API-t, ami megfelel a REST által meghatározott szabályoknak, REST API-nak vagy RESTful API-nak nevezzük. Általában mindkét névvel a RESTful web API-ra utalnak, ami kifejezetten az interneten való kommunikációt valósítja meg két rendszer között. Működési elvét tekintve a felhasználó az API végpontjain keresztül lép kapcsolatba a szerverrel [25]. A lépések a következők:

- A felhasználó egy kérést küld a szerver felé, amelyet úgy kell formáznia, hogy azt a szerver megértse.
- A szerver megbizonyosodik, hogy a felhasználónak megvan a joga, hogy kommunikáljon az adott végponttal.
- A szerver feldolgozza a kérést és előkészíti a választ.
- A szerver visszatéríti a választ a felhasználó felé. A válasz tartalmazza, hogy a kérés sikeres volt vagy sem, illetve, ha sikeres volt, akkor tartalmazza a felhasználó által kért információkat.

Vannak alapértelmezett HTTP kódok, amelyeket egy RESTful API használhat [26]. Ilyenek a következők:

- 200: A kérést sikeres volt.
- 400: A kliens hibás végpontot hívott meg, vagy hibásan adta meg a kéréshez szükséges adatokat.
- 401: A kliens felhatalmazása nem volt sikeres, ha a kliens úgy hív meg egy végpontot, hogy nincs bejelentkezve, vagy a felhatalmazáshoz szükséges token érvényessége le van járva.
- 403: A kliens felhatalmazása sikeres volt, be van jelentkezve, a token érvényes, de nincs joga elérni a kívánt adatokat például, mert nincs rendszergazdai joga.

2.1.8. RESTful API kérések

Minden kérésnek tartalmaznia kell a következő információkat, hogy sikeres legyen.

URL: A REST architektúrában legtöbbször az URL (Uniform Resource Locator) azonosítja be, hogy melyik erőforrást szeretné elérni a kliens. Ezeket a URL-eket hívjuk még az API végpontjainak is. A URL is tartalmazhat paramétereket, amelyek a kérést pontosítják.

Fejlécek: A kliens és szerver közötti kommunikáció metaadatait tároljuk itt. Ilyen adat például az felhatalmazáshoz szükséges token vagy az, hogy a kérés milyen formátumban érkezik a szerverhez.

A fejléc valójában kulcs-érték párokat tartalmaz, amelyek egyszerűszöveges formában tárolják mind a kulcsokat, mind az ahhoz tartozó értékeket.

Test: A kommunikációban a kérésnek és a válasznak is van törzse. A kérés esetében olyan adatokat tartalmaz, amelyek szükségesek a kérés sikeres végrehajtásához, a válasz esetében olyanokat, amelyek szükségesek a kliensnek, amelyekért a kérés létrejött. A kérés teste a fejlécben meghatározott formátumban kell legyen, mint JSON vagy XML.

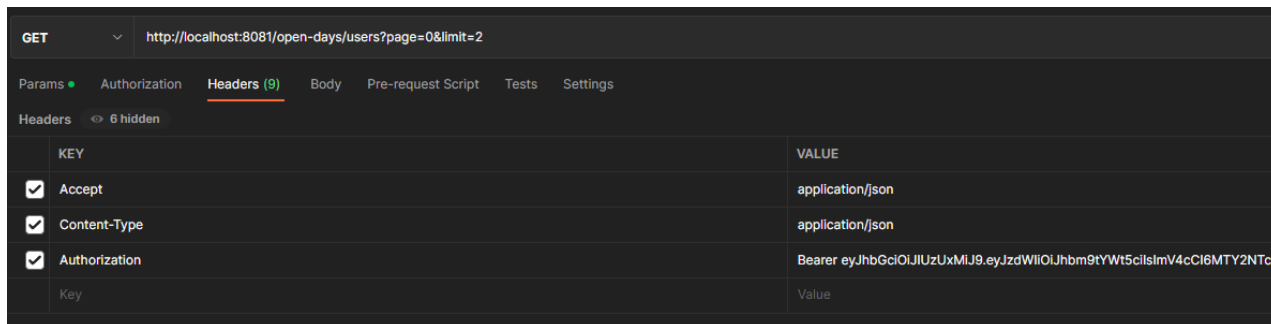
Paraméterek: A kérés tartalmazhat paramétereket, amelyek további információt adnak a szerver számára, hogy mit kell csinálnia. Ezek megadása legtöbbször opcionális.

Metódus: A RESTful API-k sokszor a HTTP (Hypertext Transfer Protocol) protokollok használatával vannak implementálva. A HTTP metódusok mondják meg a szervernek, hogy milyen műveletet kell végrehajtania a kapott adatokon [27]. A legfontosabbak a következők:

- GET: A kliens arra használja, hogy adatokat kérjen le a szervertől, amelyek az adott URL-en találhatóak.

- POST: A kliens arra használja, hogy adatokat küldjön a szerverhez, amelyeket az majd elment például egy adatbázisban, megőrzésre.
- PUT: A kliens arra használja, hogy már meglévő adatokat frissítsen a szerveren.
- DELETE: A kliens arra használja, hogy adatokat töröljön a szerverről.

A [7. ábrán](#) megtekinthető egy kérés Postman-ben, ami lekéri az első oldalról a felhasználókat, minden oldalon 2 felhasználó van. Ennek a kérésnek nincs törzse.



7. ábra: A GET kérés Postman-ben

2.2. Kliensalkalmazás

2.2.1. Flutter

A két legelterjedtebb keretrendszer platformfüggetlen mobilalkalmazások fejlesztésére a Flutter és a React Native. Én a kliens megvalósítására a Fluttert választottam, amelynek több oka is van.

A Flutter egy Google által készített SDK csomag, amely azzal a céllal született meg, hogy egy kódbázissal több platformon is futtatható, szép és natívan kompilált alkalmazást készíthessünk olyan platformokra, mint az Android, iOS, web és Windows.

Néhány előnye ennek a keretrendszernek, hogy az ebben készített alkalmazások kinézete nem függ a platformtól, mert a saját motorját használja a vizuális elemek megrajzolására a képernyőn, továbbá olyan dizájn csomagokat használ, amelyek egy adott platformra vannak optimalizálva, mint a Material és Cupertino. Saját rendermotorral rendelkezik, amely növeli az alkalmazás teljesítményét, valamint a Flutter-ben való fejlesztést gyorsítja a „Hot Reload” (grafikus változtatások esetén nincs szükség teljesen új build készítésére) és az előre biztosított UI elemek, widgetek.

A különböző csomagok segítségével használhatók olyan eszközök, mint a kamera, GPS, internet hálózat, tárhelyek vagy olyan szolgáltatások, mint a fizetések, felhőben tárolás, hitelesítés és reklámok. Mivel a Flutter automatikusan nem ad módot a függőségek befecskendezésére, ezért ez is csomagok segítségével valósítható meg (kiwi, riverpod, injectable) [30] [31].

2.2.2. Dart

A Dart egy Google által fejlesztett általános célú, nyílt forráskódú programozási nyelv, amely kifejezetten kliens alkalmazások létrehozására volt optimalizálva, mint weboldalak és mobil alkalmazások. A Dart 2011-ben jelent meg, C stílusú, de inspirálódott olyan nyelvekből, mint a Java, JavaScript és C#. Segítségével bármilyen platformra fejleszthetünk.

Egy objektumorientált nyelv, amely támogatja az általános fogalmakat, mint az osztályok, interfészek, függvények. A kód kompilálható gépi kóddá és JavaScript kóddá.

Mivel a Dart azzal a céllal volt készítve, hogy a fejlesztőknek minél gyorsabban menjen a kódolás, ezért sok előre beépített eszközzel rendelkezik. Ilyen a saját csomagmenedzsere, különféle kompilátorok, formázók, továbbá a Dart virtuális gép és a JIT (Just-in-Time) build lehetővé teszi, hogy minden változtatás azonnal futtatható legyen [32].

2.2.3. Riverpod

A Flutter alkalmazások esetében az egyik első dolog, amit meg kell határozni, hogy hogyan és mivel kezeljük az alkalmazáson belül a különböző állapotokat és azok változását. Ez akkor merül fel, amikor a háttérben megváltozik valamilyen adat és ezt tükröznie kell a felhasználói interfésznek is. Egyszerűen fogalmazva változik a képernyő tartalma.

Több módszer is létezik az állapotok kezelésére. Az alapértelmezett és egyben a legkönnyebb megoldás az a „setState” függvény használata a „Stateful” osztályokban. Annak ellenére, hogy a „setState” az alapértelmezett valójában csak ritkán használják. Legtöbbször külső megoldásokat használnak fejlesztésnél, mint a Riverpod, Redux vagy a Bloc [33].

A Riverpod a Provider csomag újradolgozása, egy olyan keretrendszer, amely meghaladja funkcionalitásában az elődjét.

- A hibákat kompilálási időben jelzi, futási idő helyett.

- Segítségével képesek vagyunk könnyen lekérni, eltárolni és frissíteni az adatokat interneten keresztül is.
- Segítségével könnyen készíthetünk szolgáltató (provider) objektumokat.
- Könnyen törölhetjük a szolgáltató objektumokat, amikor nincs már több szükségünk rájuk.
- Tesztelhető kód írható a segítségével, amelyet a widget fától független írhatunk meg.

3. Rendszerspecifikáció

3.1. Felhasználói követelmények (szerver)

A felhasználó csak a kliensprogramon keresztül, közvetett módon használja a szervert, így a felhasználói követelményeket sem elsősorban a szerverrel szemben fogalmazzuk meg. Ezek a követelmények jöhetnek a konkrét felhasználótól vagy a kliensprogramot fejlesztő programozóktól is. Itt olyan követelményekről beszélhetünk, mint:

- Legyen mindig elérhető.
- Az adatokat hibátlanul mentse el és térítse vissza.
- A kért műveletet minél gyorsabban hajtsa végre, lehetőleg 1-2 másodperc alatt.
- Ha egy művelet végrehajtása sikertelen, akkor a megfelelő hibaüzenettel jelezze azt.

3.2. Rendszerkövetelmények (szerver)

a) Funkcionális követelmények:

- Mentse el helyesen az adatbázisba a regisztrációnál megadott adatokat. .
- A jelszót titkosítva mentse el az adatbázisba.
- Küldjön egy e-mailt a felhasználónak arra az e-mail címre, amit az megadott regisztrációkor, hogy ellenőrizhessük, hogy a cím valóban a felhasználóhoz tartozik.
- Ellenőrizze, hogy létezik-e már regisztrált felhasználó azokkal az egyedi adatokkal, amiket a jelen felhasználó megadott. Ilyen adat a felhasználónév és e-mail cím.
- Ellenőrizze, hogy bejelentkezésnél a felhasználó a megfelelő felhasználónév és jelszó párost adta-e meg.
- Készítsen egy authorization header-t minden sikeres bejelentkezés után. Ez a fejléc legyen érvényes 30 napig.
- Mentse el az adatokat az adatbázisba helyesen minden új tevékenység és esemény létrehozásakor.
- Szükség esetén olvassa ki és térítse vissza a kliensalkalmazás számára a kért adatokat a felhasználókról és eseményekről.

