

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK**



**SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM**

SmartCalendar értesítő applikáció

DIPLOMADOLGOZAT

Témavezető:
Györfi Ágnes,
Egyetemi tanársegéd
Dr. Kupán Pál,
Egyetemi docens

Végzős hallgató:
Simon Csaba

2023

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ**



**UNIVERSITATEA
SAPIENTIA**

SmartCalendar aplicație de notificare

LUCRARE DE DIPLOMĂ

Coordonator științific:

Györfi Ágnes,
Asistent universitar
Dr. Kupán Pál,
Conferențiar universitar

2023

Absolvent:

Simon Csaba

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA**
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

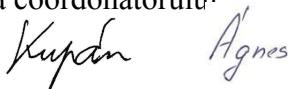
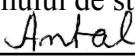
SmartCalendar Notification App

BACHELOR THESIS

Scientific advisor:
Györfi Ágnes,
Assistant professor
Dr. Kupán Pál,
Associate professor

Student:
Simon Csaba

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA Facultatea de Științe Tehnice și Umaniste din Târgu Mureș Programul de studii: Informatică		Viza facultății:
LUCRARE DE DIPLOMĂ		
Coordonator științific: dr. Kupán Pál Îndrumător: Györfi Ágnes	Candidat: Simon Csaba Anul absolvirii: 2023	
a) Tema lucrării de licență: SmartCalendar aplicație de notificare		
b) Problemele principale tratate: Studiul metodei de trimiterea a unui SMS în mod automat. Documentarea adecvată a stadiilor de proiectare a aplicațiilor.		
c) Desene obligatorii: Diagrame de proiectare pentru aplicația software realizată.		
d) Softuri obligatorii: O aplicație bazată pe tehnologia Spring Boot ca și Backend și front end bazat pe Angular		
e) Bibliografia recomandată: https://angular.io/guide/architecture https://spring.io/projects/spring-boot		
f) Termene obligatorii de consultații: săptămânal, preponderent online		
g) Locul și durata practiciei: Universitatea „Sapientia” din Cluj-Napoca, Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 414		
Primit tema la data de: 20.06.2022 Termen de predare: 02.07.2023		
Semnătura Director Departament 	Semnătura coordonatorului 	
Semnătura responsabilului programului de studiu 	Semnătura candidatului 	

Declarație

Subsemnatul/a SIMON CSABA, absolvent(ă) al/a specializării INFORMATICA, promoția 2023 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, TÎRGU-MUREȘ
Data: 16.06.2023

Absolvent

Semnătura...Simon

Kivonat

Dolgozatom témája a SmartCalendar nevű applikáció, amely lehetővé teszi az emberek számára, hogy hatékonyabban beosztassák az idejüket és jobban tervezhessék minden napjaikat. Az alkalmazás segít előre látni és kezelní az eseményeket, így nem maradnak le fontos találkozókról vagy kötelezettségekről. Emellett segít elkerülni a váratlan események okozta káoszt, mert az időt beosztva kevesebb lesz a meglepetés és a stressz. A SmartCalendar nemcsak az idő beosztásában nyújt segítséget, hanem az időpontokat szolgáltatóként is kezeli. Vállalatok számára lehetővé teszi, hogy egyszerűen és automatikusan értesítsék ügyfeleiket a programált szolgáltatásokról. Ez jelentős könnyebbséget és hatékonyságot jelent a szolgáltatók számára, mivel az ügyfeleknek időben kapnak értesítést, így nem felejtik el vagy késnek el a megbeszélt időpontokról. Az alkalmazás lényege, hogy bármilyen vállalkozás méretétől és formájától függetlenül hozzáférést biztosítson az időbeosztás és az ügyfélértesítés funkcióhoz. Ennek köszönhetően még a kis szolgáltatók is használhatják, akik korábban nem engedhették meg maguknak egy ilyen alkalmazás fejlesztését. A SmartCalendar tehát lehetőséget teremt arra, hogy az ügyfelek időben értesüljenek és ne felejtsék el a programált szolgáltatásokat. A dolgozat részletesen bemutatja az alkalmazás működését, és részletezi, hogyan biztosítja az ügyfélértesítést. Emellett megmutatja, hogyan képes az alkalmazás ingyenesen elérhetővé tenni ezt a funkciót a felhasználók számára. A SmartCalendar új lehetőségeket teremt az időbeosztás hatékonyabbá tételeben és a kis vállalkozások fejlődésében.

Rezumat

Teza mea se referă la o aplicație numită SmartCalendar, care le permite oamenilor să își gestioneze timpul mai eficient și să își planifice mai bine viața de zi cu zi. Aplicația îi ajută să anticipateze și să gestioneze evenimentele, astfel încât să nu rateze întâlniri sau angajamente importante. De asemenea, vă ajută să evitați haosul cauzat de evenimente neașteptate, deoarece, gestionându-vă timpul, vor fi mai puține surprize și mai puțin stres. SmartCalendar nu numai că vă ajută să vă gestionați timpul, dar vă gestionează și întâlnirile în calitate de furnizor de servicii. Aceasta permite întreprinderilor să își notifice cu ușurință și automat clienții cu privire la serviciile programate. Acest lucru înseamnă o ușurință și o eficiență semnificativă pentru furnizorii de servicii, deoarece clienții primesc notificări în timp util, astfel încât să nu uite sau să rateze programările. Aplicația este concepută pentru a oferi acces la funcționalitatea de programare și notificare a clientilor pentru orice afacere, indiferent de mărime sau formă. Aceasta înseamnă că o pot utiliza chiar și furnizorii de servicii mici care nu își puteau permite anterior să dezvolte o astfel de aplicație. Prin urmare, SmartCalendar oferă posibilitatea de a se asigura că clienții sunt informați în timp util și că nu uită serviciile programate. Lucrarea descrie în detaliu modul în care funcționează aplicația și cum asigură notificarea clientilor. De asemenea, se arată modul în care aplicația poate pune această funcționalitate la dispoziția utilizatorilor în mod gratuit. SmartCalendar deschide noi oportunități pentru o gestionare mai eficientă a timpului și pentru dezvoltarea micilor întreprinderi.

Abstract

My thesis is about an application called SmartCalendar, that allows people to manage their time more efficiently and plan their daily life better. The app helps them anticipate and manage events so they don't miss important appointments or commitments. It also helps you avoid chaos caused by unexpected events, because by managing your time, there will be fewer surprises and less stress. SmartCalendar not only helps you manage your time, it also manages your appointments as a service provider. It allows businesses to easily and automatically notify their customers of scheduled services. This means significant ease and efficiency for service providers, as clients receive timely notifications so they don't forget or miss appointments. The app is designed to provide access to scheduling and customer notification functionality for any business, regardless of size or form. This means that even small service providers who could not previously afford to develop such an application can use it. SmartCalendar therefore provides an opportunity to ensure that customers are informed in a timely manner and do not forget scheduled services. The paper describes in detail how the application works and how it provides customer notification. It also shows how the application can make this functionality available to users free of charge. SmartCalendar opens up new opportunities for more efficient time management and the development of small businesses.

Tartalomjegyzék

1. Bevezető	10
2. Célkitűzések	11
3. Követelmények	13
3.1. Felhasználói követelmények	13
3.2. Rendszerkövetelmények	15
3.2.1. Funkcionális követelmények	15
3.2.2. Nem funkcionális követelmények	16
4. A rendszer leírása	19
4.1. Rendszer architektúra	19
4.1.1. Backend	20
4.1.2. Frontend	28
4.1.3. Adatbázis	32
4.2. SMS küldési technológia	33
5. Felhasznált technológiák	36
5.1. Programozási nyelvek és környezetek	36
5.2. Verziókövetés	37
6. Felhasználói felületek bemutatása	38
6.1. Általános felhasználó felülete	38
6.2. SMS-t küldő applikáció felülete	43
7. Továbbfejlesztési lehetőségek	45
Összefoglaló	46
Köszönetnyilvánítás	47
Ábrák jegyzéke	48
Irodalomjegyzék	49

1. fejezet

Bevezető

Az embereket a 21. században foglalkoztatja az a nagy probléma, amely az időt taglalja. Hogyan is lehetne jobban beosztani az időt? Hogyan is lehetne ezáltal jobban megtervezni egy napot és több időt tölteni akár szeretteinkkel vagy az ember számára fontos dologgal? A SmartCalendar egy olyan applikáció, amely segítségével az emberek élete könnyebb lehet. Segít előre látni olyan eseményeket, amelyekre hivatalosak vagyunk, segít jobban megtervezni egy átlagos napot, segít abban, hogy ne raboljuk el egymás idejét, hiszen az egyetlen dolog, amit nem tudunk visszaadni senkinek, a tőle elrabolt idő.

A SmartCalendar segítségével az emberek jobban be tudják osztani az idejüket, könnyebben meg tudják tervezni napjukat illetve elkerülhetőek lesznek a váratlan események. Ez viszont csak az egyik oldala. Ezen applikáció egy másik nagy problémát is megold, ez nem más mint az idő beosztás szolgáltatóként.

Az alkalmazás alapjaiban szolgáltatóknak készült, amely segítségével a kis vállalatok is meg tudják engedni maguknak az ügyfél értesítést az időpont segítségével teljesen automatikusan. Az időpont bejegyzését követően a szolgáltatónak semmi más dolga nincs, csak várni az ügyfelet, fodrászatban, sminkesnél, körmösnél, vagy akár egy fogászon. A rendszer automatikusan értesíteni fogja az ügyfelet SMS-ben a megadott időben, például aznap reggel, vagy 24 órával az időpont előtt. Ezáltal ügyfelünket emlékeztetjük arra, hogy időpontja van nálunk, semmiképp ne felejtse el, hiszen ebben a rohanó világban ez bármikor előfordul.

Sajnos velem is előfordult már, hogy elfelejtettem az időpontot, amelyet a fodrász biztosított számonra. Beszélgettem vele és azt monda, hogy sajnos ők mint kis szolgáltatók, nem tudják megengedni maguknak, hogy beruházzanak egy alkalmazás fejlesztésre, amely értesíti az ügyfeleket, pedig sokszor jó lenne. Ekkor döntöttem úgy, hogy felépíték egy alkalmazást, amelyhez bárki hozzáfér függetlenül a vállalkozás méretétől és formájától. Ez lenne a SmartCalendar, amely kezeli az ügyfeleket, a szolgáltatásokat, illetve az időpontokat.

A dolgozatban részletesen bemutatom az alkalmazás mikéntjét, hogyan létezhet illetve hogyan képes biztosítani a felhasználóknak, hogy értesítse ügyfeleiket.

2. fejezet

Célkitűzések

Céлом, egy olyan alkalmazás létrehozása volt, amely segítségével a különböző kis vállalatok és szolgáltatók értesíteni tudják ügyfeleiket a különböző programálásokról, időpontokról teljesen automatikusan, a foglalás feljegyzése segítségével. Emellett céлом volt az is, hogy egy olyan rendszert építsek, amelyet programozói tudás nélkül, platformfüggetlenül tud bárki használni. Céлом volt, hogy a rendszer rendelkezzen alapvető funkciókkal is, melyek a következők:

- Regisztráció
- Fiók Aktiválás
- Bejelentkezés
- Jelszó módosítás
- Szolgáltatások létrehozása
- Szolgáltatások törlése
- Szolgáltatások szerkesztése
- Kliensek létrehozása
- Kliensek törlése
- Kliensek szerkesztése
- Foglalások, időpontok létrehozása
- Foglalások, időpontok törlése
- Foglalások, időpontok szerkesztése

Másodsorban céлом volt egy látható naptár implementációja, ennek segítségével az ügyfél sokkal könnyelmesebben tudja kezelni a foglalásokat. A naptárhoz is céлом volt néhány alapvető funkció beépítése, melyek a következők:

- Dinamikus naptár létrehozása

- Esemény létrehozása egyetlen napra kattintva
- Esemény módosítása
- Esemény törlése egyszeri rákattintással
- Események osztályozása szolgáltatás alapján szín szerint
- Naptár megtekintése hónap/nap/óra szinten
- Események mozgatása egyszerű *fogd* és *vidd* módszerrel

3. fejezet

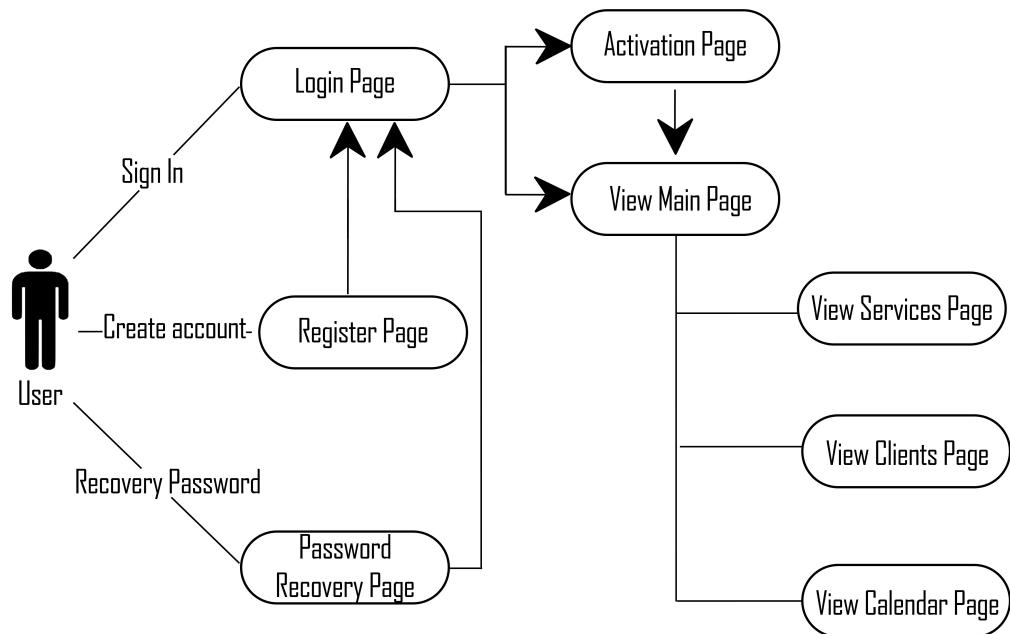
Követelmények

Miután meghatároztam a célkitűzéseket, készen álltam összeszedni a különböző követelményeket, amelyeket az alkalmazás fejlesztése során előírásnak használtam, ugyanis ezek a követelmények segítettek abban, hogyan is tervezzem meg és implementáljam az alkalmazást. A követelmények felírása után pontosan tudtam, hogy mire is fog szolgálni az alkalmazás, milyen funkciók lesznek benne illetve milyen környezetben fog működni és hogyan.

