

Milton Friedman Egyetem

TDK-dolgozat

Drahos István

S94VGD

Gazdaságinformatikus

Lezárás dátuma: 2024. május 5.

Budapest

2024

Milton Friedman Egyetem
Informatika Tanszék

Biztonságos és modern felhasználói fiókkezelő microservice alkalmazás tervezése

Témavezető:
Dr. Bárkányi Pál, PhD
főiskolai docens

Készítette:
Drahos István
S94VGD
Gazdaságinformatikus

Budapest
2024

TARTALOMJEGYZÉK

Tartalomjegyzék.....	1
1 Bevezetés	4
2 Szakirodalom feldolgozása	6
2.1 Fogalomáttekintés és történelmi kontextus	6
2.1.1 Jelszó, azonosítás és hitelesítés definíciója.....	6
2.1.2 Történelmi kontextus	6
2.2 Azonosítás és hitelesítés az informatikában.....	7
2.2.1 Felhasználói azonosítás.....	7
2.2.2 Jelszavak használata.....	7
2.2.3 Azonosítás kiszervezése.....	8
2.2.4 Többlépcsős azonosítási lehetőségek.....	10
2.2.5 Jelszómentes megoldások	10
2.3 Mikroszolgáltatásokról.....	12
2.3.1 Általános definíciója	12
2.3.2 Célja	12
2.3.3 Előnyök és hátrányok.....	13
2.4 Hosting	13
2.4.1 On-Premise	14
2.4.2 Shared Hosting.....	14
2.4.3 Cloud Hosting	14
2.5 Biztonsági kockázatok.....	15
2.5.1 Alkalmazásoldali kockázatok	15
2.5.2 Emberi tényező okozta kockázatok	16

3	Önálló Munka	19
3.1	Szoftver specifikációja	19
3.1.1	Üzleti igények meghatározása	19
3.1.2	ISO 15408:2 Common Criteria.....	21
3.1.3	User storyk meghatározása	22
3.1.4	Használatieset-diagram	24
3.2	A szoftver tervezése	24
3.2.1	Adatbázisarchitektúra	24
3.2.2	Adatbázis modellezés.....	25
3.2.3	Szoftverfejlesztési módszertan és technológia kiválasztása	30
3.2.4	A „The twelve-factor app”-módszertan	31
3.3	Külső integráció tervezése.....	32
3.3.1	REST API tervezése	32
3.4	Hosting	35
3.4.2	Cloud architektúra diagram.....	37
3.5	IT-biztonság.....	38
3.5.1	Alkalmazásoldali biztonság	38
3.5.2	Cloudalapú védelem (WAF).....	39
3.5.3	Kódminőség	40
3.5.4	CI/CD-rendszer használata	41
3.5.5	Függőség ellenőrzés (Dependency-checking)	42
3.5.6	Statikus kódanalizálás.....	42
3.5.7	Automatizált tesztek futtatása	43
4	Összefoglalás	44
5	Irodalomjegyzék	45

6	Absztrakt.....	48
7	Abstract.....	49
8	Ábrajegyzék.....	50

1 BEVEZETÉS

Átlagemberként nagyjából két tucat különböző felhasználóazonosítást megkövetelő szolgáltatást használunk nap mint nap, életünk szerves részeként. Mindannyiunknak van bankszámlája, vannak biztosításai; járunk edzőterembe, foglalunk időpontot a borbélyszalonba, fizetünk online, részt veszünk kurzusokon, járunk konferenciákra, végzünk szabadidős tevékenységeket. Ezeknek a közös metszete, hogy mind megkövetel egy saját felhasználói fiókot. Szoftverfejlesztőként ezeknek a szerteágazó ügyfélfiókoknak a biztonságos, egységes kezelése, folyamatos frissítése és technológiailag naprakészen tartása maga a rémálom. Hiszen minden szolgáltató egyedi megoldásokat alkalmaz (nem kizárólag, de például az ügyfélazonosításban), és más funkciókkal rendelkezik (van, amely használ többlépcsős hitelesítést, van amelyik nem). Ahhoz, hogy ezt a rémálmot szertefoszlassuk, szükséges egy olyan platform, amelyet a fent felsorolt szolgáltatók bármelyike, domaintól függetlenül igénybe vehet, és amellyel a felhasználói fiókok kezeléséhez szükséges funkcionalitást ki tudja szervezni, hogy ne kelljen házon belül folyamatosan frissítenie, karban tartania és új funkciókkal ellátnia. Az általam tervezni kívánt webalkalmazás a fenti feladatokra úgy adna megoldást, hogy közben erősebbé teszi a felhasználói fiókok védelmét, és egy egyszerű REST API használatával csökkenti a szoftverek, alkalmazások fejlesztési idejét. Ahogyan az a TDK dolgozatomban olvasható, a megoldás védelmet nyújt a leggyakoribb sebezhetőségek ellen, skálázható, megfelel az ISO 15408:2 funkcionális követelményeinek, és könnyen bővíthető további funkciókkal.