- Küldjön egy e-mailt, amely segítségével a felhasználó beállíthat egy új jelszót, ha a régit elfelejtette.
- Zároljon minden olyan végpontot, amelyek nem kell kötelezően nyíltak legyenek az alkalmazás működése érdekében. A regisztrációhoz, a bejelentkezéshez és a jelszó visszaállításához használ nyílt végpontokat.
- Térítsen vissza egy pontos hibaüzenetet, ha az adott kérésnél nem lehetséges a kért adatokat visszatérítése a kliensalkalmazás részére.
- Biztosítsa, hogy minden felhasználó csak a felhasználói szintjének megfelelő végpontokat érhesse el.

b) Nem funkcionális követelmények

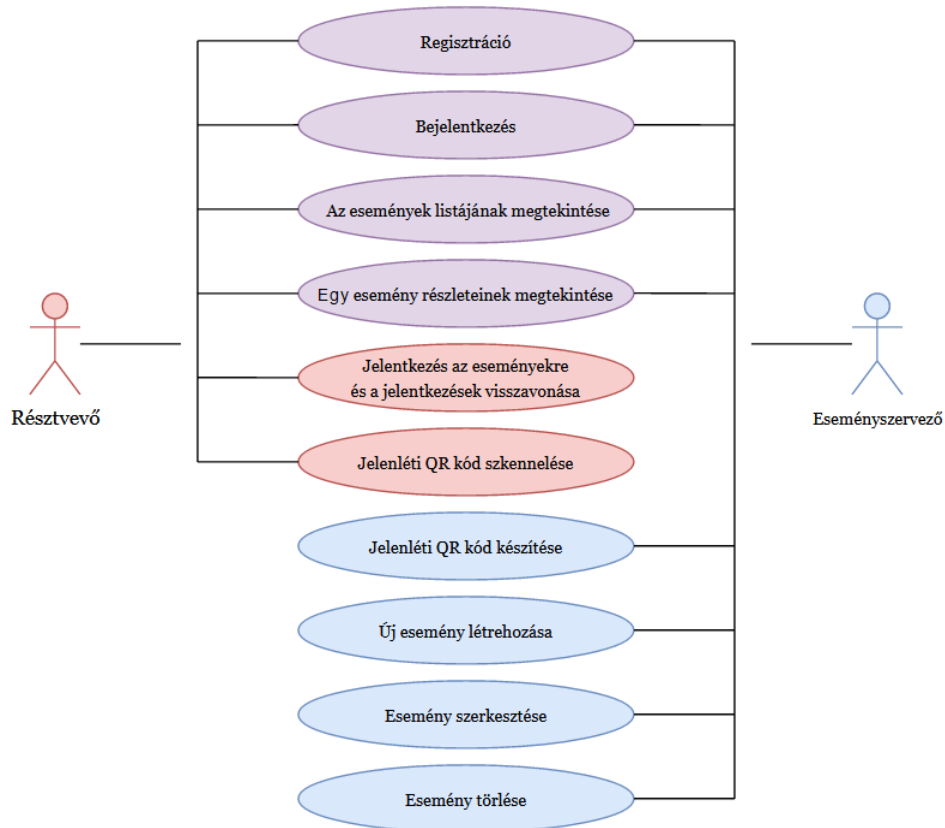
A szerver képes futni minden olyan Windows, Linux és iOS eszközön, ahol adottak bizonyos feltételek. Megtörténhet, hogy az alább említett programok újabb verziói sem megfelelők a szerver futtatásához. Ajánlatos pontosan az említett verziókat használni. A projekt bevetéséhez szükség van egy Java projektek futtatására képes webserverre. A legbiztosabb egy Apache Tomcat 9 szerver. A későbbi verziók, mint Apache Tomcat 10 nem kompatibilisek a projekttel. A Tomcat szerver és a projekt futtatásához szükség van a Java JDK-re. Mivel a fejlesztéshez Java 17 LTS verzió volt használva, és ez jelenleg a legutolsó hosszú távon támogatott verzió, érdemes ezt használni. A felhasználók által generált adatok tárolása érdekében szükség van egy MySQL adatbázisra. Szükség van továbbá internet kapcsolatra és megközelítőleg 52 MB szabad tárhelyre a WAR fájlként tárolt programnak, a fentiekben említetteken kívül. Alternatívaként a WAR fájl bevethető az Amazon AWS EC2 egy példányán is.

- Apache Tomcat 9
- Java 17 vagy nagyobb verzió
- MySQL 5.6-os vagy nagyobb verzió
- Internet kapcsolat, tárhely: 52 MB

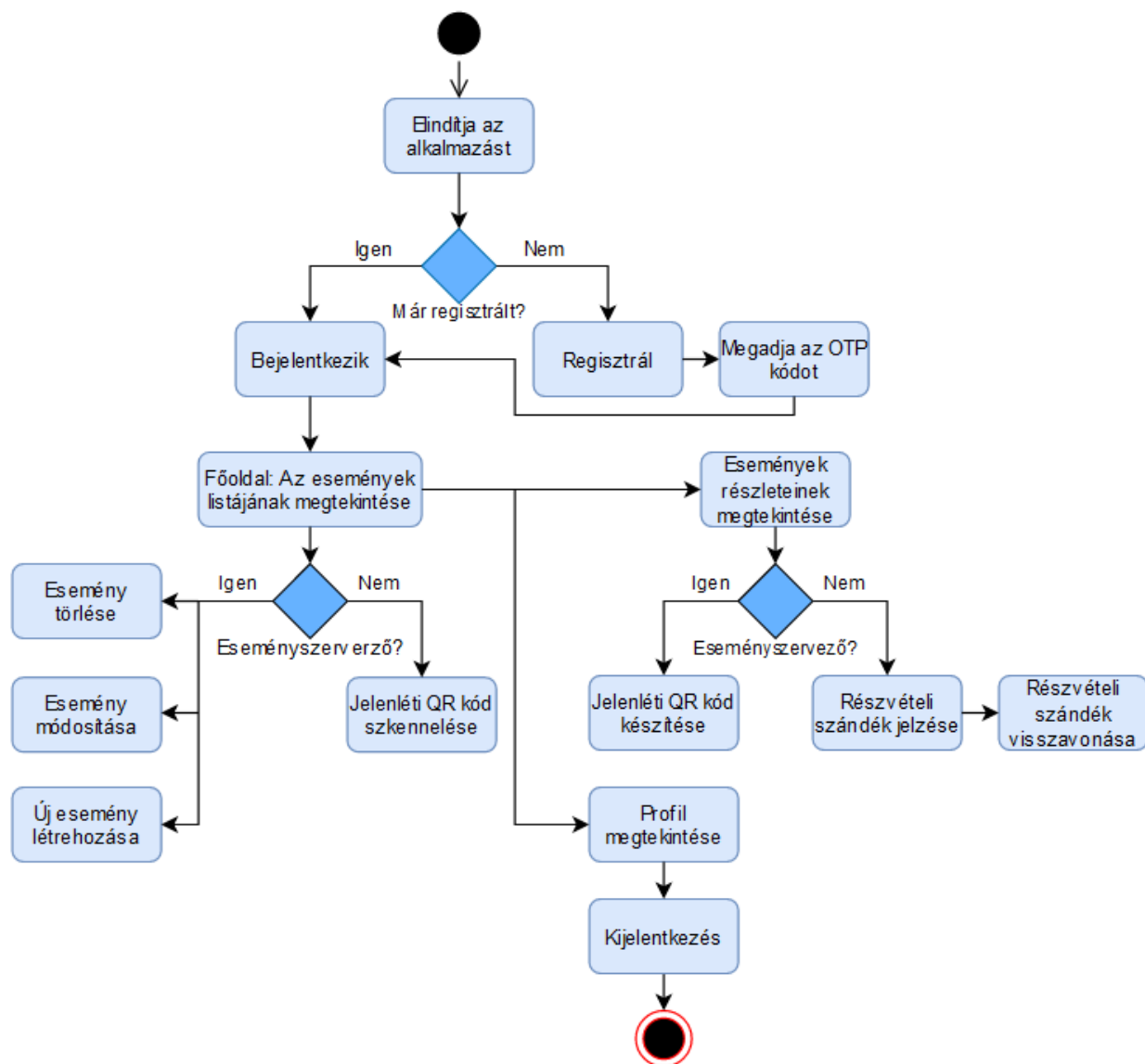
3.3. Felhasználói követelmények (kliens)

Ebben az alfejezetben találhatók meg azok az esetek, amelyekkel a felhasználó a rendszer használata közben találkozhat. Szó lesz a felhasználói követelményekről, és rendszerkövetelményekről, valamint bemutatom a fontosabb használati eseteket ábrákon (használati eset diagram, szekvenciadiagram) és részletes levezetésen keresztül. Az alkalmazás használata

közben 2 aktor jelenik meg, az a felhasználó, aki nincs regisztrálva és az, aki igen. A regisztrált felhasználónak 3 típusa van: résztvevő, eseményszervező és adminisztrátor. A felhasználó felhasználói szerepkörétől függ az, hogy milyen funkcionalitásokat ér el az alkalmazáson belül. A [8. ábrán](#) szemléltettem a felhasználó eseteket míg a [9. ábrán](#) a lehetséges aktivitásokat.



8. ábra: Használati eset diagram



9. ábra: Aktivitás diagram

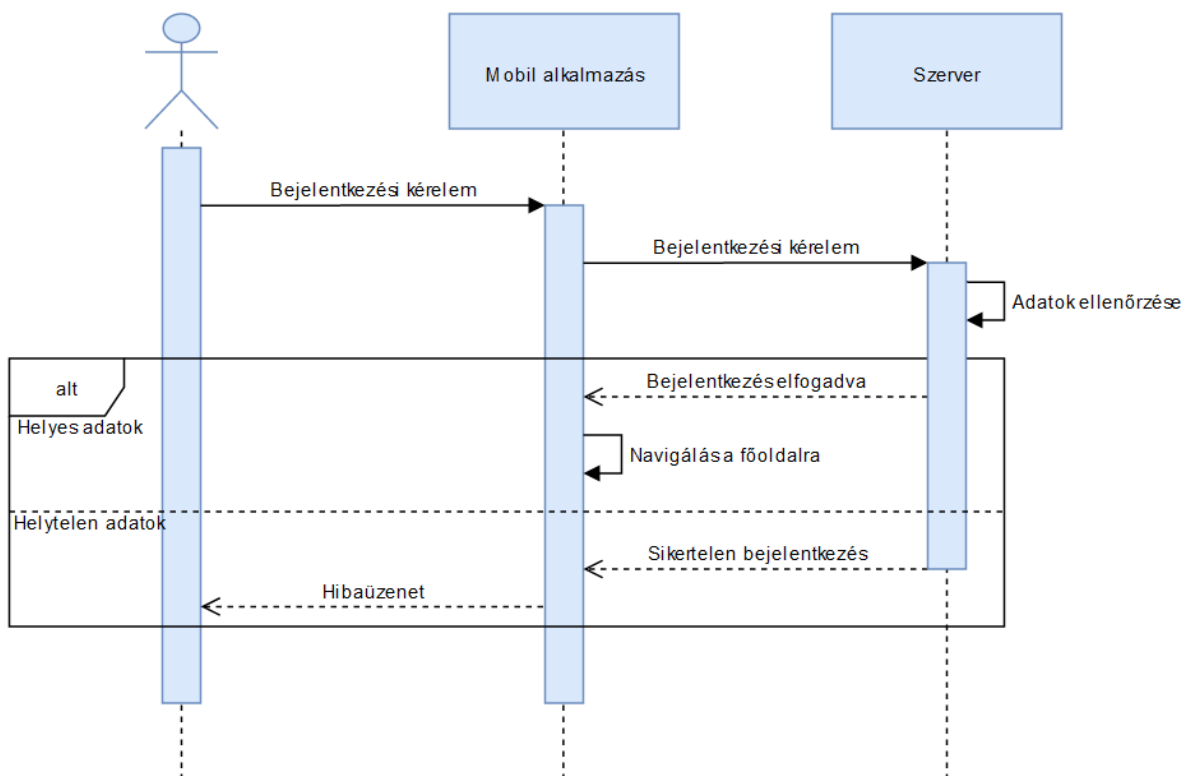
3.3.1. Fontosabb felhasználói esetek

- **Egy esemény részleteinek megtekintése**
 - Előfeltételek
 - PR1: a felhasználó be van jelentkezve (F2), amely folyamat pontosabban megfigyelhető a [10. ábra](#) szekvenciadiagramján
 - PR2: létezik legalább egy esemény
 - Folyamat 1 (nem regisztrált felhasználó)