3.1. Felhasználói követelmények

A bejelentkezés/regisztrációhoz kötött használat miatt, beszélhetünk vendégről és bejelentkezett felhasználóról. A vendégnek csak a bejelentkezéshez, a regisztrációhoz és a jelszó visszaállításhoz van hozzáférése. Készítettem egy használati eset (use-case) diagramot, annak érdekében, hogy pontosan megértsük, a felhasználó szerepkörét. Ez a diagram a 3.1-es ábrán látható.

Miután a felhasználó megnyitja a mindenki számára egységes bejelentkezési felületet, regisztrálhat vagy bejelentkezhet már meglévő fiókjába. Itt lehetősége lesz az elfelejtett jelszó visszaállítására, amelyhez e-mail cím alapú megerősítésre van szükség. Ha regisztrál, szükséges aktiválnia fiókját az első bejelentkezés után, amely szintén e-mail cím alapú megerősítést fog igényelni, ezzel elkerülve a hamis illetve nem létező e-mail címek használatát, ezzel is biztonságosabbá téve az alkalmazást. A sikeres aktiválás / bejelentkezés után a felhasználónak lehetősége lesz elsősorban új szolgáltatásokat hozzáadni, amihez szükség van a szolgáltatás nevére, időtartamára, árára, illetve, hogy lehetséges e a szolgáltatás ideje alatt más szolgáltatást bejegyezni. Ez utóbbi azért szükséges, mert például egy fodrász egy haj fetsés közben be tudd vállalni egy férfi hajvágást is, ameddig a festék szárad. Tehát egy fodrász esetében, a szolgáltatásokra jó példa lehet a 3.1-es táblázat tartalma. A szolgáltatások módosításán kívül kliensekkel tud alapvető műveleteket végezni, hozzáadni, törlni és módosítani. Ezen kívül elérhető lesz számára a naptár opció, ahol egy valódi naptárt lát, és a meglévő szolgáltatásokat meglévő kliensekhez tudja majd társítani, ezzel kész is egy foglalás, amely esetén a kliens értesítéséről a backend gondoskodik majd a háttérben, a beállított idővel előtte.



3.1. ábra. Eset Diagram

Szolgáltatás neve	Időtartam	Ár	Közbeeső szolgáltatások
Férfi hajvágás	30 perc	30 lej	Nem
Fodrászat	60 perc	100 lej	Nem
Hajfestés	90 perc	200 lej	Igen

3.1. táblázat. Példa szolgáltatások

3.2. Rendszerkövetelmények

A rendszerkövetelmények meghatározzák, hogy az alkalmazás milyen technikai követelményeknek is kell megfeleljen. Ide tartoznak például a platformok és eszközök támogatása, az adatvédelem, a teljesítmény, a megbízhatóság és a biztonság kérdései. Ezek a követelmények biztosítják, hogy az alkalmazás megfelelően működjön a kiválasztott környezetben és a felhasználók számára biztonságos és kényelmes élményt nyújtson.

3.2.1. Funkcionális követelmények

A funkcionális követelmények olyan specifikációk, amelyek leírják az alkalmazás által elvárt viselkedést és funkcionalitást. Ezek az elvárások határozzák meg, hogy milyen feladatokat kell ellátnia az alkalmazásnak és milyen módon kell működnie a felhasználók számára. Íme néhány példa a SmartCalendar alkalmazás funkcionális követelményeire:

- Regisztráció: A felhasználóknak lehetőséget kell kapniuk a regisztrációra az alkalmazásban. Ehhez meg kell majd adniuk a szükséges adatokat, mint például az email címet, jelszót és szolgáltató nevet, ez után e-mailt fognak kapni egy aktivációs kódval, amelyet az első bejelentkezés után kell megadniuk.
- Bejelentkezés: A regisztrált felhasználóknak be kell tudniuk jelentkezni az alkalmazásba a regisztrált adataik segítségével, inaktív fiók esetén pedig tudniuk kell aktiválni a fiókot.
- Szolgáltatások kezelése: A felhasználóknak lehetőséget kell kapniuk szolgáltatások létrehozására a megfelelő adatok megadása után, szerkesztésére és törlésére. Ez lehetővé teszi számukra, hogy definiálják az általuk nyújtott szolgáltatásokat.
- Kliensek kezelése: A felhasználóknak lehetőséget kell kapniuk kliensek létrehozására, szerkesztésére és törlésére. Ez segít nekik nyomon követni a kliensek adatait és a velük kapcsolatos teendőket.
- Foglalások és időpontok kezelése: A felhasználóknak lehetőségük kell kapniuk foglalásokat és időpontokat létrehozni, szerkeszteni és törleni. Ez lehetővé teszi számukra, hogy ütemezzék a szolgáltatásokat és hatékonyan kezeljék az időbeosztásukat.
- Naptár megjelenítése: Az alkalmazásnak egy látható naptárat kell biztosítania, amely segíti a felhasználókat a foglalások és események könnyebb kezelésében. A naptárban meg kell jeleníteni a foglalásokat, eseményeket és időpontokat.
- Események osztályozása: Az alkalmazásnak lehetővé kell tennie az események színkódok szerinti osztályozását a szolgáltatások alapján. Ez segít a felhasználóknak átláthatóan megjeleníteni és kategorizálni az eseményeket a naptárban.
- Események mozgatása: A felhasználóknak lehetőséget kell kapnia az események mozgatására a naptárban
- Események mozgatása: A felhasználóknak lehetőséget kell kapniuk az események egyszerű és intuitív módon történő mozgatására a naptárban. Ez lehetővé teszi

számukra, hogy könnyen átrendezzék az időpontokat és foglalásokat az ütemtervük optimalizálása érdekében.

- Események részleteinek szerkesztése: A felhasználóknak lehetőséget kell kapniuk az események részleteinek szerkesztésére, mint például a cím, időpont, helyszín vagy további megjegyzések. Ez lehetővé teszi a rugalmas és pontos információk megadását az eseményekről.
- Események törlése: A felhasználóknak lehetőséget kell kapniuk az események egy-szerű törlésére a naptárból. Ez segít nekik a felesleges vagy elmaradt események eltávolításában, és rendszerezettségét teszi az ütemtervet.
- Események kategorizálása és szűrése: Az alkalmazásnak lehetőséget kell biztosítania az események kategorizálására és szűrésére a felhasználói preferenciák alapján. Ez segíti a felhasználókat az események könnyebb megtalálásában és az ütemtervük egyéni igények szerinti megjelenítésében.
- Értesítések és emlékeztetők: Az alkalmazásnak támogatnia kell az értesítéseket és emlékeztetőket az eseményekről, időpontokról és foglalásokról a kliensek fele. Ez segít a klienseknek az időben történő értesülésben és az eseményekre való felkészülésben.

3.2.2. Nem funkcionális követelmények

A nem funkcionális követelmények a rendszer tulajdonságait, teljesítményét, meg-bízhatóságát, felhasználhatóságát, skálázhatóságát és egyéb hasonló aspektusait írják le. Ezek azok a követelmények, amelyek nem közvetlenül a felhasználói interakcióval vagy a rendszer műveleteivel kapcsolatosak, hanem az általános rendszerjellemzőkkel foglalkoznak. Azok a nem-funkcionális követelmények, amelyeket én fontosnak tartottam megemlíteni, a következők:

- Teljesítmény: A rendszer és a felhasználó szempontjából is kiemelkedően fontos a válaszidő, hiszen a gyors reakcióidő jelentős előnyökkel jár mind az üzemeltetés, mind a felhasználói élmény terén. A rendszer szempontjából a gyors válaszidő lehetővé teszi hatékony működését a nagy terhelésű környezetekben is. Amikor a rendszer képes gyorsan reagálni a kérésekre, csökken a feldolgozási idő, és a rendszer hatékonyabban tudja kezelnı a nagy adatmennyiséget vagy a nagy látogatottságot. Ez növeli a rendszer skálázhatóságát és stabilitását, valamint csökkenti a leterheltség és a túlterheltség kockázatát. A felhasználók szempontjából a gyors válaszidő alapvető fontosságú a kiváló felhasználói élmény biztosításához. Amikor a rendszer gyorsan reagál a felhasználói interakciókra, a felhasználók nem érzik az idegesítő várakozást, és könnyedén elérhetik a kívánt információkat vagy funkciókat. A gyors válaszidő hozzájárul a jobb produktivitáshoz, elégédettséghez és lojalitáshoz, mivel a felhasználók pozitív élményt tapasztalnak, és kevésbé valószínű, hogy más rendszerekhez fordulnak.
- Rendelkezésre állás: A rendelkezésre állás kritikus fontosságú a rendszer és a felhasználók számára egyaránt, mivel biztosítja a folyamatos hozzáférést és működést a rendszerben. A rendszer szempontjából a rendelkezésre állás meghatározza

a megbízhatóságot és a stabilitást. Amikor a rendszer megbízható és rendelkezésre áll, minimálisra csökken a kiesési idő, vagyis az idő, amikor a rendszer nem elérhető vagy nem működik megfelelően. Ez fontos a folyamatos üzleti működés és az ügyfelek elégedettsége szempontjából. A magas rendelkezésre állás a rendszer hibatűrő képességét is jelenti, tehát képes folytatni a működést, még ha bizonyos komponensek vagy erőforrások hibásak is. A felhasználók számára a rendelkezésre állás bizalmat és elégedettséget eredményez. Amikor a rendszer minden elérhető, a felhasználók nem szabadnak kellemetlenséget vagy késedelmet a szolgáltatásokhoz vagy információkhoz való hozzáférésben. A megbízható rendelkezésre állás elősegíti a felhasználói bizalmat, elégedettséget és hűséget is.