TDK dolgozatom célja az is, hogy a Milton Friedman Egyetem gazdaságinformatikus kurzusának lezárásaként a hallgatott tárgyaim közül a lehető legtöbbet átívelő, komplex feladatmegoldást mutassak be. TDK dolgozatomban a fent vázolt problémakörre vonatkozóan meg kívánom jeleníteni az adatbáziskezelés témaköréből az adatmodellezés, az adatparticionálás főbb korszerű megoldásait, a szoftverfejlesztési technológiák megfelelő alkalmazásának fontosságát, az információrendszerek tervezését, a projektmenedzsment fontosabb lépéseit, valamint az IT-biztonság lényegességét és időszerűségét. TDK dolgozatom egy hipotetikus szoftverfejlesztési projekt is egyben, amely egy „Software-as-a-Service”-t valósít meg és amelyben hangsúlyos szerepet kap az információmenedzsment több aspektusa is.

TDK dolgozatom részeként ugyancsak megtalálhatóak az UML use case diagramok, a kiválasztott felhőalapú szolgáltatások főbb előnyei.

2 SZAKIRODALOM FELDOLGOZÁSA

2.1 Fogalomáttekintés és történelmi kontextus

2.1.1 Jelszó, azonosítás és hitelesítés definíciója

A felhasználói azonosítás témakörében az alábbi fogalmak ismerete kiemelten fontos.

2.1.1.1 Jelszó

„Vhová tartozás igazolására az illetékesektől megállapított s csak a beavatottakkal közölt szó v. név, amelynek kimondása v. közlése az összetartozás jele.” [1]

„(katonaság) Őrszolgálatban haszn. szó v. név, amelynek kérdésével, ill. megmondásával az őt ellenőrzi, nem ellenséges v. illetéktelen személy közeledik-e feléje.” [1]

2.1.1.2 Azonosít

“tárgyas ige -ott, -son; -ni v. -ani <Két dolgot v. személyt> azonosnak, egyezőnek, egy(enlő)nek, közösnek tart, vall, ismer el.”

”a. (ritka) <Két dolgot> azonosságuk szempontjából összehasonlít, összevet.” [2]

2.1.1.3 Titkosítás (rejtjel)

„főnév (katonaság, hivatalos) Titkos írásban haszn. közönséges betű v. szám, amely egy megállapított és titokban tartott kulcs (4) alapján jelent bizonyos betűt.”[3]

2.1.2 Történelmi kontextus

A jelszavak és a különböző azonosítási módszerek alkalmazása a történelem során mindig is kulcsszerepet játszott a bizalmas információk és területek védelmében. Röviden szeretném összegezni, hogy hogyan fejlődött és változott az azonosítás, titkosítás és jelszóhasználat az ókortól az informatika megjelenéséig.

A jelszavak használatát már az ókori civilizációk is alkalmazták. Például, a római légiók használtak jelszavakat a katonák azonosítására és az információ biztonságos átadására. A középkorban, a lovagrendek és egyházi szervezetek szintén használtak jelszavakat és titkos jeleket, hogy biztosítsák a találkozókat és az átadott információk bizalmasságát.

A technológiai és tudományos fejlődéssel párhuzamosan a titkosítási és azonosítási módszerek is komplexebbé váltak. A titkos levelezés kódolási és dekódolási módszerei, például

a Caesar-rejtjel, vagy a mechanikus titkosítási eszközök, mint például a Jefferson-cső egyértelműen példázzák az információvédelem fontosságát a reneszánsz idején.

A 19. és 20. században a technológiai fejlődés hozta magával az elektromechanikus és elektronikus kódoló eszközök, mint például az Enigma, fejlesztését, melyek már összetettebb és nehezebben megfejthető titkosítási módszereket kínáltak. A kémkedés és a katonai hírszerzés új szintre emelkedett, aminek eredményeként az információvédelem még kritikusabbá vált.

Az informatika megjelenése a 20. század közepén radikálisan megváltoztatta a jelszavak és az azonosítási módszerek használatát. A digitális technológia lehetővé tette, hogy az azonosítás már nemcsak emberek, hanem gépek között is megtörténjen. Az elektronikus rendszerek fejlődésével a jelszóvédelem és az azonosítás számos új aspektusa vált fontossá.