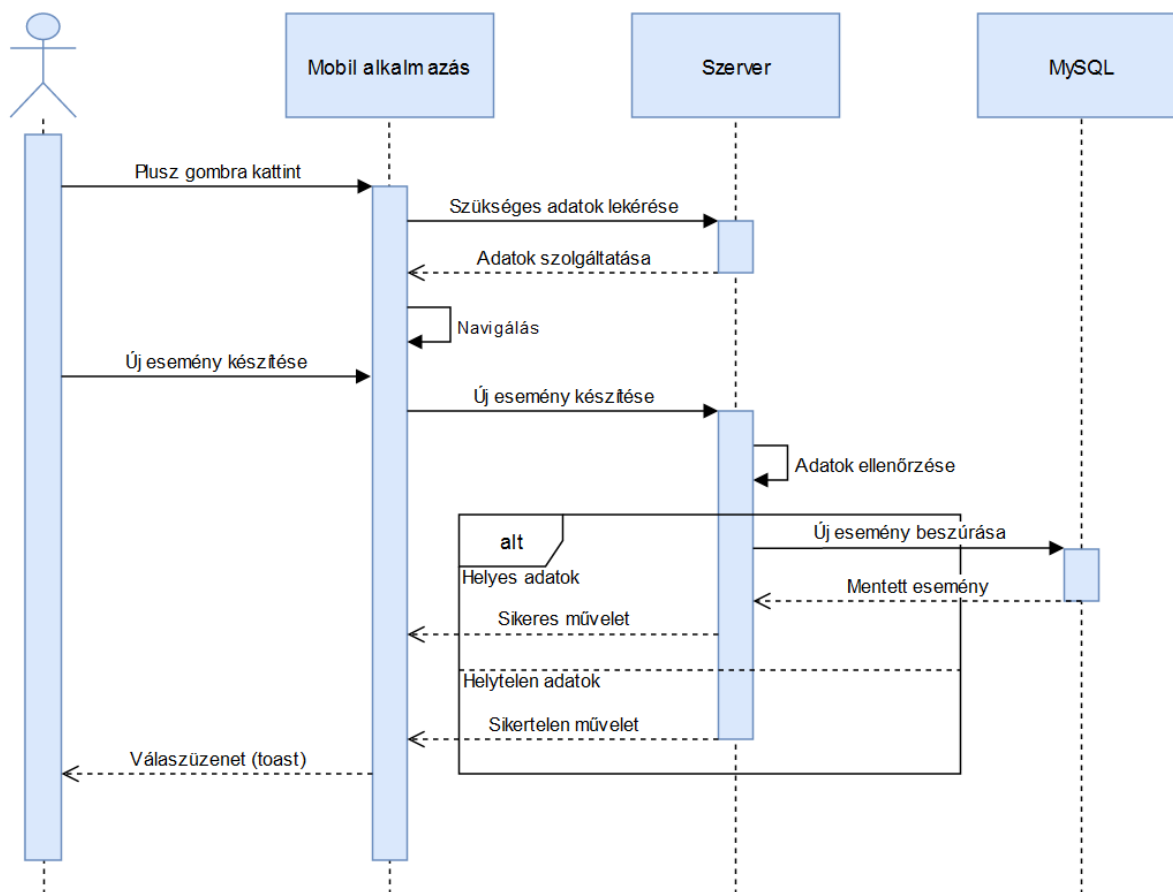
- F1F1: a felhasználó a kezdőoldalon a „Vendég” gombra kattint
- F1F2: a felhasználó a listát az ujjával mozgatva navigál az események között.
- F1F3: a felhasználó rákattint az eseményre, melynek részleteit meg szeretné tekinteni
- Folyamat 2 (regisztrált felhasználó)
 - F2F1: a felhasználó a kezdőoldalon a bejelentkezés gombra kattint
 - F2F2: a felhasználó bejelentkezik
 - F2F3: a felhasználó a listát az ujjával mozgatva navigál az események között
 - F2F4: a felhasználó rákattint az eseményre, melynek részleteit meg szeretné tekinteni
- Alternatív folyamat
 - F2AF1: ha a felhasználó be van már jelentkezve és nem látja a listát, akkor megnyomja a Főmenü gombot
 - F2AF2: a folyamat az F2F3-tól folytatódik
- Utófeltételek
 - PO1: megjelennek a kiválasztott esemény részletei
 - PO2: ha a felhasználó résztvevő szerepkörrel rendelkezik, akkor megjelenik a gomb, amellyel jelezheti részvételi szándékát
 - PO3: ha a felhasználó eseményszervező szerepkörrel rendelkezik, akkor megjelenik a gomb a QR kód elkészítéséhez
- **Jelenléti QR kód szkennelése**
 - Előfeltételek
 - PR1: a felhasználó be van jelentkezve
 - PR2: a felhasználó résztvevő szerepkörrel rendelkezik
 - Folyamat
 - F1: a felhasználó a kezdőoldalon a bejelentkezés gombra kattint
 - F2: a felhasználó bejelentkezik
 - F3: a felhasználó a QR menüre kattint
 - F4: a felhasználó beszkenneli a QR kódot
 - Alternatív folyamat

- AF1: ha a felhasználó be van már jelentkezve, de nem a főmenüben van, akkor egyszer visszanavigál oda
 - AF2: a folyamat az F3-tól folytatódik
- Utófeltétel
 - PO1: a rendszer elmenti az adatbázisba, hogy a felhasználó részt vett az eseményen
- **Új esemény létrehozása:** szekvenciadiagramja a [11. ábrán](#) látható
 - Előfeltételek
 - PR1: a felhasználó be van jelentkezve
 - PR2: a felhasználó eseményszervező szerepkörrel rendelkezik
 - Folyamat
 - F1: a felhasználó a kezdőoldalon a bejelentkezés gombra kattint
 - F2: a felhasználó bejelentkezik
 - F3: a felhasználó a + gombra kattint
 - F4: a felhasználó megadja a szükséges adatokat
 - F5: a felhasználó a „MENTÉS” gombra kattint
 - Alternatív folyamat
 - AF1: ha a felhasználó be van már jelentkezve, de nem a főmenüben van, akkor egyszer visszanavigál oda
 - AF2: a folyamat az F3-tól folytatódik
 - Utófeltétel
 - PO1: megjelenik a töltőképnyő
 - PO2: megjelenik egy üzenet, amely tájékoztatja a felhasználót, hogy az esemény létrehozása sikeres volt vagy sem
 - PO3: ha a művelet sikeres volt, akkor a rendszer elmenti az adatbázisba az új eseményt
- **Esemény törlése**
 - Előfeltételek
 - PR1: a felhasználó be van jelentkezve
 - PR2: a felhasználó eseményszervező szerepkörrel rendelkezik
 - Folyamat
 - F1: a felhasználó a kezdőoldalon a bejelentkezés gombra kattint
 - F2: a felhasználó bejelentkezik

- F3: a felhasználó az általa kiválasztott esemény kártyáján a kuka gombra kattint
- Alternatív folyamat
 - AF1: ha a felhasználó be van már jelentkezve, de nem a főmenüben van, akkor egyszer visszanavigál oda
 - AF2: a folyamat az F3-tól folytatódik
- Utófeltétel
 - PO1: ha a művelet sikeres volt, akkor az adatbázisból és a listából törlődik az esemény
 - PO2: ha a művelet sikertelen volt, akkor megjelenik egy üzenet, amely tájékoztatja a felhasználót



10. ábra: A bejelentkezés szekvenciadiagramja



11. ábra: Egy esemény elkészítésének szekvenciadiagramja

3.4. Rendszerkövetelmények (kliens)

a) Funkcionális követelmények:

Résztevő

- Alkalmazás indítása: Az alkalmazás indításakor a felhasználó egy kezdőképernyővel találja magát szemben, ahol kiválaszthatja, hogy regisztrálni vagy bejelentkezni szeretne.
- Regisztráció: A felhasználónak lehetősége van regisztrálni, ha megad minden szükséges adatot és a Regisztráció gombra kattint.
- Bejelentkezés: A felhasználó bejelentkezhet, ha beírja a regisztrálásnál megadott felhasználónév és jelszó párost.
- Események listájának megtekintése: A felhasználó megtekintheti az események listáját a főmenüben, ha bejelentkezik.

- Események részleteinek megtekintése: A felhasználó megtekintheti az esemény részleteit, ha a főmenüben rákattint az egyik listaelemre.
- Részvételi szándék jelzése: A felhasználó az esemény részletei alatt talál egy gombot, JELENTKEZÉS, amelyet megnyomva jelzi, hogy részt szeretne venni.
- Részvételi szándék visszamondása: Jelentkezés után megjelenik a LEMOND gomb, arra kattintva a felhasználó visszavonhatja a jelentkezését.
- Jelenlét regisztrálása QR kód segítségével: A felhasználó a főmenü menü sávjában, ha rákattint a QR ikonra, akkor beolvashatja a szervező által generált QR kódot, ami elmenti, mint jelenlévő.

Szervező

- Regisztráció: A felhasználónak lehetősége van regisztrálni, ha megad minden szükséges adatot és a Regisztráció gombra kattint. A rendszer automatikusan osztja ki a szervező szerepet, ha az adatbázisban szerepel az e-mail cím.
- Bejelentkezés: A felhasználó bejelentkezhet, ha beírja a regisztrálásnál megadott felhasználónév és jelszó párost.
- Események listájának megtekintése: A felhasználó megtekintheti az események listáját a főmenüben, ha bejelentkezik.
- Események részleteinek megtekintése: A felhasználó megtekintheti az esemény részleteit, ha a főmenüben rákattint az egyik listaelemre.
- Jelenlét regisztrálásához szükséges QR kód generálása: Az események részletei után lévő QR Készítése gombra kattintva egy új ablak ugrik fel, ami tartalmazza a szükséges QR kódot.

b) Nem funkcionális követelmények:

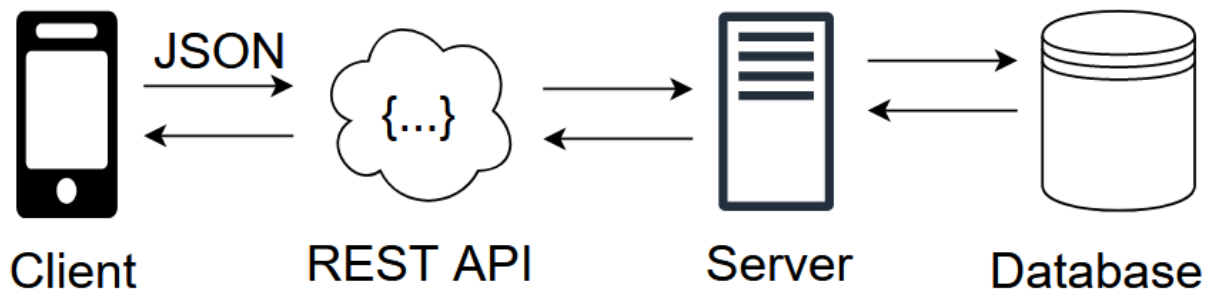
- Android 5.0+
- Minimum API szint: 21
- Tárhely: 67 MB
- Hálózati hozzáférés

Legalább a fenti követelményeket teljesíteni kell ahhoz, hogy az alkalmazás elinduljon és annak minden funkcionalitását használni lehessen Androidon.

4. Tervezés

A rendszer három fontosabb komponensből tevődik össze ([12. ábra](#)), amelyek a következők: adatbázis, szerveralkalmazás (backend), kliensalkalmazás (frontend).

Az adatbázis felel értelemszerűen az adatok hosszú távú eltárolásáért. A szerver a kliens által generált REST API kéréseket szolgálja ki. Ezek legfőbbje CRUD műveletek, mint az adatok írása és olvasása, adatok törlése és módosítása. Ugyanakkor a szerver felel továbbá a bejelentkezésért, a tokenek elkészítéséért és ellenőrzéséért, bizonyos adatok titkosításáért. A kliensalkalmazás az a mobilapplikáció, amelyet a felhasználó letölt a telefonjára és ott használja. Ez határozza meg, hogy mit lát (UI) a felhasználó és milyen lesz a felhasználói élménye (UX). A háttérben a szerver által nyújtott végpontokat használja minden olyan műveletnél, ahol az adatbázis elérése szükséges vagy olyan szolgáltatásokra van szüksége, mint e-mailek küldése.