- **Biztonság:** A biztonság kiemelkedő fontosságú mind a rendszer, mind a felhasználók számára, mivel védelmet nyújt az adatok, az erőforrások és a felhasználók számára. A rendszer szempontjából a biztonság alapvető fontosságú a sérülékenységek és támadások elleni védelem érdekében. A megfelelő biztonsági intézkedések biztosítják az adatok, az erőforrások és a rendszer integritását. Ez magában foglalja a hozzáférési jogosultságok kezelését, az adatvédelmet, a titkosítást, a biztonsági mentéseket és más védelmi mechanizmusokat. A megbízható biztonsági intézkedések segítenek minimalizálni a rendszerbejutás, a visszaélések és a károkozás kockázatát. A felhasználók számára a biztonság alapvető fontosságú a személyes adatok és az információk védelme érdekében. A megbízható biztonsági intézkedések lehetővé teszik a felhasználók számára, hogy bizalommal használják a rendszert, tudva, hogy adataik védettek és nem lesznek jogosulatlan hozzáférések vagy visszaélések. A biztonság hozzájárul a felhasználók bizalmához, elégedettségéhez és hűségéhez, és csökkenti a potenciális adatvesztés vagy a bizaomsértés kockázatát.
- **Skálázhatóság:** A skálázhatóság kiemelkedő fontosságú mind a rendszer, mind a felhasználók számára, mivel lehetővé teszi a hatékony és rugalmas növekedést a megnövekedett terhelés vagy igények kezelésére. A rendszer szempontjából a skálázhatóság meghatározza a rendszer képességét a terhelés növekedésének vagy csökkenésének megfelelő kezelésére. A skálázható rendszer rugalmasan alkalmazkodik a megnövekedett forgalomhoz, adatmennyiségekhez vagy felhasználói igényekhez anélkü, hogy veszélyeztetné a teljesítményt vagy a rendelkezésre állást. Ez lehetővé teszi a hatékony működést a változó környezetben és lehetővé teszi a rendszer fejlesztését a jövőbeli igényekre való felkészülés érdekében. A felhasználók szempontjából a skálázhatóság garantálja a folyamatos hozzáférést és a kiváló felhasználói élményt még a nagy forgalmú időszakokban vagy az igények megnövekedésekkel is. Amikor a rendszer képes kezelní a megnövekedett terhelést anélkü, hogy lassulna vagy összeomlana, a felhasználók nem tapasztalnak fennakadást vagy hosszú várakozási időt. Ez hozzájárul a felhasználói elégedettséghez, a jobb teljesítményhez és a hűséghöz.
- **Felhasználói élmény:** A felhasználói élmény kiemelkedő fontosságú mind a rendszer, mind a felhasználók számára, mivel befolyásolja az interakció minőségét, a felhasználó elégedettségét és a rendszer sikereit. A rendszer szempontjából a felhasználói élmény meghatározza a rendszer könnyű használhatóságát és intuitív működését. Az egyszerű és érthető felhasználói felület, valamint a jól szervezett navigáció és interakció lehetővé teszik a felhasználók számára, hogy könnyedén megtalálják és

használják a rendszer funkcionálitását. Ez csökkenti a felhasználók frusztrációját, növeli a hatékonyságot és javítja a termék elfogadottságát. A felhasználók szempontjából a felhasználói élmény befolyásolja az interakció zavartalanosságát és elégédettségét. Az intuitív és felhasználóbarát felület lehetővé teszi a könnyű navigációt és a felhasználók számára szükséges információk és funkciók gyors elérését. A pozitív felhasználói élmény hozzájárul a felhasználói elégédettséghez, a termék hűséghez és az esetleges ajánlásokhoz más felhasználók számára.

- Megfelelőség: A megfelelőség kiemelkedő fontosságú mind a rendszer, mind a felhasználók számára, mivel biztosítja, hogy a rendszer és az üzemeltetője megfeleljenek a jogi és szabályozási előírásoknak, valamint az iparági szabványoknak és irányelvöknek. A rendszer szempontjából a megfelelőség biztosítja a jogi és szabályozási követelmények teljesítését. Ez magában foglalja a személyes adatvédelmi szabályok, az adatbiztonsági előírások és a jogi követelmények betartását. A megfelelőség garantálja, hogy a rendszer működése összhangban legyen a vonatkozó törvényekkel és előírásokkal, és minimalizálja a jogi kockázatokat vagy a szabálytalanságokkal járó következményeket. A felhasználók szempontjából a megfelelőség bizalmat és védelmet nyújt. Amikor a rendszer megfelel a jogi és szabályozási előírásoknak, a felhasználók biztosak lehetnek abban, hogy személyes adataik és információik védettek, és a rendszer tiszteletben tartja a vonatkozó adatvédelmi és biztonsági irányelveket. A megfelelőség hozzájárul a felhasználói bizalomhoz, elégédettséghez és hűséghez, és csökkenti a reputációs kockázatokat vagy a jogi vitákat.

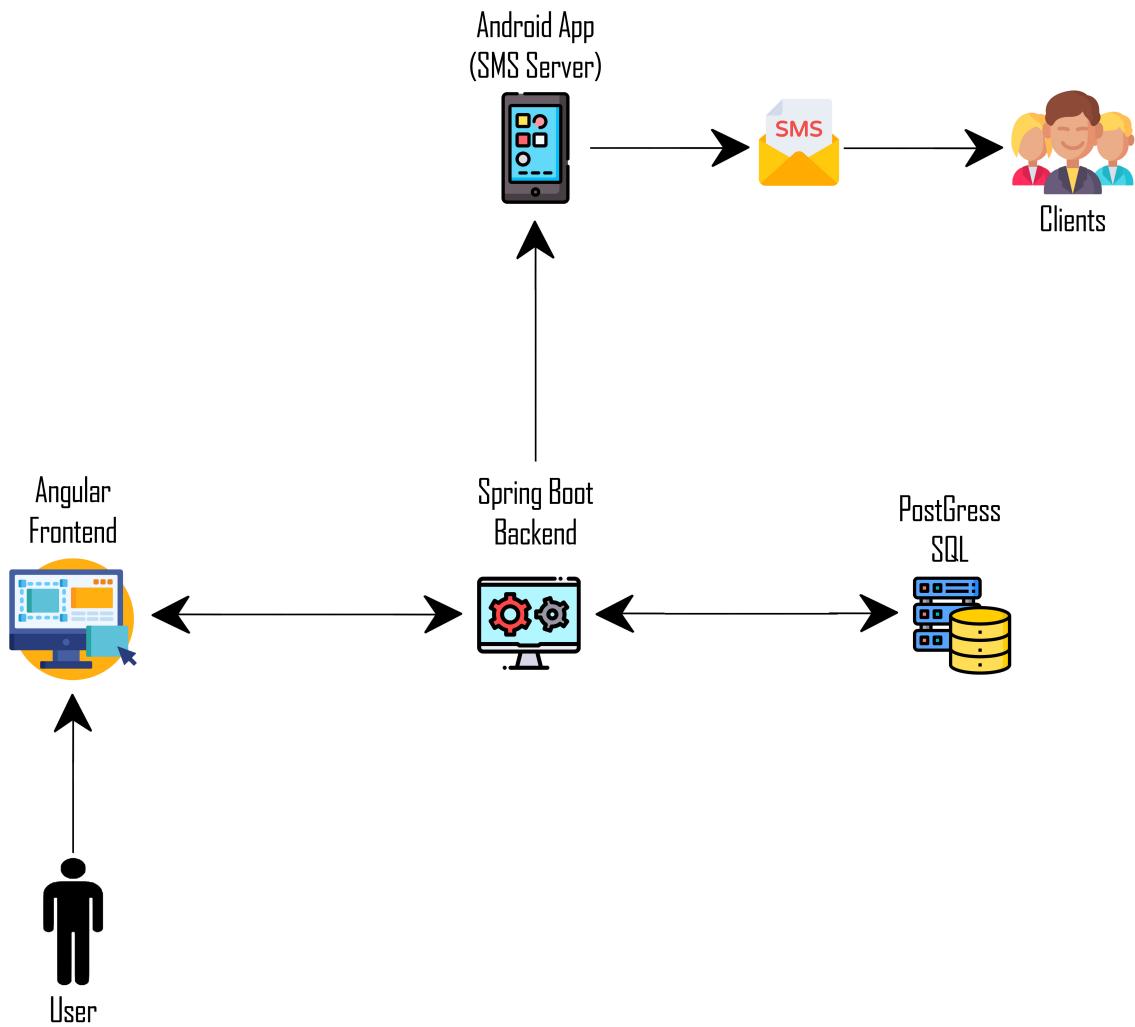
4. fejezet

A rendszer leírása

A rendszer leírása során ki fogok térti az architektúrára, a felépítésre, a biztonságra, a különböző SMS küldési technológiákra, megoldásokra, be fogom részletesen mutatni az általam választott és implementált megoldást is. E mellett szó lesz arról, hogy a projekt hogyan és miként lett elérhető élesben illetve milyen tesztek voltak elvégezve.

4.1. Rendszer architektúra

Az alkalmazás architektúrája alapvetően az Angular frontend, a Spring Boot backend és a PostgreSQL adatbázisra épül. Az Angular frontend felelős a felhasználói felület megjelenítéséért és a felhasználói interakciók kezeléséért. A frontend komponensek és szolgáltatások segítségével lehetővé teszi a felhasználók számára a bejelentkezést, az üzenetek megírását és a címzettek kiválasztását. Az Angular keretrendszer lehetővé teszi a dinamikus és reszponzív felhasználói élmény biztosítását, valamint a böngészőben történő valós idejű adatfrissítéseket. A Spring Boot backend gondoskodik az alkalmazás üzleti logikáról és az adatok kezeléséről. A backend komponensek végzik a felhasználók autentikációját és autorizációját, valamint a küldendő SMS üzenetek kezelését és továbbítását. A Spring Boot keretrendszer lehetővé teszi a hatékony adatfeldolgozást, a biztonsági intézkedések beépítését és a megbízható hálózati kommunikációt. A PostgreSQL adatbázis tárolja és kezeli az alkalmazás adatait. Az adatbázisban elhelyezett táblák tartalmazzák a felhasználói profilokat, a címzettek listáját és az elküldött üzeneteket. A PostgreSQL adatbázis relációs jellege lehetővé teszi az adatok hatékony és strukturált tárolását, valamint az adatlekérdezések optimális végrehajtását. [7] A rendszer biztonsági szempontokat is tartalmaz. Az alkalmazásban beépített autentikációs és autorizációs mechanizmusok biztosítják, hogy csak jogosult felhasználók tudjanak bejelentkezni és programálásokat végezni. Az alkalmazás adatai titkosítva vannak tárolva a PostgreSQL adatbázisban. A rendszer architektúrája megtekinthető a 4.1-es ábrán, amelyen jól látható, hogy az adatbázissal csak a backend szerver kommunikál, ezzel garantálva a felhasználók és az adatbázis biztonságát is. E mellett az architektúrán látható egy telefonos applikáció, amely segítségével az SMS küldések történetének, a kliensek fele.



4.1. ábra. A rendszer archíktúrája

4.1.1. Backend

A Spring Boot egy Java alapú keretrendszer, amelyet sok fejlesztő választ az alkalmazás backend részének fejlesztéséhez. Számos előnye miatt jó választás volt erre a projektre is, illetve előző tapasztalataim is volta vele. Az egyik legnagyobb előnye, hogy a spring alapú alkalmazásokat, csak futtatni kell. [1] A Spring Boot technológiát alapvetően nagyon logikus felépítésű rendszernek gondolom. A rendszernek több komponense van, melyeket most fel fogok sorolni illetve bemutatom az adott komponenseken belüli saját fájljaimat, részletes magyarázattal illetve ismertetővel, esetlegesen kód részletekkel ellátva:

- **Fő alkalmazásosztály (Main Application Class):** Ez az osztály az alkalmazás belépési pontja. A public static void main metódus található benne, amelyet a Java virtuális gép (JVM) futtat az alkalmazás indításakor. Ebben az osztályban történik a Spring Boot alkalmazás kontextusának inicializálása és konfigurálása. Ezen osztályom hasonló bármely más Java Spring Boot belépési osztályához, annyi különbséggel, hogy nálam van egy @EnableScheduling annotáció, amely lehetővé teszi a feladatok ütemezésének engedélyezését az alkalmazásban. Az annotációt a konfigurációs osztályok fölött alkalmazzák, és jelezheti a Spring keretrendszernek, hogy a Scheduler osztályokat és az ütemezett feladatokat használni kívánjuk. Amikor az @EnableScheduling annotációt a konfigurációs osztályra alkalmazzuk, a Spring Boot észleli és aktiválja a Scheduler funkciókat az alkalmazásban. Ez lehetővé teszi, hogy használhassuk a @Scheduled annotációt a Scheduler osztályokban vagy bármely más osztályban, ahol időzített feladatokat szeretnénk végrehajtani. Ezen keresztül később a Scheduling osztályok között megjelenő *SMScheduler* osztály fog működni. [3]
- **Konfigurációs osztályok:** A Spring Bootban a konfigurációs osztályok felelősek az alkalmazás beállításainak definiálásáért. Ezek az osztályok jellemzően annotációkkal vannak ellátva, például a @Configuration, @EnableAutoConfiguration, @ComponentScan, amelyek meghatározzák a komponenseket és a konfigurációt. [4] Itt konfigurálhatók például az adatbázis-kapcsolatok, a biztonsági beállítások és a különböző modulok aktiválása. Ezen kategóriában nálam négy osztály lehetséges fel, melyek a következők:
 - AuthenticationFilter: ez az osztály egy szűrőt implementál, amely az autentikációt kezeli a Spring alkalmazásban. A szűrőt a doFilter metódusban definiáljuk, amely a GenericFilterBean ősosztályból származik. Az osztály konstruktorában a TokenAuthenticationService osztályt injektáljuk, amely felelős a token alapú autentikációval kapcsolatos feladatokért. Ezen osztályban történik a HTTP kérésben található token érvényesség és hitelesség ellenőrzése is, a TokenAuthenticationService segítségével, melyet példányosítottam.
 - TokenAuthenticationService: ez az előző osztályban már alkalmazott TokenAuthenticationService osztály, mely felelős a JWT (JSON Web Token) alapú autentikációval kapcsolatos feladatokért. Az osztályban változók találhatóak, melyek meghatározzák a JWT érvényességi idejét, a titkos kulcsot és a HTTP kérés fejlécében használt token információkhöz tartozó neveket. Az osztályon belül található négy metódus is, melyek a token generálásáért, ellenőrzéséért illetve egy speciális fejléc hozzáadásáért felelnek, mely sikeresség esetén fut le és hozzáadja a JWT-t az Access-Control-Expose-Headers beállításával.
 - UserAuthenticationProvider: ez az osztály felelős a felhasználó autentikációjáért a Spring Security keretrendszerben. Az osztály konstruktorában injektálva van a UserRepository és a BCryptPasswordEncoder. A UserRepository segítségével az osztály hozzáfér az alkalmazás felhasználóinak adatbázisbeli tárolásához, míg a BCryptPasswordEncoder azért felel, hogy a jelszavakat biztonságosan titkosítsa és ellenőrizze. Ha sikeres volt az autentikáció, akkor a UserEntity objektumból kinyerjük a felhasználó szerepét, és létrehozunk egy

GrantedAuthority objektumot, amely tartalmazza ezt a szerepet. A visszatérési értékben a UsernamePasswordAuthenticationToken segítségével visszaadjuk az autentikált felhasználó azonosítóját, a null jelszót és a jogosultságok listáját.