2.2 Azonosítás és hitelesítés az informatikában

2.2.1 Felhasználói azonosítás

A felhasználói azonosítás kritikus fontosságú eleme az informatikai rendszereknek, különösen napjainkban, amikor az adatvédelem és a bizalmas információk megfelelő kezelése nélkülözhetetlen. Az informatikai rendszerekben történő azonosítás alapvetően a felhasználó valódiságának igazolását jelenti, és már az informatika korai szakaszától kezdve központi szerepet tölt be az iparágban.

2.2.2 Jelszavak használata

A digitális információk, beleértve a felhasználói jelszavakat is, tárolása során az adatbiztonság kiemelten fontos szempont. Az adatbázisokban tárolt jelszavak védelme létfontosságú, mivel a támadók számára ezek különösen értékes célpontok, amelyek lehetővé tehetik számukra a felhasználói fiókokkal való visszaélést.

„A jelszavak atyjának, a 2019-ben elhunyt, Fernando Jose Corbatót tekintjük. Corbato az 1950-es években az MIT-nál dolgozott, ahol az első valós időben működő digitális számítógép létrejött a „Project Whirlwind” keretein belül. Ez egy második világháború alatt indult program, amelyben az MIT az Egyesült Államok haditengerészetével együttműködve egy univerzális repülésszimulátort fejlesztett ki. Az eszköz sikere 1951-ben az MIT Lincoln Laboratory

létrehozásához vezetett. Akkoriban a számítógépek óriásiak, drágák és lassúak voltak. Az a tény, hogy csak néhány egyetemnek és nagyvállalatnak volt valódi számítógépe, rendkívül megnövelte az igényt a használatára, a gépidőre. Ennek az igénynek a kielégítésére az olyan egyetemek, mint az MIT, bevezették az időmegosztásos operációs rendszereket, mint a Compatible Time-Sharing System-et (CTSS), amely lehetővé tette több felhasználó számára, hogy megosszák az egyetlen számítógép által biztosított erőforrásokat. Corbato fejlesztette ki az MIT-n a CTSS szabványt, amit aztán évtizedekig használtak ilyen célokra. Lényegében arról van szó, hogy távolról bárki, akinek volt jogosultsága, hogy hozzáférjen a számítógéphez, beküldhette a számításait. A beérkezett igényeket sorrendbe állították, és amikor készen volt a számítással, a gép visszaküldte az eredményeket. Ezzel sokkal jobban lehetett optimalizálni az erőforrásokat. Corbato már akkor gondolt arra, hogy az egyes kutatók ne nézhessenek bele mások anyagaiba. Ezért a valódi világból átemelte a számítógépes környezetbe a jelszavak használatát. A mainframe-re kapcsolódó terminálokra egyszerű jelszavakat állítottak be az egyes felhasználókhoz. Bár a jelszavak használata egyszerű és nagyszerű újításnak számított, Corbato nem helyezett túl nagy hangsúlyt a kibervédelemre. A terminálok beállított jelszavak rövidkeek voltak, így a túl egyszerű jelszavakat könnyedén ki lehetett találni, ám a jelszavak tárolása sem történt biztonságosan.

1962-ben, Allan Scherr PhD. kutató úgy gondolta, hogy a heti 4 órás terminálidő kevés neki. Úgy döntött, hogy nemes egyszerűséggel kinyomtatja a gépen tárolt jelszavakat. Küldött egy nyomtatási parancsot a számítógépnek, hogy nyomtassa ki a jelszavakat tartalmazó fájlt. Mivel a jelszó-fájl nem védte jelszó, így reggelre már ott volt a lista kinyomtatva az összes felhasználó, összes jelszával. Ezzel lényegében korlátlan időt tudott kérni magának, más kollégák gépidejének rovására.” [4]

Tehát a jelszó feltalálása és az első jelszólopás is ugyanabban a laborban történt meg. Ez szimbolikusan is megmutatta a megoldás mellett az új problémát is. A következtetés egyértelmű volt: a jelszavakat egyszerű szöveges (plaintext) fájlban tárolni nem túl jó ötlet.