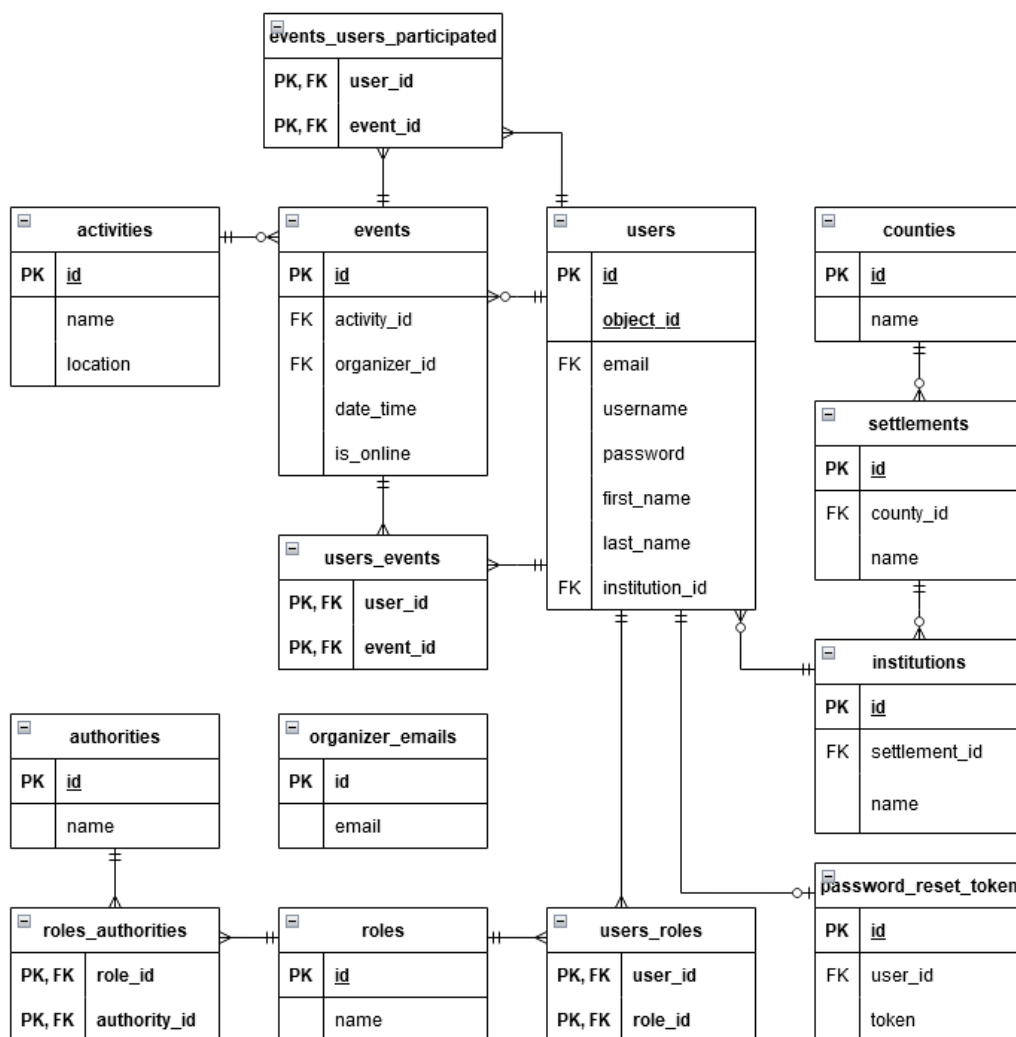


12. ábra: A teljes rendszer architektúrája

4.1. Adatbázis

A projekt megvalósítására a MySQL adatbázist választottam, mert az egy biztonságos és megbízható relációs adatbázis, mert a MySQL-ben készült adatbázisok könnyen skálázhatók, és viszonylag nagy adathalmaz esetén is hatékonyan végzi el a CRUD műveleteket. Az egyik legszélesebb körben használt adatbázis a világon, így könnyű vele kapcsolatban segítséget kapni az interneten és teljesen ingyenes, ami szintén fontos [\[17\]](#).

A [13. ábrán](#) a projekthez tartozó táblák és az azok közötti kapcsolatok láthatóak (a diagrams.net weboldal segítségével létrehozva). Mint megfigyelhető, a középpontban a *users* tábla áll, vagyis az egészet összekötő entitás az felhasználó és a hozzá kapcsolódó adatok.



13. ábra: Egyed kapcsolat diagram

4.1. Szerver

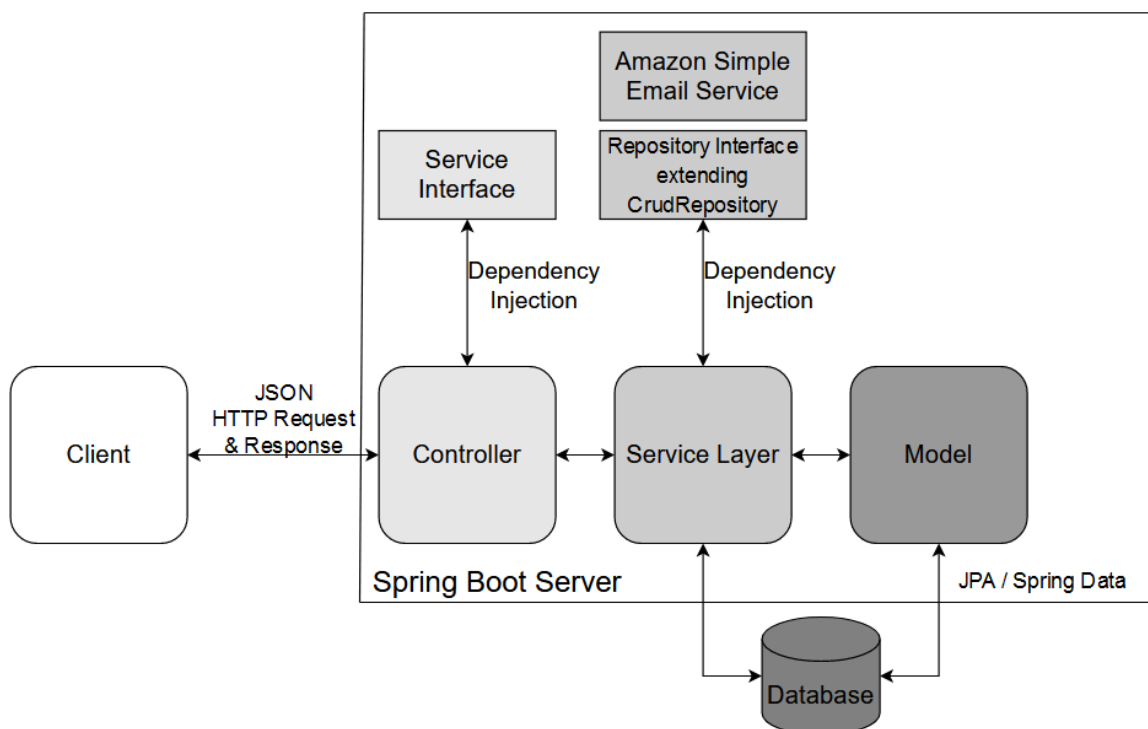
A szerver oldal megvalósítására a Spring Boot keretrendszert választottam, mert csökkenti az ismétlődő (boilerplate) kódrészleteket, kompatibilis több szervlet tárolóval is, mint az Apache Tomcat, Jetty, és sok időt meg lehet takarítani, mert nem kell a konfigurációs beállításokat

manuálisan végezni. Van saját memória-alapú adatbázisa (H2). Összegezve, nagyon jól optimalizált mikroszolgáltatások és web API-k létrehozására.

A [14. ábrán](#) látható a szerveralkalmazásom architektúrája. A szerverem a kéréseket JSON formában tudja fogadni a vezérlő rétegben meghatározott végpontok segítségével. Az érkező adat áthalad a különböző rétegeken. Az adat haladhat a vezérlőtől az adatbázis fele és az adatbázistól a vezérlő fele is. A következő rétegeken halad át az adat: vezérlő (controller), szolgáltatás (service), repository és végül az adatbázis. A különböző rétegeknek megvannak a pontos feladataik, amelyeket betartva könnyebb a fejlesztés. **Vezérlő:** A vezérlő egy protokoll interfész, amely meghatározza a különböző végpontokat, amelyeket a felhasználók használhatnak. Általában nem tartalmaz semmilyen üzleti logikát.

Szolgáltatás: A szolgáltatás réteg tartalmazza az üzleti logikát. Ez határozza meg, hogy a kapott adaton milyen műveletet hajtunk végre, és hogy milyen adatot térítünk vissza. Ez a réteg használja az Amazon Simple Email Service (SES) szolgáltatását e-mailek küldésére.

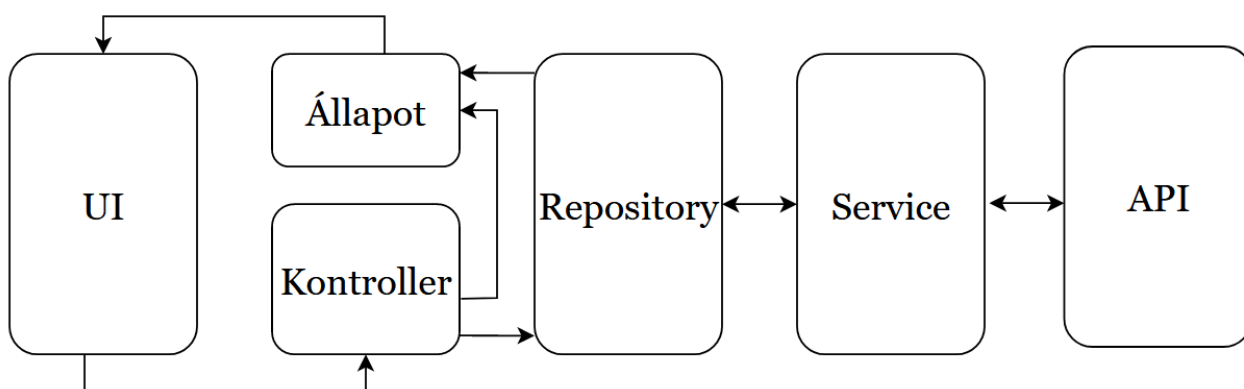
Repository: A repository réteg kezeli az adatbázison elvégezhető műveleteket, mint az adatok elmentése, kiolvasása, keresése. Egy interfészen keresztül érhetjük el ezeket a műveleteket.



14. ábra: A szerver architektúrája

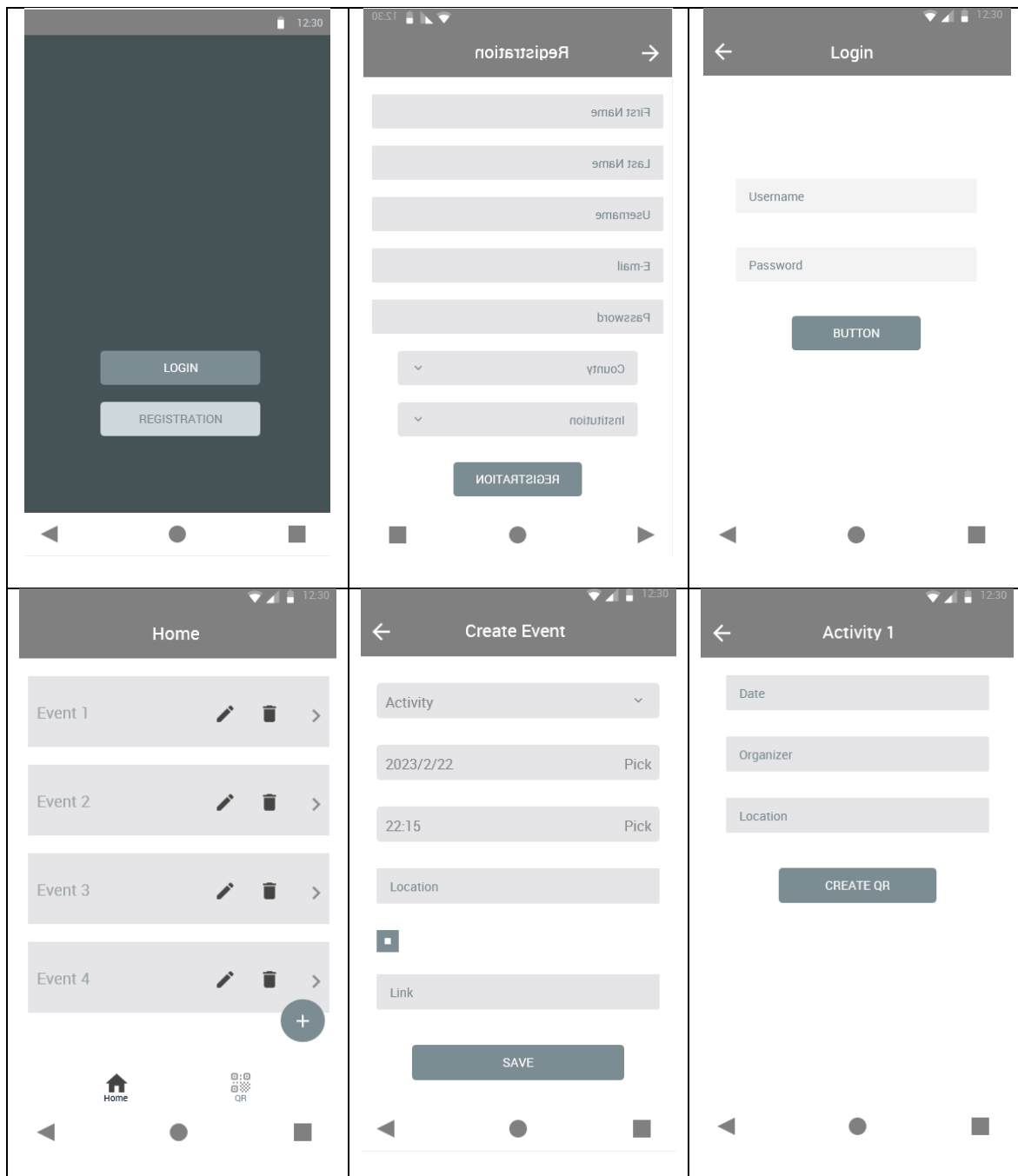
4.2. Kliensalkalmazás

A kliensprogram architektúrájának gerincét a Riverpod alkotja. A Riverpod egy olyan állapotmenedzselő csomag, amely a Provider csomag átdolgozásából született. Annak ellenére, hogy milyen egyszerű, viszonylag bonyolult feladatok is megvalósíthatók vele. Segítségével olyan vezérlő osztályokat írhatunk, amelyek aztán teljesen függetlenek a widget fától, így el tudjuk különíteni egymástól a felhasználói felületet megvalósító widgeteket az üzleti logikától. Ezzel a megközelítéssel egy tisztább kódbázist kapunk. A [15. ábrán](#) látható az architektúra vázlata.