- WebSecurityConfig: ez az osztály a Spring Security beállításait definiálja. A corsConfigurationSource metódus egy CorsConfigurationSource bean-t definiál. Ez a metódus felelős a Cross-Origin Resource Sharing (CORS) beállítások konfigurálásáért. A metódusban beállítjuk a megengedett eredeteket (allowedOrigins), a megengedett HTTP módszereket (allowedMethods) és a megengedett fejléceket (allowedHeaders).
- **Controller osztályok:** A Spring Boot alkalmazásban a controller osztályok felelősek a HTTP kérések kezeléséért és a válaszok generálásáért. Ezek az osztályok általában annotáltak a @RestController vagy a @Controller annotációval, és metódusokat tartalmaznak, amelyeket a megfelelő HTTP útvonalakhoz rendelnek. A kéréseket feldolgozzák és válaszokat generálnak, például JSON formátumban. A Controller osztályokat DTO-k segítségével hívjuk meg, melyeket a Mapper osztályok segítségével alakítunk Entity osztályokká. A controllerekben megjelenő metódusok @Get, @Post, @Put és @Delete annotációval vannak ellátva, melyek az adott útvonal lekérési típusát fogják jelölni illetve az utána zárójelben lévő útvonal fogja magát az elérés útvonalát jelenteni. Ezen utvonallal a következőképpen épül fel a különböző kontrollerek és metódusok alapján: *HOST + @RequestMapping + @PostMapping*. A Host jelöli az URL-t amelyen elérhető a szerver, a @RequestMapping a kontroller útvonalát illetve a @PostMapping a kontrolleren belüli utat a metódushoz. Az egyszerű vállalat beszurására alkalmas URL tehát így épülhet fel:
 - HOST: http://localhost:8080
 - @RequestMapping: /company
 - @PostMapping: /inserttehát az eredmény `http://localhost:8080/company/insert` mely egy POST típusú kérés küldésére alkalmas. [4] Az általam használt kontrollerek, a következők voltak:

- Company Controller
- Customer Controller
- Email Controller
- Event Controller
- Exception Controller
- Forget Password Controller
- Service Controller
- SMS Controller
- User Controller

- Service osztályok:** A service osztályok a központi üzleti logikát valósítják meg. Ezek az osztályok a controller osztályuktól elkülönülve tartalmazzák az üzleti logikát és az alkalmazás működését befolyásoló feldolgozási lépéseket. A service osztályokat általában az interfések és az implementációk használatával definiálják, amelyek lehetővé teszik a könnyű tesztelést és a kód modularitását. Ezek végett én is interfészkekkel és implementációs fájlokkal valósítottam meg az alkalmazás központi logikáját. Ezen osztályok a repository osztályok segítségével végzik el a különböző feladatokat, amelyek előre implementáltsága miatt egy-egy művelet csak egy parancs futtatását igényli. Egy új entity (a Dto-ból átalakított példány) beszúrására alkalmas kódrészlet megtekinthető a 4.1-es kódrészletben.

4.1. kódrészlet. Egy új vállalkozás beszúrása az adatbázisba

```
//CompanyService.java
package com.example.SmartCalendar.service;
import com.example.SmartCalendar.helper.mapper.company.CompanyDto;
import com.example.SmartCalendar.helper.mapper.company.CompanyMapper;
import com.example.SmartCalendar.model.CompanyEntity;
import com.example.SmartCalendar.repository.CompanyRepository;

public class CompanyServiceImpl implements CompanyService{
    private final CompanyRepository companyRepository;
    private final CompanyMapper companyMapper = new CompanyMapper();
    @Override
    public CompanyDto create(CompanyDto companyDto){
        if(companyDto.getName().isEmpty()){
            throw new ServiceException("The name can't be empty.");
        }
        CompanyEntity companyEntity = companyMapper.toEntity(companyDto);
        //Átalakítás a mapper segítségevel
        return
            companyMapper.toDto(companyRepository.save(companyEntity));
        //Mentes és visszaterítés
    }
}
```

Ezen osztályokból nekem 8 darab van, amelyek a különböző feladatok logikáit végzik, hasonló módon a 4.1-es kódrészlethez. Ezen osztályok a következők:

- CompanyService osztály amely a különböző felhasználók vállalatainak adaiért felelős
- CustomerService osztály amely a felhasználókhhoz tartozó kliensekért felelős
- EventService osztály amely a különböző felhasználókhhoz tartozó eseményekért felelős
- ForgetPasswordService osztály amely az elfelejtett jelszók visszaállításához szükséges metódusokért felelős
- ServiceService amely a különböző felhasználókhhoz tartozó szolgáltatásokért felelős

- SMSService amely az SMS-eket tartalmazó lista felépítéséért felelős
 - SmtpMailSender amely egy darab e-mail küldéséért felelős megadott címzett, tárgy és üzenet esetén
 - UserService amely a felhasználók regisztrációáért, aktuális felhasználó lekérésséért illetve a felhasználók aktiválásáért felel.
- **Adatbázis-hozzáférési réteg (Repository osztályok):** Ha az alkalmazás adatbázist használ, akkor a repository osztályok adatainak elérésére és kezelésére szolgálnak. Ezek az osztályok az adatbázis műveletek végrehajtásáért felelnek, például a lekérdezések végrehajtása és az adatok tárolása. A Spring Boot egyszerűsíti az adatbázis-hozzáférést és kezelést a Spring Data JPA keretrendszer segítségével. A Spring Data JPA egy nagyon hatékony és produktív megközelítést kínál az adatbázis-hozzáférésre, amely automatikusan generálja az adatbázis műveleteket a repository interfések alapján. A repository osztályokat az @Repository annotációval jelöljük meg, ami lehetővé teszi a Spring keretrendszer számára, hogy felismerje és kezelje ezeket az osztályokat. A repository osztályok legtöbbször az org.springframework.data.repository.CrudRepository vagy az org.springframework.data.jpa.repository.JpaRepository interfést implementálják. Ezek az interfések definiálnak egy alapvető műveletkészletet az adatok eléréséhez és kezeléséhez, például a létrehozás, olvasás, frissítés és törlés műveleteket. A repository osztályok együttműködnek az adatbázis ORM (Object-Relational Mapping) réteggel, amely a Java objektumokat és az adatbázis táblákat közötti leképezést végzi. A Spring Data JPA használatával a repository osztályok automatikusan generálják az SQL-lekérdezéseket és az adatbázis-műveleteket az ORM réteg által. Ezek alapján nekem is van 7 interfész, amelyek mindegyike a JpaRepository osztályból van származtatva. Az alap műveletek mellett, egyszerűen a különböző fejlécök segítségével lehet saját metódusokat is implementálni, amelyek a JpaRepository segítségével generálodni fognak. A 4.2-es kód részletben láthatunk egy példát amely a CompanyRepository osztályt mutatja be, amely már ki van bővítve egy findByName típusú metódussal, amely a különböző már regisztrált vállalatokat keresi meg név alapján. Mivel jelenleg Romániában egyetlen vállalatnak sem lehet ugyanaz a neve és céghozzájárulása, így ez a metódus csak egyetlen értéket térít vissza, vagyis az általa keresett név mező az adatbázisban unic.

4.2. kód részlet. CompanyRepository.java

```
//CompanyRepository.java
package com.example.SmartCalendar.repository;

import com.example.SmartCalendar.model.CompanyEntity;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CompanyRepository extends
    JpaRepository<CompanyEntity, Long> {
    CompanyEntity findByName(String name);
}
```

- **Modell osztályok:** Ezek az osztályok definiálják az adatstruktúrákat és azok attribútumait, amelyeket az alkalmazás használ. A modell osztályok gyakran a táblákhoz vagy entitásokhoz kapcsolódnak az adatbázisban. A modell osztályokat általában a Java osztályokként definiálják, amelyek az alkalmazásban használt adatokat reprezentálják. Ezek az osztályok tartalmazhatnak adattagokat, gettereket és settereket, valamint egyéb műveleteket az adatok kezeléséhez és manipulálásához. Az én esetben ezek hozzá vannak rendelve táblához a @Table annotációval, illetve az oszlopok az oszlopokhoz a @Column annotációval. E mellett a külső kulcsok esetén jelölve van a kapcsolat a megfelelő annotációval - például @ManyToOne a sok az egyhez kapcsolat - illetve a különböző adattagok privátak, ezért gettereket és settereket is alkalmaztam. Ezen kívül az id-k @Id annotációval vannak ellátva és egy @GeneratedValue annotációval amelyben megadható az Id automatikus generálási módja. A model osztályok között fellelhető 7 különböző modell osztály, melyek a következők:

- CompanyEntity
- CustomerEntity
- EventEntity
- ForgetPasswordEntity
- ServiceEntity
- SMSEntity
- UserEntity

Ezek közül bemutatom a CompanyEntity-t mely egy vállalatot definiál a *company* táblában, ez látható a 4.3-as kódrészleten, a kód továbbá tartalmaz gettereket és settereket is, de azt nem tartom fontosnak itt bemutatni.

4.3. kódrészlet. CompanyEntity.java

```
//CompanyEntity.java
@Entity
@Table(name = "company")
public class CompanyEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "logo")
    private String logo;

    @Column(name = "createDate")
    private String createDate;
}
```

- **Helper/Mapper osztályok:** Ezen osztályok segítséget nyújtanak az adatok átalakításában és a különböző osztályok közötti konverzióban. A Spring Boot alkalmazásban gyakran alkalmazzák őket, amikor adatokat kell átváltani a modell osztályok, a DTO-k (Data Transfer Object) vagy más reprezentációk között. A helper mapper osztályok segítségével könnyedén átalakíthatjuk az egyik osztályt a másikba vagy a közöttük lévő adatokat más formátumokba. Például, amikor egy entitás objektumot szeretnénk átalakítani egy DTO objektummá, vagy amikor az adatbázisból származó eredményeket alakítjuk át modell objektumokká. Ezen osztályok között azonos mappába vannak tárolva a Dto osztályok összesen 14 különböző DTO illetve 7 mapper osztály. Ezen Dto-k elég hasonlóak a modell osztályokhoz, viszont ezek a frontend oldal objektum attributumait tartalmazzák. A 4.5-ös kód részletben bemutatok egy mapper függvényt, amely egy Dto-t alakít át Entity-vé.