2.2.3 Azonosítás kiszervezése

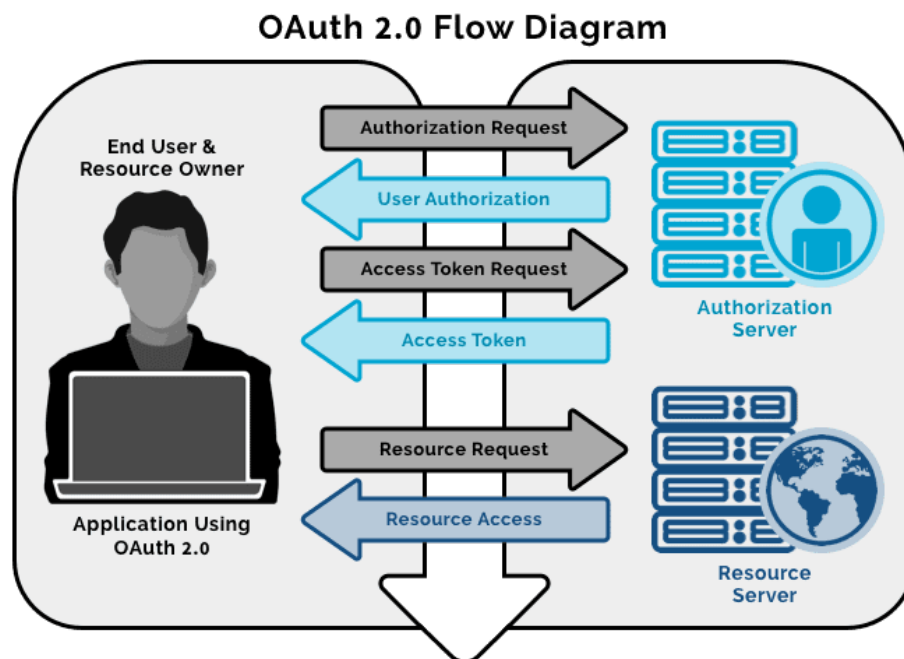
A külső fél által végzett azonosítás egy olyan folyamat, ahol egy szervezet vagy alkalmazás nem saját maga végzi az azonosítási és hitelesítési feladatokat, hanem egy külső szolgáltatóra bízta ezt a feladatot.

2.2.3.1 Single Sign-On (SSO)

Az SSO lehetővé teszi a felhasználók számára, hogy egyetlen hitelesítési folyamaton keresztül hozzáférjenek több, egymástól független rendszerhez vagy alkalmazáshoz. Az SSO jelentősen egyszerűsíti a bejelentkezési folyamatot, mivel a felhasználóknak nem kell külön-külön megjegyeznie minden egyes szolgáltatás jelszavát. SSO-val leggyakrabban vállalati környezetben találkozhatunk.

2.2.3.2 OAuth

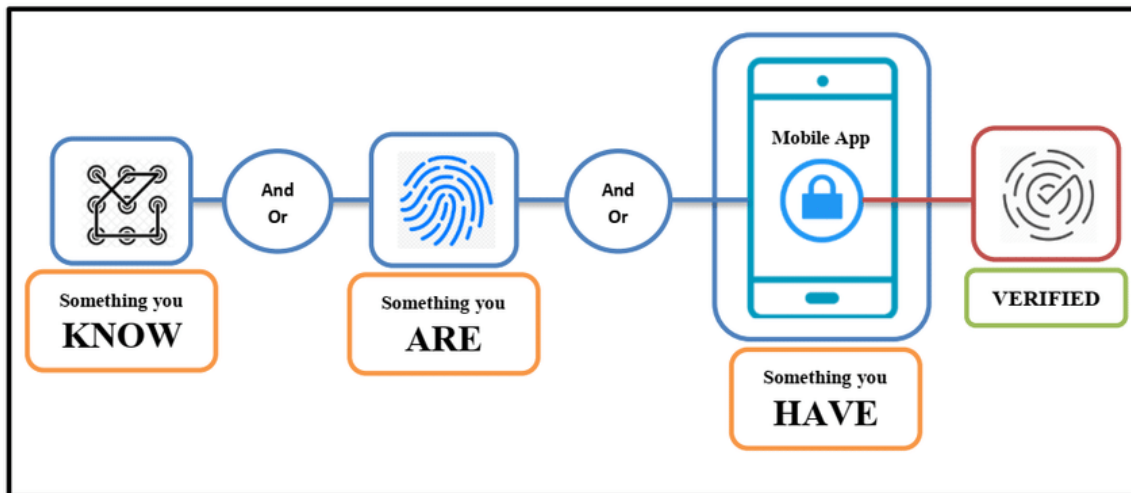
Az OAuth egy nyílt szabvány, amelyet a hitelesítésre és a jogosultságok delegálására használnak internetes alkalmazásokban. Lehetővé teszi a felhasználók számára, hogy hitelesítő információk megadása nélkül engedélyezzék harmadik fél által vezérelt alkalmazásoknak a hozzáférést bizonyos információikhoz, amelyeket egy szolgáltatás tárol. Erre a leggyakoribb példa a “Sign in with Facebook”, vagy a “Login with Google” típusú szolgáltatások. Gyakran összekeverik az SSO-val, mert a folyamat a felhasználók avatatlan szemének kísértetiesen hasonló.