15. ábra: A kliens architektúrája

A felhasználói felület megtervezésében a Fluid UI segített, amellyel könnyen lehet drótvázakat és prototípusokat készíteni. A kliensalkalmazás drótváza a [16. ábrán](#) látható.



16. ábra: A kliens drótváza

5. Kivitelezés

5.1. Szerveralkalmazás

5.1.1. Előfeltételek

Ahhoz, hogy az API el tudja tárolni az adatokat, szüksége van egy adatbázisra, ami ebben az esetben egy MySQL adatbázis. Az adatbázis kapcsolatot az application.properties fájlban kell meghatározni. Ehhez a következő három paramétert kell megadni. Az én esetemben az adatbázis a saját gépemem fut, ezért a localhost (sajátgép) cím van megadva. A MySQL alapértelmezetten a 3306-os portot használja.

- spring.datasource.url: A link, amelyen keresztül az alkalmazás eléri az adatbázist.
- spring.datasource.username: Az adatbázis telepítésénél megadott, az hitelesítéshez szükséges felhasználónév.
- spring.datasource.password: Az adatbázis telepítésénél megadott, a hitelesítéshez szükséges jelszó.

```
spring.datasource.username=root  
spring.datasource.password=theSIS-2022_  
spring.datasource.url=jdbc:mysql://localhost:3306/open_days_database
```

1. kódrészlet: Az adatbáziskapcsolat létrehozása
az application.properties fájlban

5.1.2. A szerver elindítása

A projekt elindítása több módon is történhet. Mivel az IntelliJ IDEA alapértelmezetten tartalmazza az Apache Tomcat szerveret, ezért a fejlesztés nagy részében elég, ha a projektet megnyitjuk ebben a fejlesztői környezetben, majd itt futtatjuk. A legelső alkalommal, hogy az IntelliJ automatikusan konfigurálja a projektet, közvetlen a main függvényt kell futtatni. Ezt megtehetjük a zöld nyilak megnyomásával az „OpenDaysBackendApplication” fájlban.

```

@SpringBootApplication
public class OpenDaysBackendApplication {
    public static void main(String[] args) {
        SpringApplication.run(OpenDaysBackendApplication.class, args);
    }
}

```

2. kódrészlet: A szerver „main” függvénye

Szerverem esetében a végpontokat a localhost címen keresztül érhetjük el. A regisztráció linkje például a „http://localhost:8081/open-days/users” link. Az API által használt portot az application.properties fájlban beállított server.port tulajdonsággal konfigurálhatjuk. A könnyebb hordozhatóság érdekében a projektből készíthető egy WAR fájl is, amely aztán könnyen átvihető egy másik eszközre. Fontos, hogy ebben az esetben az Apache Tomcat vagy más hasonló programot magunknak kell telepíteni az rendszerünkre. A WAR fájl elkészítéséhez a főosztálynak ki kell bővítenie a „SpringBootServletInitializer” osztályt, majd felül kell írnia a „configure (SpringApplicationBuilder builder)” metódust.

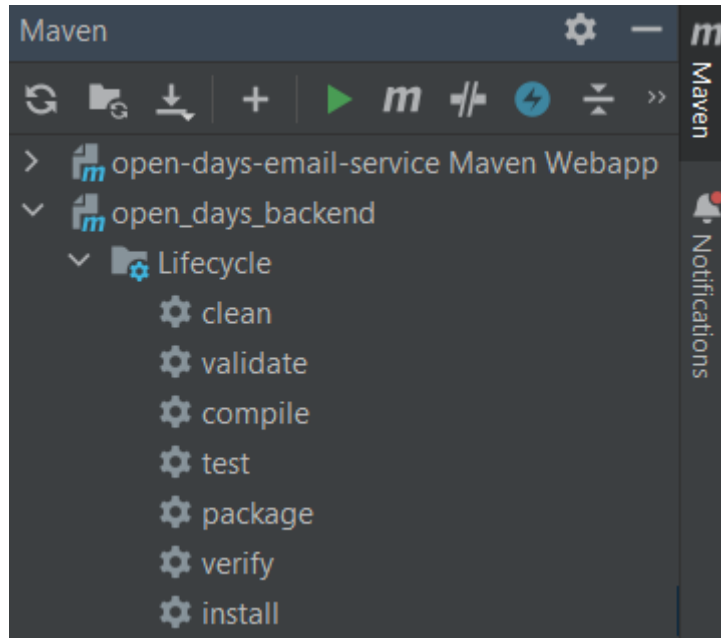
```

@SpringBootApplication
public class OpenDaysBackendApplication {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(OpenDaysBackendApplication.class);
    }
}

```

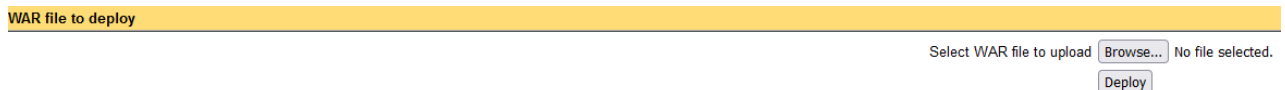
3. kódrészlet: A WAR fájl automatikus elkészítéséhez szükséges osztály

Továbbá a pom.xml fájlban meg kell határozni a csomagolás formátumát a „<packaging>war</packaging>” formában, és importálni kell a „spring-boot-starter-tomcat” tárolót. Ezután az IDEA Maven fülét kiválasztva először futtatjuk a „clean” majd az „install” ([17. ábra](#)) parancsot, ami pedig generál egy .war fájlt a projekt „target” nevű mappájában.



17. ábra: A WAR fájl elkészítése, parancsok

Az így létrehozott fájl egyszerűen bemásolható a Tomcat „webapps” nevű mappájába, vagy hozzáadható a Tomcat webes felületét használva is ([18. ábra](#)), mely a „http://localhost:8080/manager/html” linken keresztül érhető el, miután a szerveret elindítottuk a rendszerünkön. Itt először ki kell tallózni a fájlt, majd bevetni a „Deploy” gomb megnyomásával.



18. ábra: A WAR fájl bevetése a Tomcat webes felületén

5.1.3. Entitás osztályok

Az entitás osztályok az adatbázisban szereplő táblákat, míg az osztályok attribútumai pedig a táblák oszlopait reprezentálják. Minden entitás osztályt az „@Entity” annotációval kell ellátni, hogy a JPA tudja értelmezni. Ha azt szeretnénk, hogy az adatbázisban megjelenő tábla neve ne ugyanaz legyen, mint az osztály neve, akkor használhatjuk az „@Table” annotációt, ahol a „name” argumentummal meghatározhatjuk a tábla nevét.

Az entitás osztályoknak szükségük van egy azonosító attribútumra, amelyet az „@Id” annotációval lehet meghatározni. Ez az én esetemben egy Long típusú érték, amelyet ellátok egy „@GeneratedValue” annotációval. Ez megmondja az adatbázisnak, hogy minden új rekordnak az azonosító értékét eggyel növelje az utolsóhoz képest, így garantálva az egyedi értéke.

```
@Id
@GeneratedValue
private long id;

@Column(nullable = false, length = 50)
private String name;

@OneToMany(mappedBy = "activity")
private Set<EventEntity> events;
```

4. kódrészlet: Egy entitás osztály attribútuma

A többi attribútumot elláthatjuk az „@Column” annotációval, amely segítségével megadhatjuk az oszlopok egyéb tulajdonságait, mint a nevét, a hosszát, vagy azt, hogy lehet-e üres (null) az értéke.

A táblák közötti kapcsolatokat is az entitás osztályokban határozzuk meg. Itt be kell állítanunk, hogy milyen kapcsolatot szeretnénk két tábla között, és azt, hogy ebben a kapcsolatban milyen oszlopok vesznek részt. A „@ManyToMany” annotáció több a többhöz kapcsolatot hoz létre, míg a „@JoinTable” annotáció határozza meg az oszlopokat, amely szerint a táblák kapcsolódnak. Mindkét táblában be kell állítani a kapcsolatot, hogy megfelelően működjön. Az alábbi képek például az „activities” és „events” táblák közti egy a többhöz kapcsolat létrehozását ábrázolják [\[18\]](#) [\[28\]](#) [\[29\]](#).

```
@OneToMany(mappedBy = "activity")
private Set<EventEntity> events;
```

5. kódrészlet: Az egy a többhöz kapcsolat

```
@ManyToOne
@JoinColumn(name = "activity_id", nullable = false)
private ActivityEntity activity;
```

6. kódrészlet: A több az egyhez kapcsolat

5.1.4. Vezérlő osztályok

A szerveralkalmazásban a kapcsolódási pontot a vezérlő osztályokban meghatározott metódusok jelentik. Minden ilyen metódus el van látva egy annotációval, amely meghatározza, hogy a metódus milyen típusú műveletet végez el. A fő annotációk a „@PostMapping”, „@GetMapping”, „@PutMapping”, „@DeleteMapping”, amelyek az azonos nevű HTTP metódusokból jönnek. Ezek segítségével bővíthető az útvonal, amellyel elérjük ezt a végpontot. Ez lehet egy statikus bővítmény vagy egy paraméter is.

A metódusok paramétereit is annotációkkal látjuk el. Ha a linkben megadott paramétert szeretnénk elérni, akkor a „@PathVariable” annotációt, ha a kérés törzsét szeretnénk elérni, akkor a „@RequestBody” annotációt kell használni. Ennek a JSON az alapértelmezett formátuma, de ez megváltoztatható például XML-re [\[18\]](#) [\[28\]](#) [\[29\]](#).

```
@PostMapping(path = "/password-reset")
public OperationStatusModel resetPassword(@RequestBody PasswordResetModel
passwordResetPayload) {
    // Code block
}
```

7. kódrészlet: Egy @PostMapping végpont elkészítése

```
@DeleteMapping(path =("/{publicId}")
public OperationStatusModel deleteUser(@PathVariable String publicId){
    // Code block
}
```

8. kódrészlet: Egy „@DeleteMapping” végpont elkészítése

5.1.5. DTO osztályok

A vezérlő osztályokban lévő függvények nem mindig pontosan azokat az adatokat vagy abban a formában kapják meg a klienstől, melyet tovább kell küldjenek a szolgáltatás réteg fele. Hogy a szolgáltató rétegben lévő függvények a megfelelő adatokat tároló objektumokat kapják meg, DTO (Data Transfer Object) objektumokat készítünk a vezérlő rétegben. Először bemásoljuk a klienstől kapott adatokat ezekbe az objektumokba, majd kibővítjük vagy módosítjuk azokat, ahol szükség van rá. A másolást a „BeanUtils” osztály „copyProperties” függvényével végzem el, ahol csak meg kell adni első paraméterként az objektumot, amit másolni szeretnénk, majd az objektumot, ahova másolni szeretnénk. A függvény automatikusan átmásolja az azonos névvel rendelkező attribútumokat az egyik objektumból a másikba. Ezek az osztályok csak a konverzióra vannak használva a két réteg között [18] [28] [29].

5.1.6. Szolgáltatás osztályok

A vezérlő osztályokban nincs hozzáférésünk az adatbázissal dolgozó függvényekhez, és egy jól megvalósított architektúrában üzleti logikát sem tárolunk ebben az osztályban. Hogy ezt kiküszöböljük, meg kell hívnunk a szolgáltató réteg függvényeit. Ezt úgy tehetjük meg, hogy az adott vezérlőhöz tartozó szolgáltató interfészét befecskendezzük a vezérlő osztályba, amelyen keresztül elérhetjük a megfelelő függvényeket. Ezt a műveletet a Spring Boot-ban az „@Autowired” annotációval valósíthatjuk meg.