4.4. kód részlet. CompanyMapper.java - toEntity Function

```
//CompanyMapper.java

public CompanyEntity toEntity(CompanyDto companyDto){
    CompanyEntity companyEntity = new CompanyEntity();

    companyEntity.setId(companyDto.getId());
    companyEntity.setName(companyDto.getName());
    companyEntity.setLogo(companyDto.getLogo());

    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd
        HH:mm:ss");
    LocalDateTime now = LocalDateTime.now();
    companyEntity.setCreateDate(dtf.format(now));

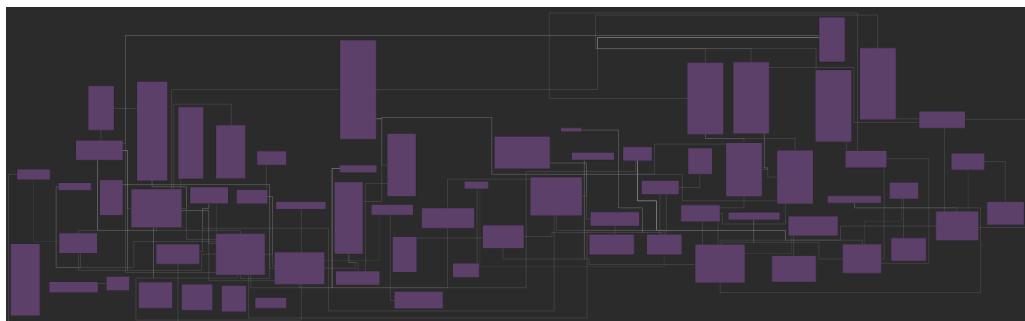
    return companyEntity;
}
```

- **Scheduler osztályok:** Ezek az osztályok a Spring Boot alkalmazásban a feladatok ütemezéséért és időzítéséért felelnek. Ezek az osztályok lehetővé teszik, hogy meghatározzunk olyan feladatokat, amelyeket időzítve szeretnénk végrehajtani, például rendszeres időközönként, naponta, hetente vagy bármilyen más időpontban. [5] Nálam egy ilyen osztály van, amely az SMS küldésért felelős, ez az SMS Scheduler osztály. Ez az osztály tartalmazza a @Scheduled annotációt amelyben paraméterként meg van adva, hogy milyen időközönként fusson le az időzített kód részlet. Nekem ebben az esetben a teszt ideje alatt, 1 percenként fut le, amely lekéri az összes programálást az events táblából, és amelyik egy perces különbségen belül 24 óra múlva lesz aktív, azon adatokból létrehoz egy smsEntity-t, amelyet majd elment az adatbázisba. Ez a tábla lesz a köztes kapcsolat a szerver és az sms-t küldő applikáció között. Az smsEntity itt megkapja a telefonszámot illetve az sms szöveget, melyet az SMS-t küldő eszköz egyből le tud kérni és használni tud. Az SMS Scheduler osztály megtekinthető a 4.6-os kód részletben.

4.5. kódrészlet. SMSScheduler.java

```
@Component
@AllArgsConstructor
public class SMSScheduler {
    final int beforeHours = 24;
    final int plusminute = 1;
    private final EventRepository eventRepository;
    private final SMSRepository smsRepository;
    @Scheduled(fixedRate = 60000) // 5 perc = 300000 milliszekundum
    public void checkSMS() {
        List<EventEntity> events = eventRepository.findAll();
        for (EventEntity event:events) {
            LocalDateTime fiveMinutesLater =
                LocalDateTime.now().plusMinutes(plusminute);
            LocalDateTime s1 = event.getStartTime();
            if(s1.minusHours(beforeHours).isAfter(LocalDateTime.now()) &&
               s1.minusHours(beforeHours).isBefore(fiveMinutesLater)){
                SMSEntity smsEntity = new SMSEntity();
                CustomerEntity customerEntity = event.getCustomerEntity();
                smsEntity.setNumber(customerEntity.getPhoneNumber());
                smsEntity.setMessage("Hy "+customerEntity.getFirstName() +
                    " " + customerEntity.getLastName() + "! You have to go
                    here tomorrow: " +
                    event.getServiceEntity().getName() +", then:
                    "+event.getStartTime().getHour() +
                    ":"+event.getStartTime().getMinute());
                smsRepository.save(smsEntity);
            }
        }
    }
}
```

Ezen osztályok közötti kapcsolatoknak bonyolultsága a 4.2-es ábrán megtekinthető, ezzel indokolnám azt, hogy miért nem jelent meg ebben a dolgozatban egy teljes osztálydiagram. Összesen szám szerint 66 osztály és interface kapcsolódik egymáshoz!



4.2. ábra. Osztály diagram bonyolultsága

4.1.2. Frontend

Az Angular egy nyílt forráskódú, TypeScript-alapú webes keretrendszer, amelyet a Google fejlesztett ki. Célja a hatékonyabb és strukturáltabb webalkalmazások fejlesztése. Az Angular lehetővé teszi a fejlesztők számára, hogy dinamikus, reszponzív és keresőbarát felhasználói felületeket hozzanak létre. Az Angular alapvetően egy komponens-alapú keretrendszer, amelynek központi elemei a komponensek. A komponensek önálló egységek, amelyeket újra felhasználható módon építhetünk fel. A komponensek segítségével különböző részekre osztatjuk fel az alkalmazást, és ezáltal könnyen karbantarthatóvá és tesztelhetővé válik. [2] Ebben az esetben is, a backendhez hasonlóan megpróbáltam a lehető legoptimálisabb és "legszebb" kódot létrehozni, amely programozói szempontból is megfelel a követelményeknek. Az angular projekt létrehozása után egy reszponzív design-t választottam a felület megalkotásához, amely ingyenesen elérhető volt az interneten. Ezt felhasználva építettem fel egy olyan reszponzív kezelő felületet, amely alkalmas ezen kaliberű projekt felépítésére. Ebben az esetben az applikáció felépítését két nagy részre lehetne osztani, az első a *core* és a második a *modules*.

- Core: A core mappa tartalmazza a projekt alapvető és központi részeit, melyek a következők:
 - Helpers (segédosztályok): A Helpers mappa olyan segédosztályokat tartalmaz, amelyek különböző hasznos funkciókat valósítanak meg, például a hiba kezelésére szolgáló Error Interceptor osztály vagy a Jwt Token ellenőrzésére szolgáló Jwt Interceptor osztály.
 - Models (modellek): A Models mappa a projektben használt adatmodell definíciót tartalmazza. Ebben az esetben 10 modell található a modell osztályban, amelyek struktúrája hasonló. Ezen modellek felépítése megfelel a backend szerinten a Dto osztályoknak, hiszen ezen modellek segítségével küldök adatokat a backend szerverre. A 4.7-es kódrészletben megtekinthető a Company Model.

4.6. kódrészlet. Company Model

```
//company.model.ts
export class CompanyDto {
    id!: number;
    name!: string;
    logo!: string;
    createDate!: string;
}
```

- Services (szolgáltatások): A Services mappa olyan osztályokat tartalmaz, amelyek az alkalmazás üzleti logikáját valósítják meg. Ezek a szolgáltatások kommunikálnak a backenddel vagy más külső erőforrásokkal, kezelik az adatok lekérdezését és módosítását, és általában a komponensekkel való kommunikáció közvetítői. Nyolc különböző ilyen szolgáltatásom van amelyből a 4.8-as kódrészletben bemutatom a Company Service-t, amely egy ilyen vállalat készítésére alkalmas, ahol természetesen a közvetített objektum egy CompanyDto.

4.7. kódrészlet. Company Service

```
//company.service.ts
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { CompanyDto } from '../models/company.model';

@Injectable({ providedIn: 'root' })
export class HotelService {
    constructor(private http: HttpClient) {}

    createCompany(companyDto: CompanyDto):
        Observable<CompanyDto> {
        return this.http.post<CompanyDto>(
            'http://localhost:8080/company/create/', CompanyDto
        );
    }
}
```

- Modules: A modules mappában az applikáció moduljai találhatóak, melyekkel ténylegesen fel van építve a felhasználó által is látott weboldal. A modules mappában három nagy csoport van. Ezeknek a moduloknak különböző útvonal és modul fájlai vannak, amelyek megmondják, hogy az adott module-on belül melyik url-re melyik komponens hívódjon meg, illetve, hogy az adott module-ba milyen importok és deklarációk használhatóak. Az általam elkülönített modulok a következők:
 - Access (hozzáférés): Ez a modul általában az autentikáció, autorizáció és hozzáféréskezelés funkciót tartalmazza. Itt találhatók olyan komponensek, szolgáltatások vagy direktívák, amelyek a felhasználói hozzáférést, bejelentkezést, jogosultságkezelést és más kapcsolódó funkciókat implementálnak. Ezen belül található a bejelentkezés, a regisztráció, a jelszó visszaállítás illetve a profile aktiválás is.
 - Apps (alkalmazások): Ez a modul a konkrét alkalmazásokat vagy alrendszerket képviseli. Itt helyezhetők el a fő komponensek, amelyek az adott alkalmazásban vagy alrendszerben láthatók és használhatók lesznek. Ez lehet például a kezdőoldal, a navigációs sáv, a fő tartalomterület és más alkalmazásspecifikus komponensek. Ezen module-on belül kapott helyet a naptár (calendar), az ügyfelek (customer), a főoldal (home) illetve a szolgáltatások is (service).
 - Shared (megosztott): A Shared modul olyan elemeket tartalmaz, amelyeket több modulban lehet újra felhasználni. Ide tartozhatnak közös komponensek, szolgáltatások, direktívák vagy csővezetékek. A Shared modul célja, hogy kód újra felhasználhatóságot és egységesítést biztosítson, így a fejlesztőknek nem kell ugyanazokat az elemeket többször megismételniük. Ide tartozik az a réteg amely minden oldalon használatos és tartalmazza a különböző főleg stílusokra

vonatkozó importokat (layout), illetve ezek mellett ide tartozik még a lábléc illetve a menü (footer, navbar).

Ezen module-ok elérési útját az app-routing.module.ts fájlban deklaráltam illetve az összes modul által elérhető fő importokat és deklarációkat az app.moduel.ts-ben. [6] A továbbiakban be szeretnék mutatni egy komponens felépítését, az apps module-ból, ez lesz a szolgáltatások (service) komponens. A komponensek négy fájlból épülnek fel, de mivel template-el dolgoztam és nincs komponensenként egyedi stílus fájlom, így én csak kettőt használlok ezekből. (.html és .ts fájl) A service komponens tartalmazza a szolgáltatás megjelenítését, törlését, módosítását. A 4.9-es kód részletben bemutatom a service.component.html fájl egy részét, amely egy táblázatot jelenít meg a felhasználónak, amiben kilistázza a szolgáltatásait.

4.8. kódrészlet. Részlet a service.component.html fájlból

```
<div class="table-responsive">
    <table class="table table-centered w-100 dt-responsive nowrap"
        id="products-datable">
        <thead class="table-light">
            <tr>
                <th class="all">Service</th>
                <th>Duration</th>
                <th>Price</th>
                <th>Status</th>
                <th>Intermediate</th>
                <th style="width: 120px;">Action</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let item of categories">
                <td>
                    <p class="m-0 d-inline-block align-middle
                        font-16">{{item.name}}</p>
                </td>
                <td>
                    {{item.duration}} minutes
                </td>
                <td>
                    {{item.price}}
                </td>
                <td *ngIf="item.active == true">
                    <span class="badge bg-success">Active</span>
                </td>
                <td *ngIf="item.active == false">
                    <span class="badge bg-danger">Inactive</span>
                </td>
                <td *ngIf="item.intermediate == true">
                    <span class="badge bg-success">YES</span>
                </td>
                <td *ngIf="item.intermediate == false">
```

```

        <span class="badge bg-danger">N0</span>
    </td>
    <td class="table-action">
        <a href="javascript:void(0);"
            (click)="modify(item.id)" class="font-18 text-info
            me-2" data-bs-toggle="tooltip"
            data-bs-placement="top" aria-label="Edit"><i
            class="uil uil-pen"></i></a>
        <a href="javascript:void(0);"
            (click)="delete(item.id)" class="font-18
            text-danger" data-bs-toggle="tooltip"
            data-bs-placement="top" aria-label="Delete"><i
            class="uil uil-trash"></i></a>
    </td>
</tr>
</tbody>
</table>
</div>

```

Az ehhez a module-hoz tartozó typescript fájlban történik a *categories* lista feltöltése, a service segítségével. Ez megtekinthető a 4.10-es kód részletben, amely egy függvényt mutat be. Ez a függvény az ngOnInit() függvényben van meghívva, így az oldal inicializálásakor le fog futni.