1. ábra - OAuth alapú azonosítás működése [5]

2.2.4 Többlépcsős azonosítási lehetőségek

A jelszóalapú azonosítás önmagában sok kihívással szembesül a biztonság terén, különösen a széles körű támadások, és a felhasználó oldali gyenge jelszókezelésnek köszönhetően. Ennek javítása érdekében alakultak ki többszempontos azonosítási lehetőségek (Multi-Factor Authentication, MFA)



2. ábra - Többlépcsős azonosítás működése [6]

A többszempontos azonosítás egy vagy több további védelmi réteg bevezetésével növeli a rendszerek biztonságát. Az MFA különböző faktorok kombinációját használja, például „valamit, amit a felhasználó tud” (jelszó), „valamit, amit a felhasználó birtokol” (okostelefon vagy egy másik eszköz), vagy „valamit, ami maga a felhasználó” (biometrikus adatok). Ezzel a módszerrel, még ha egy jelszó illetéktelen kézbe kerül is, a rendszer továbbra is védelmet nyújt az illetéktelen hozzáférés ellen.

2.2.5 Jelszómentes megoldások

Az informatikai biztonság terén a jelszómentes azonosítás, más néven jelszó nélküli bejelentkezés (Passwordless Authentication), a felhasználói élmény és a biztonság finom egyensúlyát keresi. A jelszómentes autentikáció egy olyan rendszer, mely nem kér jelszót a felhasználótól a bejelentkezés során, és helyette más azonosítási módokat alkalmaz.

2.2.5.1 Hardveres megoldások

2.2.5.1.1 *Biometrikus Azonosítás*

Az ujjlenyomat- és arc- vagy íriszfelismerés a leggyakrabban használt biometrikus azonosítási formák. Kényelmesek és gyorsak, de érdemes megfontolni az adatvédelmi és a biztonsági kérdéseket is. Például azt, hogy az azonosítás során felhasznált biometrikus személyes adat, hol (milyen eszközön) és hogyan (milyen technológiával) van eltárolva, és ki felelős ezeknek az adatoknak a sértetlenségért. Napjaink uralkodó szakmai trendje, hogy amennyiben az általam, mint szoftverfejlesztő által egy folyamatban szükségesnek ítélt egy jelszómentes biometrikus azonosítási metódus, akkor azt nem fejlesztem le az elejéről, hanem igénybe veszem az adott platformok már létező technológiáját (iOS Secure Enclave, Windows Hello, Android TEE), amelyek az adott eszköz környezetében szabványosítva és biztonságosan kezelik a biometrikus azonosítást.

2.2.5.1.2 *Smart Cardok*

Ezek a kártyák egyfajta fizikai hitelesítést biztosítanak egy beépített chip segítségével, ami azonosítja a birtokost.

2.2.5.1.3 *USB Security Key-k*

Ugyancsak fizikai biztonsági eszközök, ilyen például a YubiKey, melyek lehetővé teszik az azonosítást egy USB-interfész segítségével.

2.2.5.2 Szoftveres Megoldások

2.2.5.2.1 *Mobiltelefon alapú azonosítás*

Ezen módszerben a felhasználó mobiltelefonja szolgál az azonosítás eszközeként, gyakran egy alkalmazás vagy egy beépített biztonsági funkció, például a biometrikus azonosítás formájában.

2.2.5.2.2 *E-mail vagy SMS alapú azonosítás*

Ebben az esetben a felhasználó egy linket vagy kódot kap e-mailben vagy SMS-ben, ami az azonosítás eszközeként szolgál.

2.2.5.3 Jelszómentes megoldások előnyei

- Felhasználóbarátság - Sok esetben egyszerűbb és gyorsabb a felhasználó számára.

- Megnövelt biztonság - Azokban az esetekben, ahol a jelszó helyettesítésére szolgáló eszközök nehezebben illetékteleníthetők vagy reprodukálhatóak (pl. biometrikus adatok), a rendszer biztonságosabb lehet.
- Kevesebb elfelejtett jelszó - Kevesebb esély van arra, hogy a felhasználóknak helyre kell állítaniuk az elfelejtett jelszavaikat, ami csökkenti a felhasználói frusztrációt és az IT-támogatás terheit.