Minden ilyen interfészhez tartozik egy konkrét osztály, amelyben megadjuk az interfészben deklarált függvények implementációját. Ezekben az osztályokban adjuk meg az üzleti logikát és ide fecskendezzük be a repository réteg megfelelő interfészeit. Ezek az interfészek deklarálják a függvényeket, amelyek közvetlenül az adatbázissal dolgoznak. Adatokat mentenek el és módosítanak ott, adatokat olvasnak ki vagy törölnek onnan [18] [28] [29].

```
@Autowired
UserService userService
```

9. kódrészlet: A befecskendezés megvalósítása Spring Boot-ban

5.1.7. Repository osztályok

Ha egy repository interfésszel kibővítjük a „CrudRepository” vagy egy másik Repository típusú interfészt, akkor hozzáférünk az adatbázissal való kommunikációhoz szükséges alapvető függvényekhez. Ugyanakkor létrehozhatunk saját függvényeket is, csak az elnevezésre kell odafigyelnünk. Ha például keresni szeretnénk az adatbázisban egy attribútum alapján, akkor annak követnie kell a findBy + attribútum név formátumot. Pl.: findByEmail. Ezeknek a függvényeknek a Spring Boot automatikusan kigenerálja az implementációit [\[18\]](#) [\[28\]](#) [\[29\]](#).

```
@Repository
public interface UserRepository extends PagingAndSortingRepository<UserEntity, Long>
{
    // Code block
}
```

10. kódrészlet: Egy repository osztály saját e-mail alapján kereső függvénnyel

5.1.8. Biztonság

A projektben használok a Spring Security keretrendszert, ezért automatikusan minden végpont el van zárva a publikus hívásoktól. Ahhoz, hogy egy végpont mégis publikus legyen, külön konfigurálnom kell erre azt. Ezt a „WebSecurity” osztályban tettem meg. Ez az osztály el van látva a „@EnableWebSecurity” annotációval és kibővíti a „WebSecurityConfigurationAdaptert”, így a „configure” metódusában egyenként meghatározhattam, hogy mely végpontok legyenek publikusak. Erre szükség is van, mert például, egy regisztrációnál vagy jelszó visszaállításnál nem várhatom el a felhasználótól, hogy az rendelkezzen a tokennel, ami biztosítja a titkosított végpontok elérését [\[29\]](#).

```
.antMatchers(HttpMethod.POST, SecurityConstants.SIGN_UP_URL)
.permitAll()
```

11. kódrészlet: Egy végpont publikussá tétele

Az összes többi végpont elérhetetlen minden olyan hívás számára, amely nem rendelkezik egy érvényes tokennel. Ezt a tokenet bejelentkezéskor generálja ki a szerver. A kliens ezt a tokenet kell a kérések fejlécébe belerakja, hogy képes legyen a végpontokkal kommunikálni. A tokeneknek van egy lejáratási idejük is, ami az én esetemben 30 nap. Ez azt jelenti, hogy minden token 30 napig használható a szerverrel való kommunikáció felhatalmazására [29].

```
String token = Jwts.builder()
    .setSubject(username)
    .setExpiration(new Date(System.currentTimeMillis() +
        SecurityConstants.EXPIRATION_TIME))
    .signWith(SignatureAlgorithm.HS512,
        SecurityConstants.getJwtSecretKey())
    .compact();
```

12. kódrészlet: A hitelesítéshez szükséges token kigenerálása

Minden felhasználóhoz tartozik egy felhasználói szerepkör, ezeket a szerepköröket használva az érvényes tokennel rendelkező felhasználók kéréseit is blokkolja a Spring Security, ha egy adott művelethez egy speciális szerepkörre van szükség. Például bizonyos adatok törlését nem végezheti el minden felhasználó, csak a rendszergazdai joggal bírók, vagy az érzékeny felhasználói adatokat csak az a felhasználó kérheti le, akihez tartoznak.

Egy adott végpont ilyen jellegű szűrését megtehetem a „@Secured”, „@PreAuthorize” vagy „@PostAuthorize” annotációk használatával az adott végpontot generáló függvényen, mert a korábban említett „WebSecurity” osztályhoz hozzáadtam az „@EnableGlobalMethodSecurity” annotációt [29].

```
@PostAuthorize("hasRole('ADMIN') or returnObject.publicId ==
    principal.publicId")
@GetMapping(path =("/{publicId}")
public UserResponseModel getUser(@PathVariable String publicId) {
    // Code block
}
```

13. kódrészlet: Egy végpont korlátozása a szerepkör alapján

5.2. Kliensalkalmazás

5.2.1. Felhasználói felület

A felhasználói felületért a „screens” mappában található osztályok felelnek, amelyek olyan Flutter SDK-hoz tartozó osztályokat bővítenek ki, mint a „StatelessWidget”, és „StatefulWidget” vagy olyan a Riverpodhoz tartozókat, mint a „ConsumerWidget” és a „ConsumerStatefulWidget”. Ezek az osztályok felelősek mindenért, amit a felhasználó lát a képernyőn. Ezekhez az osztályokhoz különféle állapotok tartoznak, amik ha megváltoznak, akkor a Flutter automatikusan frissíti a képernyőt is. A felhasználó a különféle UI elemek segítségével tud parancsokat és adatokat küldeni az alkalmazásnak. Például, ha a felhasználó megnyom egy gombot, akkor az meghívja a megfelelő parancs függvényét a vezérlő osztályból, ami megváltoztathatja az osztály állapotát, az pedig következképpen megváltoztatja a képernyő tartalmát.

```
class EventScanner extends ConsumerStatefulWidget {  
  const EventScanner({Key? key}) : super(key: key);  
  
  @override  
  _EventScannerState createState() => _EventScannerState();  
}  
  
class _EventScannerState extends ConsumerState<EventScanner> {  
  // Code block  
}
```

14. kódrészlet: Az EventScanner osztály, amelyik kibővíti a „ConsumerStatefulWidget” osztályt

5.2.2. Vezérlők

A vezérlő osztályok tartalmazzák az állapotokat és az azokon dolgozó függvényeket. Bármilyen bemenet érkezik a felhasználtól, amely valamilyen logika futtatását vagy internetes kommunikációt igényel, akkor ezek a függvények hívódnak meg. Ezek aztán közvetlen módosítják az állapotváltozókat.

Annak érdekében, hogy minden widget osztály hozzáférjen a megfelelő vezérlőhöz, azokat függőségi befecskendezéssel osztom meg a szükséges osztályokban. Ezt szintén a Riverpod teszi lehetővé, mert a Flutter SDK-ban nincs erre lehetőség.

Az állapotokat speciális szolgáltató objektumokban tárolom el, mint a Provider, amely bármilyen objektumot el tud tárolni, vagy a StateProvider, amely egyszerűbb típusok eltárolására használható, mint egy String vagy bool.

Ezeket a szolgáltató objektumokat majd a „ref” objektumon keresztül érem el, amelyet szintén a Riverpod biztosít. A „read” függvénnyel kiolvasom az értéket, míg ha a „watch” függvényt használom, akkor a Flutter automatikusan frissíti a képernyőt minden változásnál.

```
_eventScannerController = ref.read(eventScannerControllerProvider);
```

15. kódrészlet: Egy szolgáltató elérése a „ref” segítségével

5.2.3. Repository

A repository osztályok megakadályozzák a vezérlő osztályok közvetlen hozzáférését az adatbázishoz és az API hívásokhoz, így a rendszer könnyebben tesztelhető. Az itt használt DTO objektumokkal elérhetjük, hogy a vezérlők már csak a számukra fontos adatot kapják meg. Ez a réteg segít ideiglenesen eltárolni az adatokat, amik aztán használhatók, akkor is amikor nincs internet kapcsolat.

A repositoryk is függőség befecskendezéssel vannak elérhető téve más osztályok számára.

```
final baseRepositoryProvider = Provider((_) => BaseRepository());
```

16. kódrészlet: Szolgáltató készítése egy repository osztályból

5.2.4. API hívások

A „services” mappában vannak azok a fájlok, amelyek az API hívásokért felelős aszinkron függvényeket tartalmazzák. Flutterben a legegyszerűbb módját annak, hogy az interneten keresztül

adatokat kapjunk, a http csomag biztosítja. A HTTP függvények, mint a „get”, „post”, „delete” vagy „put” egy Future értéket térítenek vissza, amely egy „Response” objektumot tartalmaz. Ezt az értéket át kell alakítani egy Dart objektummá, hogy könnyebben tudjunk dolgozni vele.

Azt a folyamatot, amikor egy objektumot adatfolyammá alakítunk, sorosításnak nevezzük. Lényege, hogy adatfolyamként könnyebben küldhetjük tovább az interneten vagy tárolhatjuk el egy adatbázisban. A sorosítás megszüntetése az előbb említett művelet ellentéte, amikor az adatfolyamot alakítjuk át egy objektummá.

Ha a válasz a 200-as kódot tartalmazza, akkor az adatfolyamot átalakítom a szükséges objektummá, máskülönben egy hibaüzenetet küldök válaszként.

```
if (response.statusCode == 200) {  
  Iterable decodedResponse = json.decode(response.body);  
  return decodedResponse.map((e) => Institution.fromJson(e)).toList();  
} else {  
  throw Exception('Failed to get county names');  
}
```

17. kódrészlet: Egy HTTP művelet eredményének kezelése

5.2.5. Az alkalmazás bemutatása

Az alkalmazás elindítása után a felhasználó egy kezdőképernyővel találja szembe magát ([19. ábra](#)), ahol választhaja azt, hogy bejelentkezik, regisztrál vagy belép vendégként. Ha már egyszer bejelentkezett, akkor ez az oldal nem jelenik meg, egyből a főoldalt fogja látni a töltőképernyő után.

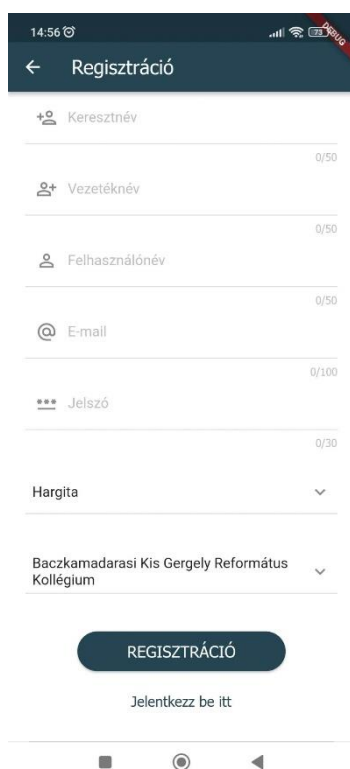


19. ábra: Kezdőoldal

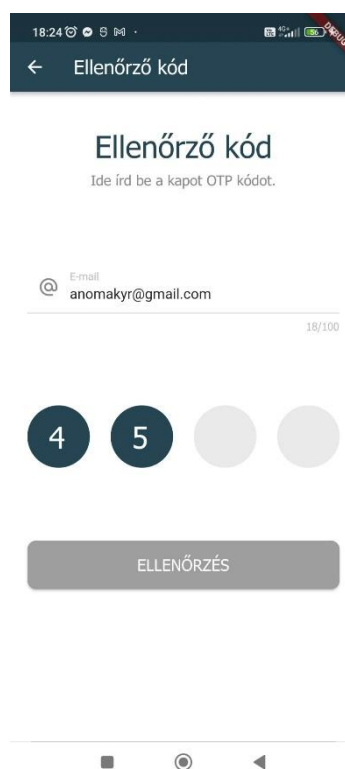
Regisztráció:

A felhasználó ezt az oldalt (20. ábra) a kezdőképernyőről érheti el, ha rákattint a regisztráció gombra. Regisztrációnál a felhasználónak meg kell adnia a személyes adatait, mint a keresztnév és vezetéknév. Választania kell egy még nem foglalt és legalább 3 karakter hosszú felhasználónevet. Szükséges még, hogy megadjon egy érvényes e-mail címet, ahova be tud jelentkezni, mert sikeres regisztráció esetén kapni fog egy e-mailt. Meg kell még adnia a jelszót, amely legalább 8 karakter hosszú. Legvégén pedig ki kell választania a megyét illetve a tanintézményt, ahol tanul.

Ha valamit elront vagy elfelejt megadni, akkor a regisztráció sikertelen lesz, az interfészen pedig megjelenik a megfelelő hibaüzenet. A szövegdobozok sarkaiban egy számláló jelzi az egyes helyeken megadható maximális karakterhosszt, illetve azt, hogy eddig hány karaktert írt be a felhasználó. A regisztráció utáni oldalon (21. ábra) az e-mailben kapott kódot kell megadni.



20. ábra: Regisztrációs oldal



21. ábra: Az ellenőrző (OTP) kód megadása

Bejelentkezés:

A felhasználó ezt az oldalt ([22. ábra](#)) a kezdőképernyőről érheti el, ha rákattint a bejelentkezés gombra. Itt a felhasználónak be kell írnia a regisztrációnál megadott felhasználónevet és jelszót. Fontos, ha a felhasználó nem erősítette meg regisztrációs szándékát az e-mailben kapott OTP kód segítségével, akkor a bejelentkezés nem lesz sikeres.

Ha a felhasználó nem ad meg, vagy rosszul ad meg valamilyen adatot, esetleg a felhasználónév és jelszó nem egyezik, akkor a bejelentkezés sikertelen lesz, és a képernyőn megjelenik a megfelelő hibaüzenet.