4.9. kód részlet. Részlet a service.component.ts fájlból

```

ngOnInit(): void {
    ...
    this.getAllService();
}

getAllService(){
    this.serviceService.getAllByCompany().subscribe({
        next: (response) => {
            this.categories = response;
        },
        error: (error) => {
            this.categories = [];
        }
    })
}

```

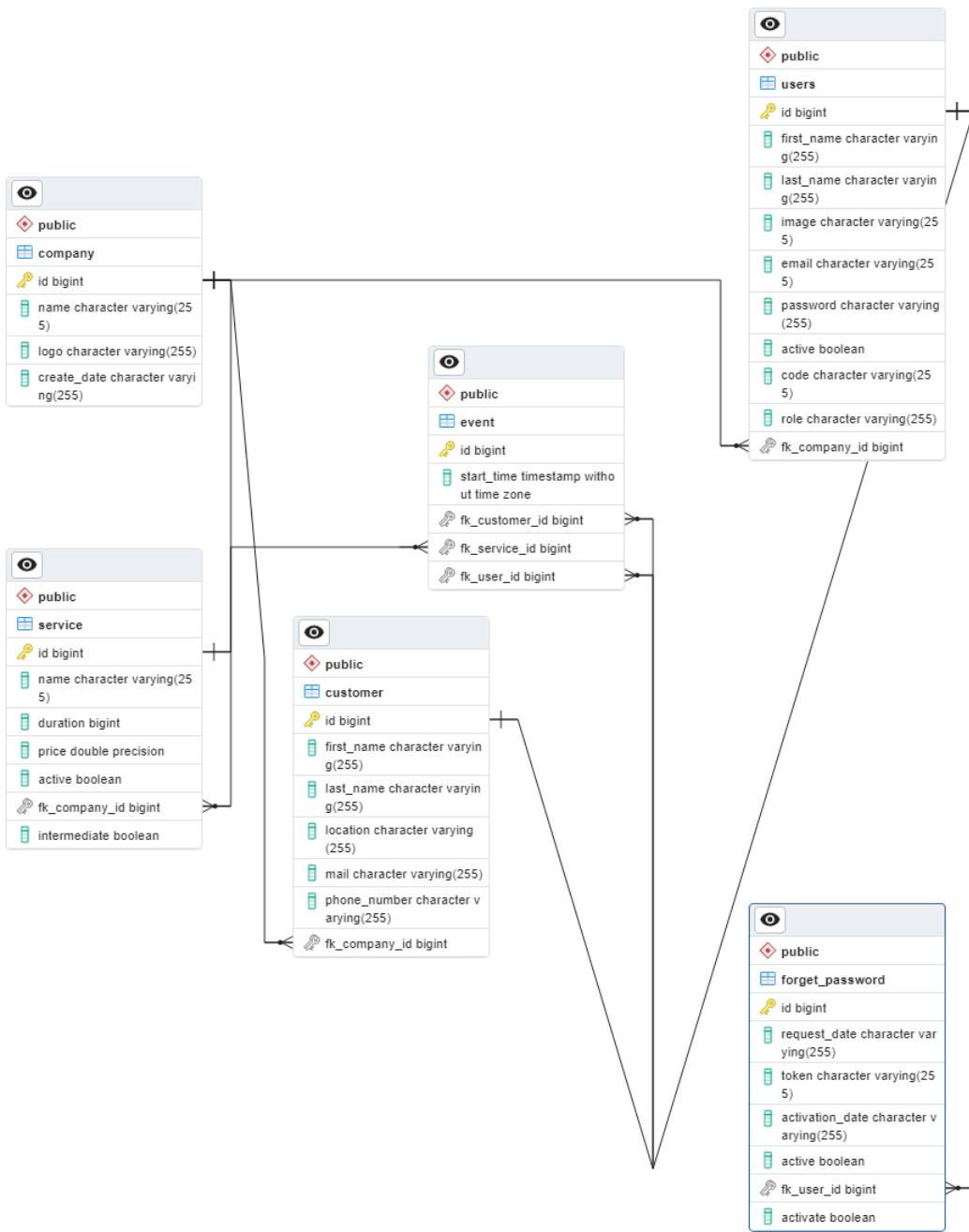
Ezen példákhoz hasonlóan vannak felépítve a komponensek, persze jelentős része ennél jóval bonyolultabb. A komponensekben használva van a sweetalert2 module, amely segítségével nagyon szép figyelmeztetéseket illetve visszajelzéseket tudtam létrehozni. Ilyen figyelmeztetés jelenik meg például törlés előtt illetve visszajelzés sikeres hozzáadás esetén.

4.1.3. Adatbázis

Adatbázisnak a PostgreSQL adatbázist használtam, amely segítségével igencsak könnyen boldogultam. A PostgreSQL egy nyílt forráskódú, relációs adatbázis-kezelő rendszer (ORDBMS). Az eredeti neve "Postgres", ami az Ingres nevéről származik, de az "SQL" kifejezést is hozzátették a nevéhez, hogy utaljon a SQL (Structured Query Language) nyelvre, amelyet használ. [8] Néhány fontos jellemző, ami miatt ezt választottam:

- Többplatformos támogatás: A PostgreSQL számos operációs rendszeren futtatható, ideértve a Windows-t, Linuxot, macOS-t és másokat. Ez lehetővé teszi, hogy a fejlesztők és üzemeltetők kiválasszák a preferált platformot.
- Adattípusok és adatkiterjesztések: A PostgreSQL számos beépített adattípust támogat, mint például számok, karakterláncok, dátumok, tömbök stb. Ezenkívül lehetőség van új adattípusok létrehozására is, amelyek speciális adatkötetelményeket fednek le.
- Biztonság: A PostgreSQL erős biztonsági mechanizmusokat kínál, például az adatbázis-szintű hozzáférés-szabályozás (ACL), SSL-titkosítás, felhasználói jogosultságok kezelése és mások. Ez lehetővé teszi az adatok védelmét és a jogosulatlan hozzáférés elleni védelmet.
- Nyílt forráskód: A PostgreSQL nyílt forráskódú projekt, ami azt jelenti, hogy ingyenesen hozzáférhető és szabadon felhasználható.

Az adatbázis kapcsolat felépítése után a projektben liquibase segítségével építettem fel a táblákat, mely alapján a SpringBoot szerver indulásakor generálódtak a táblák. Később felismertem, hogy a megfelelő JPA Repository beállításokkal az entitások képesek létrehozni ezeket a táblákat, így ezt abba hagytam és autómatikusan generálodtak az entitások alapján. A liquibase mappának és a hozzá tartozó fájloknak még mindig nyoma van a springboot projektben. Az adatbázis megtekintésére és ellenőrzésérte PgAdmin-t használtam, amely a PostgreSQL alap adminisztrációs felülete. A 4.3-as ábrán bemutatom az adatbázis szerkezetét, a táblákat és az ezek közötti kapcsolatokat.



4.3. ábra. Adatmodell diagram

4.2. SMS küldési technológia

Hosszu tanulmányokat folytattam arra, hogy milyen megoldásokat tudok használni az SMS küldésre. A legolcsóbb megoldás, amely egy API-n keresztül képes SMS-t küldeni megadott számra, 0.07 RON/SMS árban volt elérhető. Ezen fizetés elkerülésére a régi Android-os telefonomra felépítettem egy applikációt, amely bejelentkezés után automatikusan percenként lekéri a szervertől, az új elküldendő SMS-ek listáját. Ha a lista nem üres, a megadott számokra kiküldi a megfelelő SMS-eket. Ennek működése havonta pár

lejbe illetve egy folyamatosan töltőn lévő telefonba kerül. Hosszú távon érdemes lehet egy szerződés kötés egy szolgáltatóval, aki API-n keresztül biztosítani tudja az SMS-ek küldését, ezzel kiküszöbölv az esetleges eszköz hibákat. A továbbiakban bemutatom ezen SMS küldésekre alkalmas applikáció felépítését. Az applikáció két fragmensből illetve két ViewModelből áll. Miután az applikáció elindul, ellenőrzi, van-e érvényes tokenünk a SharedPreferences-ben, ha nincs, akkor be kell jelentkeznünk egy "admin" szintű felhasználóval, ha van akkor autómatikusan be leszünk lépve. Az applikáció főoldalán helyet kapott egy frissítés gomb, amellyel a backend szerver állapotát lehet lekérni, illetve két kijelző. Az egyik a következő lekérésig hátramaradott időt számítja, a másik a legutolsó lekérésben szereplő küldendő SMS-ek számát mutatja. A továbbiakban bemutatom az applikáció SMS küldéséért felelős kód részletet (4.11 kód részlet), amelyet el is magyarázok.

4.10. kód részlet. Részlet az SMS küldő applikációból

```
private val runnable = object : Runnable {
    override fun run() {
        val period:Long
        if(minutes == 0){
            period = (1*seconds*1000).toLong()
        }else if(seconds == 0){
            period = (minutes*60*1000).toLong()
        }else{
            period = (minutes*seconds*1000).toLong()
        }
        // SMS lekerdezés elkuldése az smsViewModel.getSms() függvennyel
        smsViewModel.getSms()
        // LiveData objektum figyelese a válasz eredmények kezeléséhez
        smsViewModel.result.observe(this@SMSFragment) { result ->
            if (result == LoginResult.SUCCESS) {
                arraySize.text = smsViewModel.smsArray?.size.toString()
                sendSms()
                timer.start()
            }
        }
        smsViewModel.result = MutableLiveData()
        handler.postDelayed(this, period)
    }
}
```

A run() metódusban történik az időzített műveletek végrehajtása. Az alábbi lépések történnek: A period változó inicializálása, amely a futás időtartamát határozza meg. A minutes és seconds változók alapján számítódik ki a periódus hossza milliszekundumban. Az SMS üzenetek lekérdezése történik a smsViewModel.getSms() függvény meghívásával. Az smsViewModel.result LiveData objektumának figyelése a válasz eredményének kezeléséhez. Ha a válasz LoginResult.SUCCESS értékkel tér vissza, akkor megtörténik az SMS küldése (sendSms() függvény hívása) és elindul az időzítő (timer.start()). Az smsViewModel.result LiveData objektumának újra inicializálása egy MutableLiveData objektummal, hogy újra figyelni lehessen a válaszra. Az időzítő újra elindítása a handler.postDelayed(this, period) függvényhívással. Ez a következő period időpontban fogja

újra végrehajtani a run() metódust. Ez a kódrészlet lehetővé teszi az SMS üzenetek rendszeres lekérdezését és küldését az alkalmazásban, a megadott időközönként. Az időzített műveletek végrehajtása a run() metódusban történik, és a LiveData objektum figyelése biztosítja, hogy a választ megfelelően kezeljék.

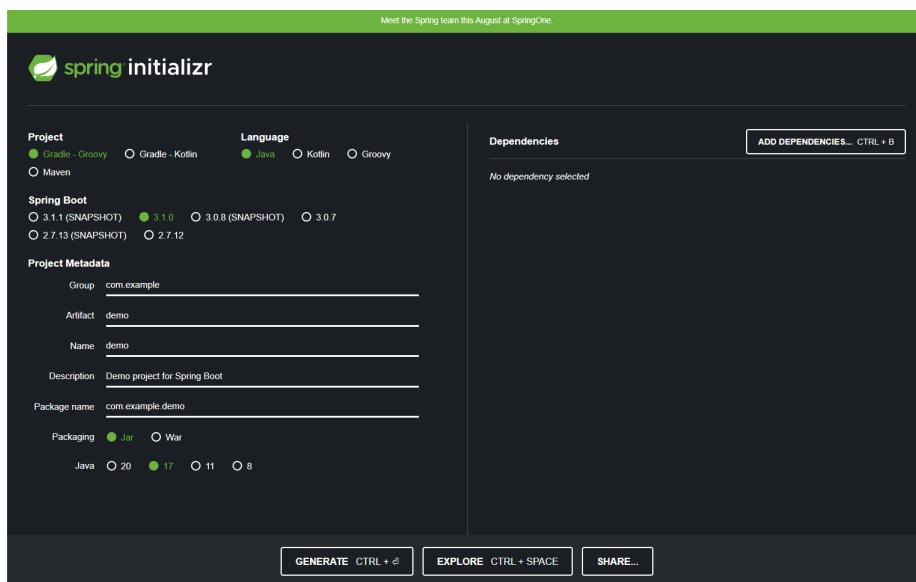
5. fejezet

Felhasznált technológiák

5.1. Programozási nyelvek és környezetek

A programozási nyelvkről nem fogok különösen sokat beszélni ebben a fejezetben, ugyanis a rendszer leírásnál már meséltem róluk illetve azok előnyeiről és arról, hogy miért is pont azt választottam.

- A backendnél alkalmazott SpringBoot technológia mivel Java alapú, ezért ennek felépítésére, szerkesztésére, módosítására a JetBrains által ingyenesen elérhető IntelliJ IDEA szoftvert használtam, hiszen az egyetemen is ezt használtuk és már részletesen ismertem. Az itt használatos nyelv a Java volt. A Spring Boot projektet a <https://start.spring.io/> oldalon inicializáltam és hoztam létre, ez után kezdtem az itt letöltött üres projektet felépíteni. A weboldal kinézete megtekinthető az 5.1-es ábrán.