2.2.5.4 Jelszómentes megoldások hátrányai

- Adatvédelem - A biometrikus adatok tárolásával kapcsolatban adatvédelmi aggályok léphetnek fel.
- Technikai korlátozottságok - Egyes megoldások speciális hardvert vagy szoftvert igényelnek, amely nem mindig elérhető vagy nem mindig kompatibilis minden eszközzel.
- Sérülékenységek - Bár a jelszómentes megoldások más típusú támadásoknak lehetnek ellenállóbbak, ezen megoldásoknak is vannak specifikus sérülékenységeik és kockázataik.

2.3 Mikroszolgáltatásokról

2.3.1 Általános definíciója

A mikroszolgáltatás-architektúra lényege az, hogy az alkalmazás a felhasználó számára egy nagy alkalmazásként látszik, azonban a háttérben több különálló folyamat, mondhatni minialkalmazás dolgozik azon, hogy a felhasználó által elvégezni kívánt művelet a leggyorsabban végbemenjen. Ebben az esetben ezek a különálló szolgáltatások valamilyen egyszerű mechanizmust használva kommunikálnak egymással.

A mikroszolgáltatás-architektúra hosszú távon jelentős költségeket spórolhat meg fejlesztési és terméktámogatási oldalon, azonban üzemeltetési szempontból költségesebb, mint a klasszikus monolitikus alkalmazásmodell/architektúra.

2.3.2 Célja

„Egy klasszikus monolitikus rendszernél ha bele akarunk avatkozni bármilyen folyamatba, hogy változtassunk rajta (pl. egy új komponenst akarunk elhelyezni a szoftverben), az

gyakorlatilag a teljes architektúrát, a teljes szoftvert érinteni fogja. Emiatt a bevezetés mindig nehézkes, hiába csak apró módosításokat akarunk végrehajtani, hiszen rengeteg következményt von maga után minden ilyen lépés.

A microservice-architektúránál viszont ahelyett, hogy mindent egy nagy szoftverrel próbálnánk megoldani, egy sor különböző apró szoftverek együttműködéseként hozzuk létre a rendszert. Ennek a logikájára egy jó párhuzam, ha arra gondolunk, hogy a focicsapatokban is különböző képességű játékosok játszanak különböző pozíciókon, és nem egyfajta játékosból, mondjuk a csatárból van 11 darab. Könnyen belátható, hogy egy ilyen rendszer nagyobb fokú rugalmasságra képes, és másfajta lehetőségei vannak.” [7]

2.3.3 Előnyök és hátrányok

2.3.3.1 A mikroszolgáltatás-architektúra előnyei

- Technológiai vagy szállítói függőség elkerülhető, használható akár szolgáltatásonként eltérő technológia
- A kód és a szolgáltatások gyorsan átláthatók, megérthetők
- Kisebb alkalmazásegységek könnyebben tesztelhetők, kiadhatók
- Hiba esetén nem omlik össze a teljes alkalmazás
- Könnyen valósítható meg a skálázhatóság

2.3.3.2 A mikroszolgáltatás-architektúra hátrányai

- Szolgáltatások közötti kommunikáció komplex, külön tervezni szükséges velük
- Több szolgáltatás több erőforrással jár
- A teljes alkalmazás tesztelése nehezebb a különálló szolgáltatások sokasága miatt
- Az alkalmazás fejlesztése időigényesebb

2.4 Hosting

A mikroszolgáltatások futtatásának számos módja létezik, amelyek különböző infrastrukturális és menedzsment szempontok szerint oszlanak meg. A választott módszer nagyban függ a projekt követelményeitől, mint például a skálázhatóság, megbízhatóság, költségvetésétől és a fejlesztőcsapat tapasztalatától. Az alapvető lehetőségek közé tartozik az On-Premise, a Shared Hosting és a Cloud Hosting. Mindegyik kategórián belül többféle technikai megoldás érhető el.

2.4.1 On-Premise

Az On-Premise megoldásoknál a szervezet saját infrastruktúráján futtatja a mikroszolgáltatásokat, ami teljes kontrollt biztosít a fizikai szerverek és hálózati konfigurációk felett.