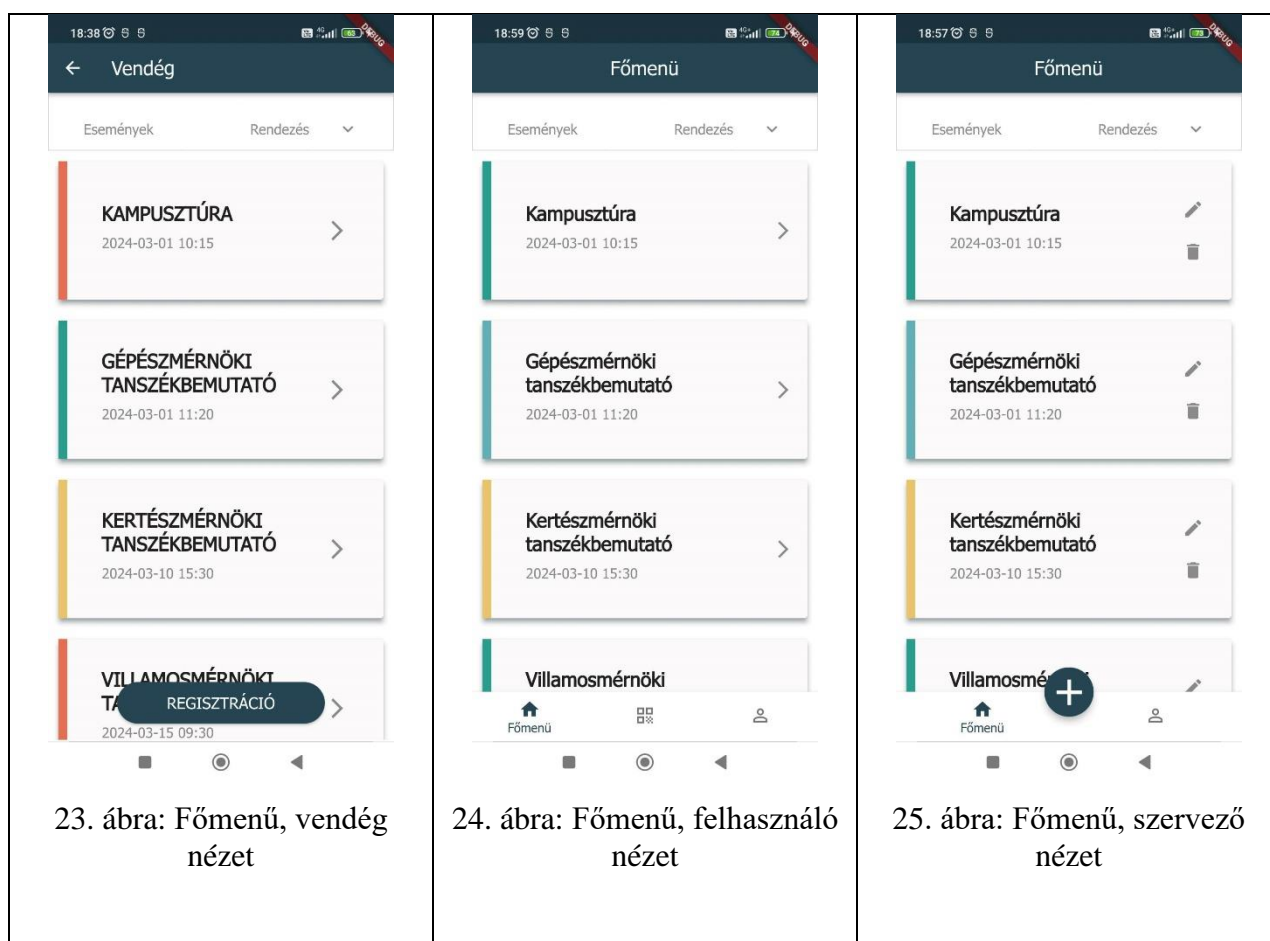


22. ábra: Bejelentkezési oldal

Főmenü:

A főmenü 3 nézetben található meg az alkalmazásban. Ha valaki vendégként (23. ábra) vagy felhasználóként (24. ábra) lép be, akkor minden eseményt lát, ami még nem járt le, ha valaki szervezőként (25. ábra) lép be, akkor minden eseményt lát, amit ő szervez a dátumtól függetlenül. Minden listaelemen látható az esemény neve és a dátum, amikor megszervezésre kerül.

Szervező: a szervezők a főmenüben találják meg azokat a gombokat, amelyekkel törölhetnek egy eseményt és elérhetik az új esemény készítése és meglévő esemény módosítása oldalakat.



Egy esemény részletei:

Ezt az oldalt úgy érheti el a felhasználó, hogy rákattint az egyik listaelemre a főoldalon lévő események listájából. Ez az oldal is változik annak függvényében, hogy valaki vendégként (26. ábra), felhasználóként (27. ábra) vagy szervezőként (28. ábra) van bejelentkezve.

Itt a felhasználó megtekintheti az esemény részleteit, mint a szervező neve, dátum, időpont és helyszín. A bejelentkezett felhasználó itt tud felíratozni az eseményre vagy leíratkozni arról a megfelelő gombra kattintva.

Szervező: A szervező ranggal rendelkező felhasználó itt láthatja a QR készítése gombot, amelyre ha rákattint, akkor megjelenik a jelenlét írásához szükséges QR kód (29. ábra). Ezt beolvasva könnyedén elkészíthető az eseményen résztvevők listája.



26. ábra: Esemény részletei, vendég nézet



27. ábra: Esemény részletei, felhasználó nézet



28. ábra: Esemény részletei, szervező nézet

A jelenléti QR kód beolvasása

A felhasználó a főmenüből a menüsávban lévő QR ikonra kattintva megnyitja a kamerát ([30. ábra](#)). Ha ennek segítségével beolvas egy szervező által generált QR kódot, akkor jelenléte az adott eseményre automatikusan elmentésre kerül az adatbázisban.



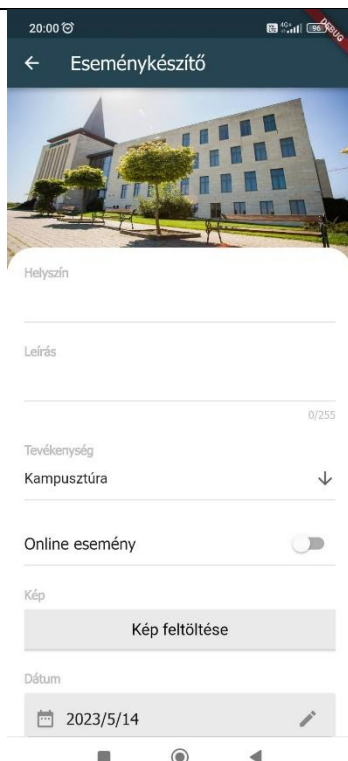
29. ábra: Szervező által generált QR kód



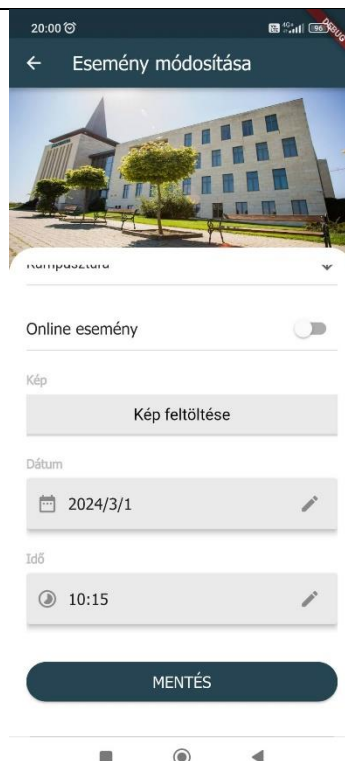
30. ábra: QR kód beolvasása

Esemény manipuláció

A szervezőknek lehetőségük van új esemény létrehozására (31. ábra) és meglévő események módosítására (32. ábra). Mindeket egy külön oldalon tehetik meg. Az eseménykészítés oldalát úgy érheti el egy szervező, hogy a főmenüben rákattintanak a plusz gombra, míg az eseményszerkesztést oldalt úgy, hogy rákattint egy esemény kártyáján a ceruza ikonra.



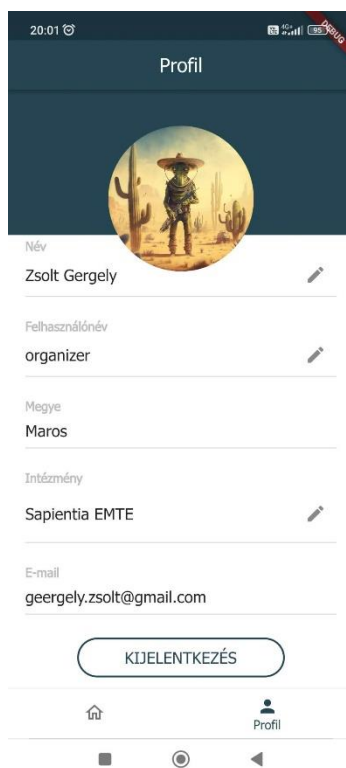
31. ábra: Esemény készítése



32. ábra: Esemény módosítása

Profil

Minden bejelentkezett felhasználónak és szervezőnek lehetősége van elérni a profilját ([33. ábra](#)) a navigációs menüből kiválasztva a profil ikont. Itt a képre kattintva elmenthet magának egy profilképet, és módosíthatja a személyes adatait, mint a nevét, felhasználónevét és az intézményét a megfelelő részen lévő ceruza ikonra kattintva.



33. ábra: Profil

6. Eszközök

6.1. Integrált fejlesztői környezet

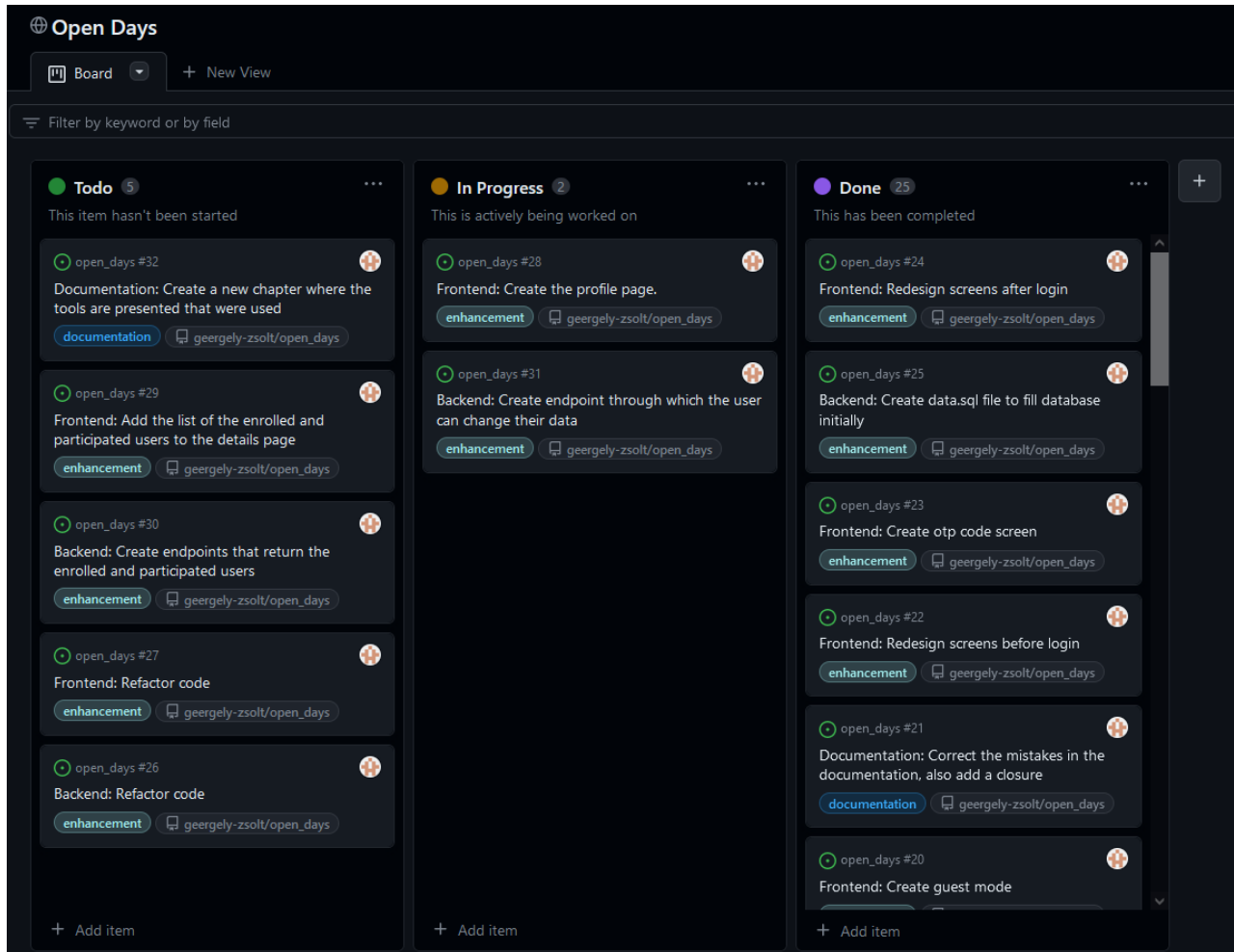
A fejlesztés során két integrált fejlesztői környezetet használtam. A kliensalkalmazás kódolásánál a Visual Studio Code, míg a szerverprogram kódolásánál az IntelliJ IDEA segítségét vettem igénybe. Az előbbi egy könnyen telepíthető fejlesztői környezet, amelyhez egyszerű hozzáadni olyan kiegészítőket, amelyek megkönnyítik a Flutterben és Dartban való fejlesztést. A második kifejezetten jól működik Java kóddal és könnyen létrehozható az kapcsolat az AWS webszolgáltatásaival. Mindkettő támogatja a kód automatikus formatálását, a kód könnyebb olvashatóságát különböző témák segítségével és a git-el való egyszerű kommunikációt.