5.1. ábra. <https://start.spring.io/> weboldal

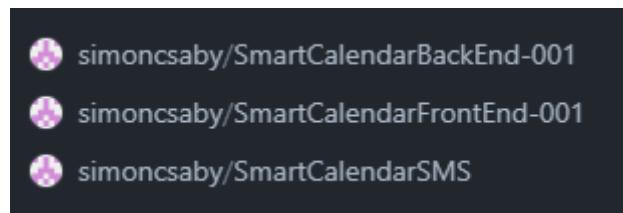
- A frontend esetén az angular-t egyszerű Visual Studio Code-ban szerkesztettem, futtattam és építettem fel. Az angular futtatásához telepítenem kellett a Node.js-t,

ez elérhető a <https://nodejs.org> oldalon. Ez után telepítettem az Angular CLI-t, amelyet terminálból tettem meg a `npm install -g @angular/cli` parancs segítségével. Ez után létrehoztam az új angular projektet a `ng new SmartCalendarFronted` paranccsal. Az angularban belül használatos technológiák a TypeScript, a HTML, a CSS, a JavaScript és a Reactive Extensions (RxJS). Ezek az alapvető technológiák segítenek az Angular keretrendszerben történő webalkalmazások fejlesztésében. A TypeScript a statikus típusosságát és a kibővített JavaScript funkcionalitását biztosítja, a HTML és a CSS pedig az alkalmazások felhasználói felületének megjelenítéséért felelős. A JavaScript a dinamikus programozási nyelv, amely a kliensoldali logika megvalósításához használható, míg az RxJS segítséget nyújt az aszinkronos adatfolyamok kezeléséhez és az esemény vezérelt programozáshoz. Ezek a technológiák együttműködve alkotják az Angular keretrendszerét, amely egy erőteljes eszköz a modern webalkalmazások készítéséhez. [2]

- Az android applikáció felépítéséhez az Android Studio nevű szoftvert használtam, amely segítségével alapvetően elég egyszerűen sikerült elkészítenem az egyetemen szerzett tudásomnak hála.

5.2. Verziókövetés

A verziókövetést igencsak fontosnak tartottam az applikáció fejlesztése során, hiszen ennek segítségével hozzáférhetünk a projektünk teljes történetéhez, beleérte az összes változtatást amit a fejlesztés során végrehajtottam. Ezen feladatra az egyik népszerű verziókövető rendszert használtam, a GitHub-ot. Mivel három különböző technológiára épülő projektről van szó, ezért külön repository-ba mentettem a frontend-et, a backend-et és az Android technológiára épülő SMS applikációt. Ezekről látható egy kép az 5.2-es ábrán.



5.2. ábra. GitHub repozitorik

6. fejezet

Felhasználói felületek bemutatása

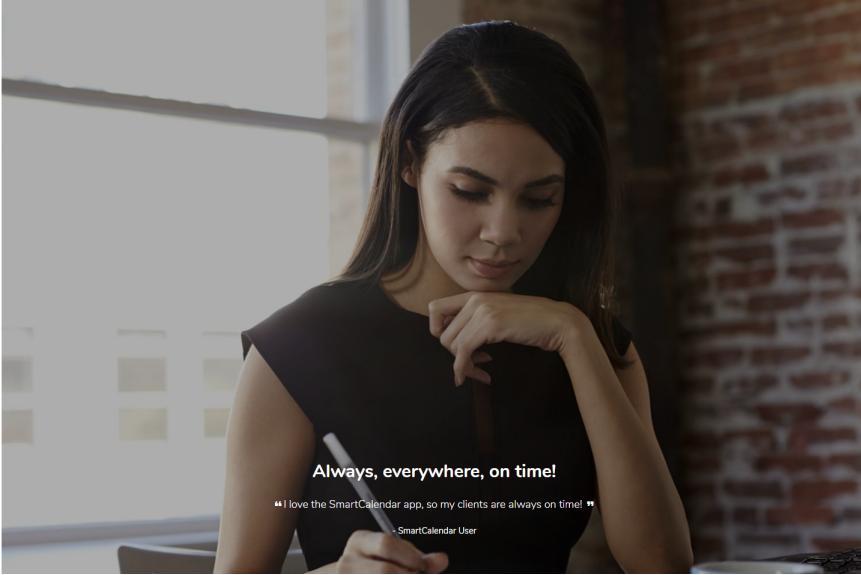
6.1. Általános felhasználó felülete

A felhasználói felület egy template-ből készült el, amelyet ingyenesen elértem az interneten. A template kiválasztása során, különösen nagy figyelmet fordítottam a letisztultságra, illetve arra, hogy egy olyan felületet tudjak vele építeni, amely egyszerű, érthető és átlátható laikus felhasználók számára is. E mellett fontosnak tartottam, hogy az információk és a funkciók könnyen elérhetők és érthetők legyenek. Az egyértelmű elrendezés, a logikus navigáció és a vizuális elemek megfelelő használata mind hozzájárulnak a felhasználóbarát felülethez. Az általam felület egy teljesen responsive weboldal, amely segítségével a felhasználói felület minden készüléken teljesen jól mutat. A felhasználói felület bemutatása során bemutatom a következőket:

- Bejelentkező felület
- Bejelentkezés utáni főoldal, néhány statisztikával (ügyfelek száma, szolgáltatások száma, programálások száma, eddig befolyt összeg száma, programálások száma lebontva hónapokra az aktuális évből)
- Szolgáltatások megtekintése táblázat szerüen, módosítás és törlés lehetőséggel
- Szolgáltatás módosítás modál ablakban
- Szolgáltatás törlésének megerősítése sweetalert2 modál segítségével
- Ügyfelek megtekintése
- Időpontok megtekintése
- Időpontok hozzáadása

A továbbiakban bemutatom a felhasználói felületet, amely segítségével kezelní tudja ügyfelüink az alkalmazáshoz regisztrált fiókját.

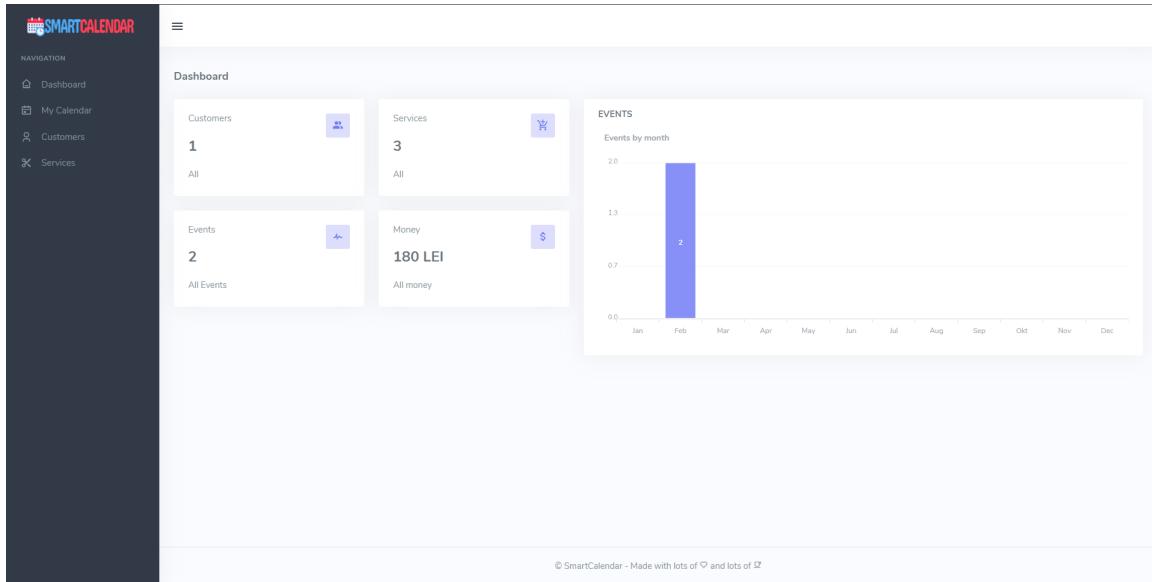
- Bejelentkezés



The image shows the SmartCalendar sign-in page on the left and a testimonial overlay on the right. The sign-in page has fields for email address and password, and a 'Log In' button. The testimonial features a woman writing in a notebook with the text 'Always, everywhere, on time!' and a quote from a user.

6.1. ábra. Bejelentkező felület

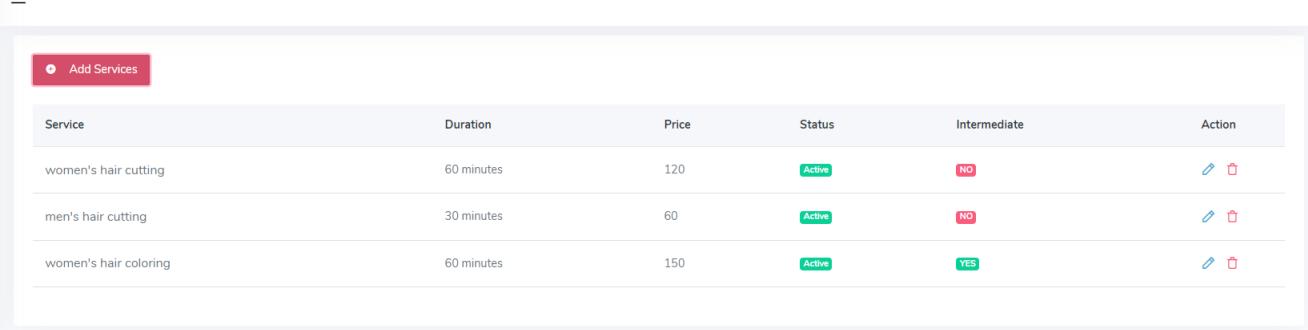
- Bejelentkezés utáni főoldal



The image shows the SmartCalendar dashboard. It includes a navigation sidebar with links for Dashboard, My Calendar, Customers, and Services. The main area displays four cards: 'Customers' (1), 'Services' (3), 'Events' (2), and 'Money' (180 LEI). A chart titled 'Events by month' shows two events in February.

6.2. ábra. Bejelentkezés utáni főoldal

- Szolgáltatások megtekintése

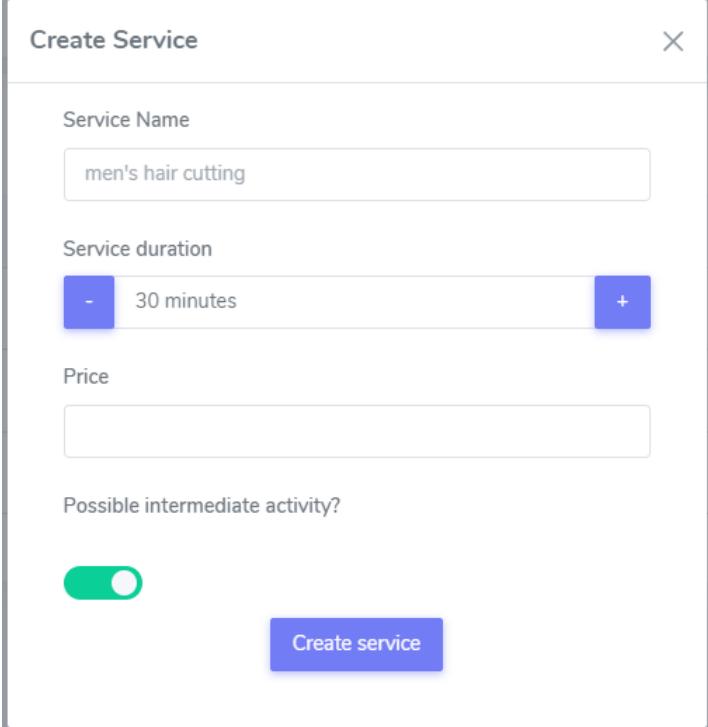


The screenshot shows a table with columns: Service, Duration, Price, Status, Intermediate, and Action. There are three rows of data:

Service	Duration	Price	Status	Intermediate	Action
women's hair cutting	60 minutes	120	Active	NO	 
men's hair cutting	30 minutes	60	Active	NO	 
women's hair coloring	60 minutes	150	Active	YES	 

6.3. ábra. Szolgáltatások megtekintése

- Szolgáltatás hozzáadása

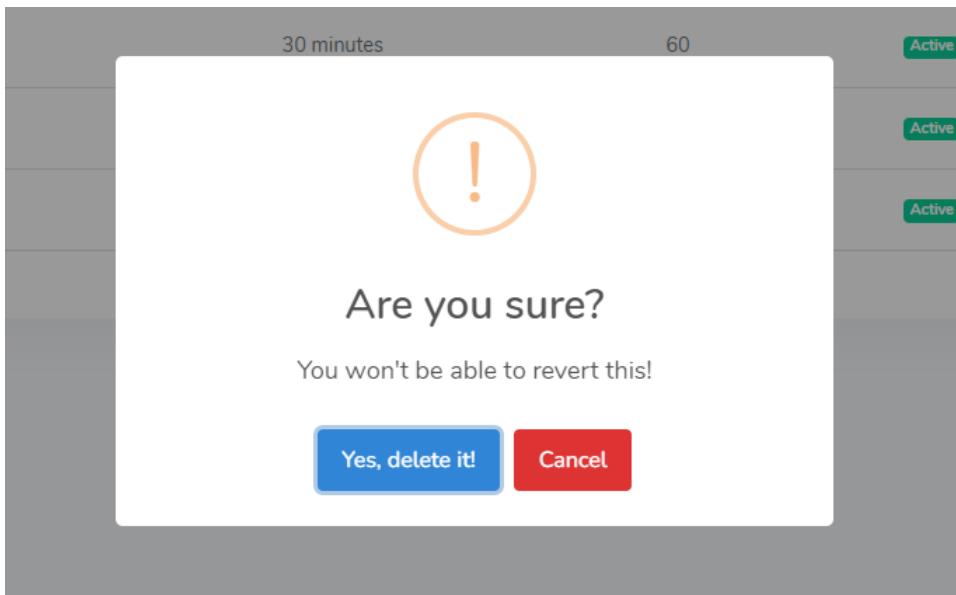


The dialog box has the following fields:

- Service Name:** men's hair cutting
- Service duration:** 30 minutes (with minus and plus buttons)
- Price:** (empty input field)
- Possible intermediate activity?**: A toggle switch is turned on (green).
- Create service**: A blue button at the bottom.