- **Dedikált Szerverek:** A szervezet saját szervereket üzemeltet, amelyek kifejezetten az ő mikroszolgáltatásaik futtatására vannak konfigurálva.
- **Virtualizáció (VPS bérlet):** Virtuális magán szerverek (VPS) létrehozása az On-Premise szervereken belül, amelyek lehetővé teszik a különböző mikroszolgáltatások izolált futtatását.
- **Konténerizáció:** Technológiák, mint a Docker és Kubernetes, segítségével konténereket hozhatunk létre, amelyek könnyen telepíthetők és skálázhatók az On-Premise infrastruktúrában belül.[8]

2.4.2 Shared Hosting

A Shared Hosting egy megosztott erőforrásokkal rendelkező hosting megoldás, ami általában kisebb projektekhez vagy olyan alkalmazásokhoz ajánlott, ahol a teljesítmény nem kritikus tényező.

- **Web Hosting Szolgáltatások:** Több webalkalmazás és szolgáltatás futtatása egy megosztott szerveren. Ez a megoldás korlátozott testreszabást és kevesebb erőforrást kínál.

2.4.3 Cloud Hosting

A Cloud Hosting lehetőséget ad a mikroszolgáltatások rugalmas, skálázható és megbízható futtatására a felhőben, fizikai infrastruktúra kezelése nélkül.

- **VPS Bérlet (IaaS):** Infrastruktúra mint szolgáltatás (IaaS) megoldások, mint az AWS EC2 vagy az Azure Virtual Machines, ahol virtuális szervereket bérelhetünk és teljes mértékben konfigurálhatunk.
- **Managed Cloud Services (PaaS):** Platform mint szolgáltatás (PaaS) megoldások, mint az AWS Elastic Beanstalk vagy Azure App Services, ahol a felhőszolgáltató kezeli az infrastruktúrát és a felhasználók csak az alkalmazásukat telepítik és futtatják.

- **Serverless Services (FaaS):** Funkció mint szolgáltatás (FaaS) megoldások, mint az AWS Lambda vagy Azure Functions, ahol a kód futtatására kerül sor események alapján, és nem szükséges az infrastruktúrával vagy a szerverekkel foglalkozni. Ez a megközelítés különösen jól illeszkedik a mikroszolgáltatások architektúrájához, ahol minden egyes funkció különálló egységként kezelhető.

Mindegyik hosting típusnak és technológiának megvannak a maga előnyei és hátrányai, és a választás során figyelembe kell venni a projekt egyedi követelményeit, a költségvetést, a skálázhatóságot, a rendelkezésre állást és a biztonságot.[8]

2.5 Biztonsági kockázatok

2.5.1 Alkalmazásoldali kockázatok

2.5.1.1 Jelszavak megfelelő titkosítása

A jelszavak nyers szöveggént, titkosítatlan formában történő tárolása katasztrofális következményekkel járhat. Ha egy támadó hozzáfér egy ilyen adatbázishoz, azonnal használhatja vagy másolhatja a felhasználói jelszavakat, ami az adatok, a számlák és egyéb erőforrások illetéktelen hozzáféréséhez vezethet.

2.5.1.2 Megfelelő titkosítás fontossága

A jelszavak biztonságos tárolásának egyik bevett módszere a hashelés és a salt használata, a teljesen biztonságos megoldás azonban ezek együttes használata.

2.5.1.3 Hashelés

A hashelés során a jelszót egy egyirányú függvényen keresztül "áteresztik", amely egy fix hosszúságú, egyedi karaktersorozatot, azaz hash-t generál. Mivel a hashfüggvényeket tervezésük során úgy alakították ki, hogy nem lehetséges a kimenetből az eredeti bemenetet visszaállítani, ezért a támadók számára a hash-értékek önmagukban nem használhatók felhasználói jelszóként. Az azonosítás során a felhasználó által megadott jelszó is hashelésre kerül, és az alkalmazás csak a két hasht hasonlítja össze.

2.5.1.4 Salt

A salt egy véletlenszerűen generált kódsor, amelyet a jelszóhoz adnak hozzá, mielőtt azt a hashelő függvényen átvezetik. Ennek a módszernek köszönhetően, még azonos jelszavak esetén

is különböző hash-értékek keletkeznek, megnehezítve a támadók munkáját, például az úgynevezett "rainbow-table" támadások ellen.

2.5.1.5 Iteratív hashelés

A hashelési folyamatot többször is megismételhetjük, hogy tovább növeljük a jelszavak védelmét a brute-force támadásokkal szemben.

2.5.1.6 Pepper használata

Egy további védelmi rétegként bevezethetünk egy „peppert” is, ami egy további, globális, rejtett érték, amit a salt-hoz hasonlóan adnak hozzá a jelszóhoz hashelés előtt, de annak tárolási módja és kezelése eltér. [9]

2.5.2 Emberi tényező okozta kockázatok

A technikai megoldások mellett nem szabad elfelejtenünk, hogy az emberi tényező is kulcsfontosságú, a felhasználóknak is tudniuk kell a jelszóbiztonsági alapelveket.