6.2. Verziókövetés

A verziókövetésre a Git-et, GitHub-ot és a Sourcetree-t használtam. A Git-el lokálisan lehet kezelni a projektben történő változtatásokat a git parancsnokon keresztül. A projektnek létrehoztam lokálisan egy git repository-t így a Git minden változtatást érzékelt. Hogy ne kelljen mindig minden parancsot beírni, a Sourcetree segítségét vettem igénybe. Ebben meg lehet nyitni egy git repository-t és ad egy könnyen használható felhasználói felületet minden olyan műveletre, amit egyébként git parancsokon keresztül kellene végrehajtani. Könnyű használni és felgyorsítja a fejlesztést.

Végül, a GitHub egy felhő alapú tárhely, ahol a projekteket kódbázisát lehet tárolni. Ez lehetővé teszi, hogy bárki könnyedén elérjen bármilyen kódbázist, ha az publikus vagy engedélyt kapott rá. Így nem csak elérni lehet az adott kódbázist, de fejleszthető is könnyedén a már meglévő kód bárki által. A teendők menedzselésére is a GitHubot használtam, ahol készíthető egy “projekt” és számon lehet tartani, hogy milyen feladatok vannak és azt, hogy egy adott feladat melyik állapotban van. Az én esetemben 3 állapotot különböztettem meg, amikor csak létrehoztam a teendőt, de még várt arra, hogy dolgozzak rajta, amikor épp dolgoztam rajta és amikor már kész volt a feladat. Ez a [34. ábrán](#) tekinthető meg.

A projekt GitHub linkje: https://github.com/geergely-zsolt/open_days



34. Projektmenedzsment GitHub Project-ben

6.3. Tesztelés

6.3.1. API végpontok tesztelés

A végpontok tesztelésére a Postman-t használtam. A Postman egy intuitív program, amely egy könnyen kezelhető felületet biztosít HTTP kérések tesztelésére. Segítségével egyszerű megadni minden szükséges adatot, mint fejlécek, paraméterek vagy a kérések törzse.

6.3.2. Egységtesztelés

A szerveralkalmazásban a Service réteget függvényeit egységteszteltem le a JUnit5 keretrendszer segítségével. Az egységtesztetek olyan tesztek, amelyeket a fejlesztők írnak, hogy biztosítsák

általában egy adott függvény helyes működését. A lényeg, hogy a teszt a program egy adott kis, elszigetelt részét vizsgálja. A JUnit 5 egy olyan keretrendszer, amely több komponensből áll, mint a JUnit Platform, JUnit Jupiter, JUnit Vintage. Ezek közül az első, a JUnit Platform integrálva van néhány fejlesztő környezetben, mint a IntelliJ IDEA, NetBean vagy Visual Studio Code. A teszteket elkészítésére használtam a OpenAI ChatGPT chat botját. Tapasztalatom szerint, ha a fejlesztő tudja mit csinál, akkor a ChatGPT komoly segítség lehet, főként olyan helyeken, ahol a szükséges kód kellően elszigetelt [\[34\]](#).

6.4. Deploy

Azért, hogy a szerverem ne csak lokálisan legyen elérhető, szükségem volt egy platformra, ahol az futtat a gépemtől függetlenül és egy platformra, ahol tárolhatom az adatbázist. Mindkettőt a Heroku segítségével oldottam meg, amely egy felhő alapú platform. A Heroku önmagában nem támogatja a MySQL adatbázist, de vannak kiegészítők, mint a ClearDB MySQL, amelyek könnyen hozzáadhatóak a Herokuban létrehozott projektedhez. A Herokun futtatott szerver könnyen frissíthető, mert az össze van kötve a GitHubon tárolt projekttel. Minden alkalommal, ha GitHub-on frissül a projekt, a Herokuban is frissül.

7. Összefoglaló

7.1. Következtetések

Összességében megállapítható, hogy sikerült elérni a projekt főbb célkitűzéseit annak ellenére, hogy sok akadályba ütköztem, amelyekre nem számítottam. Készítettem egy olyan alkalmazást, ami futtatható Android és iOS operációs rendszereken, ugyanakkor segíti az egyetemi nyílt napok szervezőit abban, hogy információt osszanak meg az eseményekről, résztvevőkkel kapcsolatos adatokat gyűjtsenek minimális erőfeszítéssel, a résztvevőket pedig segíti azzal, hogy egy platformot biztosít, ahol minden esemény megtekinthető, a fontos információk elérhetők.

A főbb megvalósított funkciók a következők:

- Események listájának megtekintése vendégként.
- Új felhasználó készítése.
- Bejelentkezés, hogy a felhasználó elérhesse saját fiókját.
- Események listájának megtekintése bejelentkezett felhasználóknak.
- Események részleteinek megtekintése.
- Új esemény létrehozása szervezőként.
- Események törlése szervezőként.
- Események módosítása szervezőként.
- Személyes adatok módosítása a profil oldalon.
- Jelentkezés egy adott eseményre.
- Jelenlét jelzése az adott QR kód szkennelésével.

A projekt GitHub linkje: https://github.com/geergely-zsolt/open_days

7.2. Továbbfejlesztési lehetőségek

Az alkalmazás jelenlegi formájában is használható, de van néhány dolog, amit érdemes lenne a jövőben fejleszteni. Először is át kellene dolgozni a felhasználói felületet, hogy iOS rendszereken a Cupertino könyvtár grafikus elemeit használja. Ezáltal az alkalmazás megkapná az iOS rendszerekre jellemző stílust. Néhány új funkcionalitással is ki lehetne bővíteni, például, hogy értesítő e-mailt vagy telefonos értesítőt küldjön a jeletkezett felhasználónak azokról az eseményekről, amik aznap

lesznek, biztosítani az rendszergazda joggal rendelkező felhasználóknak egy weboldalt, ahol könnyebben megismerhető a rendszer, tovább bővíteni a rendszert, hogy bizonyos felhasználók egész csoportokat regisztrálhassanak egy adott eseményre anélkül is, hogy mindenkinek saját fiókja legyen. Hasznos lenne a tanárok számára, akik egy csoportot hoznak.

A szerveralkalmazást is lehetne fejleszteni. Jelenleg a Spring Boot által biztosított tokeneket használom, amivel adatbázis kommunikáció nélkül tudom ellenőrizni, hogy egy token érvényes vagy sem. Ezzel az a baj, hogy szerver oldalon nem tudom törölni vagy érvényteleníteni a tokenet. Csak akkor válik érvénytelenné, ha lejárt a beállított idő.

8. Irodalomjegyzék

- [1] „The History of Mobile Apps” <https://inventionland.com/blog/the-history-of-mobile-apps/> (Hozzáférés dátuma: 2023-01-01).
- [2] „Smartphone History: Looking Back (And Ahead) At a Modern Marvel” <https://blog.textedly.com/smartphone-history-when-were-smartphones-invented#third> (Hozzáférés dátuma: 2023-01-01).
- [3] „Top 8 Event Management Software Solutions for 2022” <https://connectteam.com/top-event-management-apps/> (Hozzáférés dátuma: 2023-01-01).
- [4] Planning Pod webhely <https://www.planningpod.com/> (Hozzáférés dátuma: 2023-01-01).
- [5] Localist webhely <https://www.localist.com/> (Hozzáférés dátuma: 2023-01-01).
- [6] Bitrix24 webhely <https://www.bitrix24.com/> (Hozzáférés dátuma: 2023-01-01).
- [7] Douglas E Comer, „The Internet Book: Everything You Need to Know About Computer Networking and How the Internet Works”, 5-ik kiadás, Chapman and Hall/CRC, 2018.
- [8] „What is the difference between webpage, website, web server, and search engine?” https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines (Hozzáférés dátuma: 2023-01-01).
- [9] „What is a URL” https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL (Hozzáférés dátuma: 2023-01-01).
- [10] „Responsive Design” <https://www.interaction-design.org/literature/topics/responsive-design> (Hozzáférés dátuma: 2023-01-01).
- [11] Jason Grigsby, „Progressive Web Apps”, A Book Apart, 2018.
- [12] „Introduction to progressive web apps” https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction (Hozzáférés dátuma: 2023-01-01).
- [13] „What is PWA? Progressive Web Apps in eCommerce Explained” <https://vuestorefront.io/pwa> (Hozzáférés dátuma: 2023-01-01).
- [14] „Mobile Application (Mobile App)” <https://www.techopedia.com/definition/2953/mobile-application-mobile-app> (Hozzáférés dátuma: 2023-01-01).

- [15] „5 Key Benefits of Native Mobile App Development” <https://www.velvetech.com/blog/native-mobile-app-development/> (Hozzáférés dátuma: 2023-01-01).
- [16] „Native vs cross-platform mobile app development” <https://circleci.com/blog/native-vs-cross-platform-mobile-dev/> (Hozzáférés dátuma: 2023-01-01).
- [17] „Diagrams.net webhely <https://app.diagrams.net/> (Hozzáférés dátuma: 2023-01-01).
- [18] Spring keretrendszer dokumentációja <https://docs.spring.io/spring-framework/docs/current/reference/html/> (Hozzáférés dátuma: 2023-01-01).
- [19] Spring Boot Reference Documentation <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> (Hozzáférés dátuma: 2023-01-01).
- [20] „What is Java Spring Boot?” <https://www.ibm.com/topics/java-spring-boot> (Hozzáférés dátuma: 2023-01-01).
- [21] „Spring Data JPA – Reference Documentation” <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> (Hozzáférés dátuma: 2023-01-01).
- [22] „Hibernate ORM” <https://hibernate.org/orm/documentation/6.1/> (Hozzáférés dátuma: 2023-01-01).
- [23] „Difference Between Spring vs Hibernate” <https://www.educba.com/spring-vs-hibernate/> (Hozzáférés dátuma: 2023-01-01).
- [24] „What is an API” <https://aws.amazon.com/what-is/api/> (Hozzáférés dátuma: 2023-01-01).
- [25] „What is REST?” <https://www.codecademy.com/article/what-is-rest> (Hozzáférés dátuma: 2023-01-01).
- [26] „HTTP Status Codes” <https://restfulapi.net/http-status-codes/> (Hozzáférés dátuma: 2023-01-01).
- [27] „HTTP requests” <https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests> (Hozzáférés dátuma: 2023-01-01).
- [28] „Building REST services with Spring” <https://spring.io/guides/tutorials/rest/> (Hozzáférés dátuma: 2023-01-01).
- [29] „RESTful Web Services, Java, Spring Boot, Spring MVC and JPA” <https://www.udemy.com/course/restful-web-service-with-spring-boot-jpa-and-mysql/> (Hozzáférés dátuma: 2023-01-01).
- [30] Flutter dokumentáció <https://docs.flutter.dev/> (Hozzáférés dátuma: 2023-01-01).

- [31] „What is Flutter and Its Advantages?” <https://www.adservio.fr/post/what-is-flutter-and-what-are-its-advantages> (Hozzáféres dátuma: 2023-01-01).
- [32] Dart dokumentáció <https://dart.dev/> (Hozzáféres dátuma: 2023-01-01).
- [33] „Flutter Riverpod 2.0: The Ultimate Guide” <https://codewithandrea.com/articles/flutter-state-management-riverpod/> (Hozzáféres dátuma: 2023-01-01).
- [34] What is JUnit5 and What Can You Use It For? <https://www.makeuseof.com/junit-5-what-use-for/>