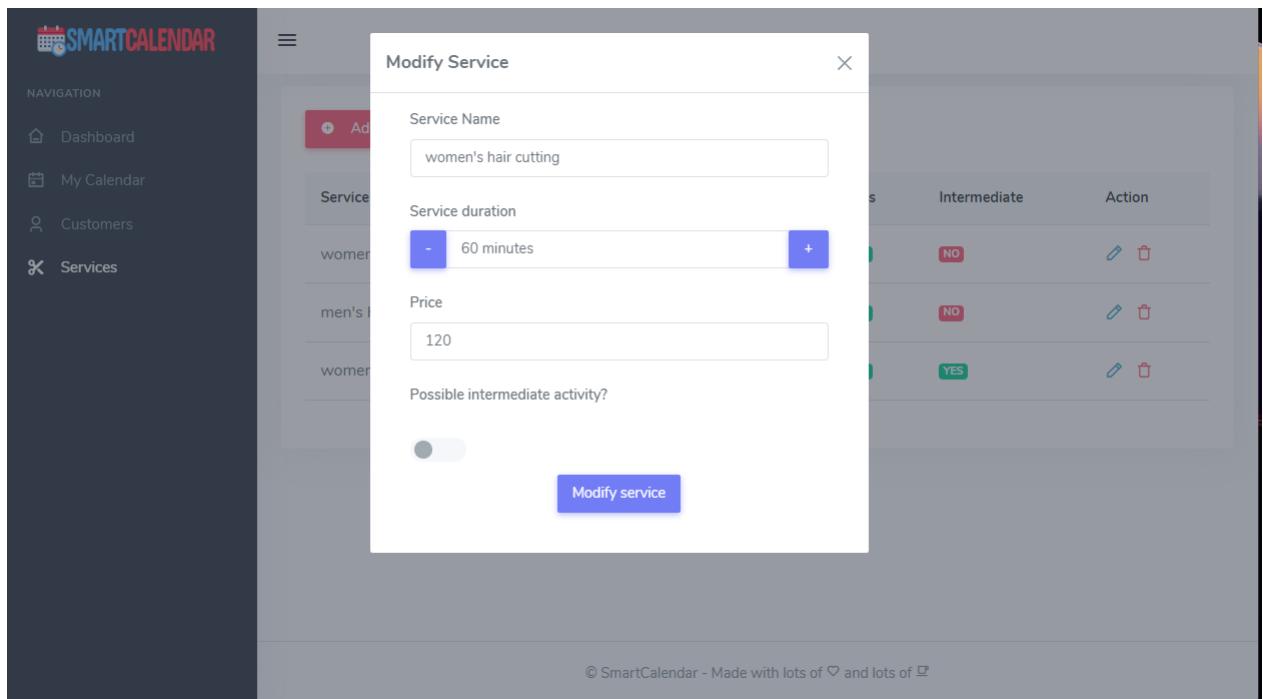
6.4. ábra. Szolgáltatás hozzáadása

- Szolgáltatás törlésének megerősítése



6.5. ábra. Szolgáltatás törlésének megerősítése

- Szolgáltatás módosítása



6.6. ábra. Szolgáltatás módosítása

- Az ügyfelek hozzáadása, törlése és módosítása nagyon hasonló az előzőleg bemutatott szolgáltatások oldalhoz, így csak a megtekintést mutatom be.

The screenshot shows the SmartCalendar application interface. On the left is a dark sidebar with a navigation menu:

- Dashboard
- My Calendar
- Customers** (highlighted)
- Services

The main content area is titled "Add Customer" and displays the following data:

Firstname	Lastname	Phone Number	Mail	Location	Action
Simon	Csaba	0754334135	scsaba555@gmail.com	Eremitu 82	

At the bottom right of the main area, there is a copyright notice: © SmartCalendar - Made with lots of ❤ and lots of ☕.

6.7. ábra. Ügyfelek megtekintése

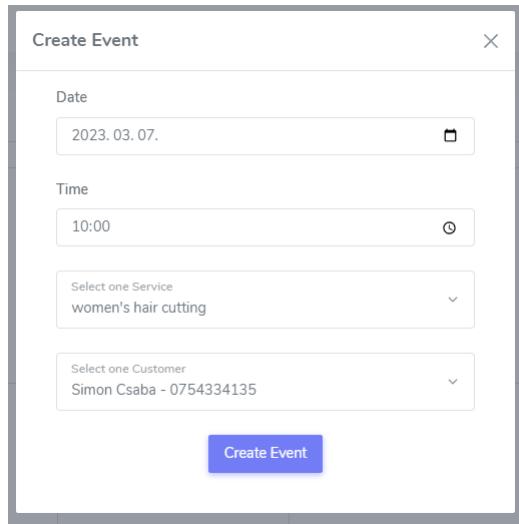
- Időpont foglalások

The screenshot shows the SmartCalendar application's monthly calendar view for March 2023. The calendar grid includes the following information:

- Weekdays:** Sun, Mon, Tue, Wed, Thu, Fri, Sat
- Dates:** 26, 27, 28, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
- Appointments:**
 - On March 10, from 16:45 to 17:00, there is a red slot labeled "men's hair cutting".
 - On March 13, from 13:35 to 14:00, there is a red slot labeled "women's hair cutting".

6.8. ábra. Időpont foglalások

- Időpont hozzáadása

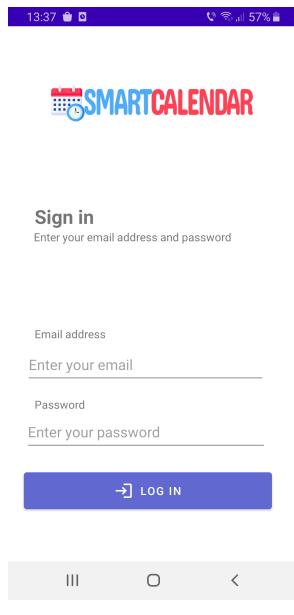


6.9. ábra. Időpont hozzáadása

6.2. SMS-t küldő applikáció felülete

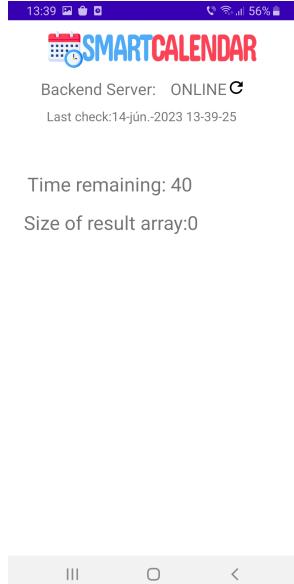
A következőkben bemutatom azon kis SMS küldéséért felelős applikációm felhasználói felületét, amely kommunikál a backend szerverrel, és ezáltal biztosítja a kliensek számára, hogy időben értesítést kapjanak. A bejelentkezés után, látható lesz egy oldal, ahol ellenőrizhető a backend szerver állapota, egy kis visszaszámláló, amely a következő lekérésig hátramaradott időt számolja másodpercben, illetve egy eredmény tömb méret, amely alapján pontosan látjuk, hogy az adott ciklusban hány SMS küldés történt.

- Bejelentkezés



6.10. ábra. Bejelentkező felület

- SMS lekéréséért felelős felület



6.11. ábra. SMS lekéréséért felelős felület

7. fejezet

Továbbfejlesztési lehetőségek

Az applikáció nagyon sok irányban fejleszthető tovább, ezekből én összeírtam néhányat, amelyet majd meg is szeretnék valósítani, nézzük őket:

- Tulajdonos és alkalmazott szintű rendszer építése
- A különböző felhasználók tudjanak különböző egyéni SMS szövegeket beállítani maguknak.
- Az SMS küldő applikációs megoldást helyettesíteni szeretném majd egy valódi szolgáltató által biztosított API alapú SMS küldő rendszerrel
- Külön telefonos applikációk fejlesztése Android-ra és IOS-re egyaránt.
- Az SMS-be legyen lemondási lehetőség, amelyet ügyfelünk saját maga engedélyez vagy sem.
- Többnyelvesítés

Összefoglaló

A diplomadolgozatom keretein belül, egy olyan alkalmazás jött létre, amely segítségével a vállalkozások illetve az egyszerű felhasználók akik rendelkeznek programálható ügyfelekkel valamilyen szolgáltatás tekintetében, értesíteni tudják az ügyfeleket az adott programálás időpontjáról SMS-ben, ezáltal elkerülhetőek bizonyos félreértesek. A rendszer felépítése az egyszerűség kedvéért web alapú, bárki elérheti a megfelelő eszköz és internet kapcsolat segítségével, illetve responsive ezáltal egy kicsit eszközfüggetlen is, bár ezen még van mit fejleszteni.

A felhasznált technológiák Java Spring Boot illetve Angular, e mellett pedig az SMS küldésre alkalmas szerverként működő applikáció Android. A verziókövetésre a GitHub-ot használtam, az alkalmazás tesztelésére pedig néhány barátomat vontam be, akik "ügyfél" szerepet kaptak, én én programáltam őket. A célkitűzésekben leírt fontos funkciók közül, sikerült megvalósítani az összest, ezáltal az alkalmazás első verzióját abszolút sikernek könyvelem el, véleményem szerint ebben a formában alkalmas már hosszútávú használatra, azt leszámítva, hogy nem volt egy éles szerveren futtatva. Ez a projekt publikusan elérhető a GitHub fiókomon, a következő linkeken:

- Java Spring Boot - Backend
<https://github.com/simoncsaby/SmartCalendarBackEnd-001.git>
- Angular - Frontend
<https://github.com/simoncsaby/SmartCalendarFrontEnd-001.git>
- Android - SMS küldő app:
<https://github.com/simoncsaby/SmartCalendarSMS.git>

Összességeben az alkalmazás fejlesztése számomra igencsak jó tapasztalat szerzés volt, e mellett pedig sokat tanultam a felhasznált technológiák segítségével.

Köszönetnyilvánítás

Ezen részen szeretnék köszönetet mondani Györfi Ágnes vezető tanáromnak, aki idejét tőlem nem sajnálva, mindenben a segítségemre állt, amiben csak szükségem volt. E mellett szeretném neki megköszönni azt is, hogy naptól és ünneptől függetlenül bármikor számíthattam a segítségére, bármikor kérdezhettem és a rendelkezésemre állt. Köszönöm szépen a sok segítséget és nagyon örültem a közös együttműködésnek!

Ábrák jegyzéke

3.1. Eset Diagram	14
4.1. A rendszer archítektúrája	20
4.2. Osztály diagram bonyolultsága	27
4.3. Adatmodell diagram	33
5.1. https://start.spring.io/ weboldal	36
5.2. GitHub repozitorik	37
6.1. Bejelentkező felület	39
6.2. Bejelentkezés utáni főoldal	39
6.3. Szolgáltatások megtekintése	40
6.4. Szolgáltatás hozzáadása	40
6.5. Szolgáltatás törlésének megerősítése	41
6.6. Szolgáltatás módosítása	41
6.7. Ügyfelek megtekintése	42
6.8. Időpont foglalások	42
6.9. Időpont hozzáadása	43
6.10. Bejelentkező felület	44
6.11. SMS lekéréséért felelős felület	44

Irodalomjegyzék

- [1] Spring boot hivatalos oldala. <https://spring.io/projects/spring-boot>.
- [2] Angular hivatalos oldala. <https://angular.io/guide/architecture>.
- [3] Spring boot – code structure. <https://www.geeksforgeeks.org/spring-boot-code-structure/>.
- [4] Spring boot annotations. <https://www.javatpoint.com/spring-boot-annotations>.
- [5] Scheduling tasks. <https://spring.io/guides/gs/scheduling-tasks/>.
- [6] Angular workspace and project file structure. <https://angular.io/guide/file-structure>.
- [7] What is postgresql? <https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/>.
- [8] Benefits of using postgresql. <https://aws.amazon.com/rds/postgresql/what-is-postgresql/>.