Az emberek gyakran nem tudják, milyen kockázatokkal jár, ha gyenge jelszavakat használnak, vagy ugyanazt a jelszót több helyen is használják.

Ha egy jelszóval több fiók is hozzáférhető, a támadóknak egyetlen jelszót kell csak kitalálniuk vagy megszerezniük ahhoz, hogy hozzáférjenek a felhasználó további fiókjaihoz. Az egyedi jelszavak biztosítása minimálisra csökkenti a kockázatot. Ebben az esetben ha egy jelszó kompromittálódik, a többi fiók biztonságban marad.

A sokasodó jelszavak menedzseléséhez és biztonságos tárolásához jelszómenedzsment eszközök használata ajánlott, melyet egy későbbi fejezetben részletezek.

2.5.2.1 Kibertámadások

Manapság nincs olyan informatikai rendszer, amelyet ne próbálnának feltörni. A felhasználó adataira több millió hacker vadászik, éppen ezért, ha informatikai rendszert tervezünk, kalkulálnunk kell azzal, hogy ezt a rendszert meg fogják próbálni feltörni.

Az alábbiakban bemutatom a leggyakrabban használt módszereket, és az ellenük felhasználható védekezési lehetőségeket is.

2.5.2.1.1 Brute Force támadás

Ez a módszer a felhasználónevek és jelszavak kitalálására irányul, ismételt próbálkozásokkal, gyakran automatizáltan, szoftver segítségével. Az ilyen támadások ellen úgynevezett rate-

limiting technikával lehetséges. Amennyiben egy adott felhasználó rövid időn belül többször elrontja a bejelentkezését, a fiókjának hozzáférhetősége korlátozásra kerül.

2.5.2.1.2 *Dictionary Attack*

Ez hasonlít a Brute Force támadáshoz, de itt a támadó egy előre definiált "szótárban" található jelszavakat próbálja meg felhasználni a sikeres autentikációhoz.

2.5.2.1.3 *Phishing támadások*

Ezek során a támadó adathalász e-maileket és/vagy hamis weboldalt használ, hogy megszerezze a felhasználók bejelentkezési adatait. Ezzel a módszerrel a támadó úgy juthat be egy által megcélzott rendszerbe, hogy valójában valós adatokat ad meg, és a felhasználó nevében jelentkezik be. Védekezni az ilyen támadások ellen nem könnyű, hiszen a felhasználót célozzák. Azonban az esetek többségében a támadó gyakran több ezer kilométerre lévő helyszínről próbál meg bejelentkezni az általa megtámadott felhasználó fiókjába, így a gyanús bejelentkezéseket érdemes monitorozni, és automatikusan egy második ellenőrzési faktorrall ellátni.

2.5.2.1.4 *Credential Stuffing*

Ebben az esetben a támadó már korábban kiszivárgott felhasználóneveket és jelszavakat használ fel más rendszerekben való bejelentkezésre. Ez ellen is hatásos védelem, ha szűrjük a gyanús bejelentkezéseket.

2.5.2.1.5 *Man-in-the-Middle támadás*

Ebben az esetben a támadó az adatok és a felhasználó közé ékelődik be, így hozzáférhet a kommunikáció során küldött érzékeny információkhoz. Általában ezeket a támadásokat SSL-titkosítás, valamint különféle biztonsági tanúsítványok meglétének ellenőrzésével könnyen ki lehet zárni.

2.5.2.1.6 *SQL Injection*

Ezzel a módszerrel a támadó rosszindulatú kódokat injektál az adatbázisba a bejelentkezési űrlapokon keresztül, amely segítségével akár a teljes adatbázisunkhoz is hozzáférhet. Ez ellen manapság alkalmazásoldalon nagyon könnyű védekezni, hiszen az összes modern fejlesztőkörnyezet tartalmaz megoldásokat az ilyen típusú sérülékenységek ellen.

2.5.2.1.7 *Cross-Site Scripting (XSS)*

Ez a támadás akkor történik, amikor a támadó rosszindulatú szkripteket illeszt be egy weboldalba, amit a felhasználók böngészője végrehajt. Az SQL Injection-höz hasonlóan szintén alkalmazás oldalon lehet védekezni ellene.

2.5.2.1.8 *Session Hijacking / Cookie lopás*

A támadó megszerzi a felhasználók munkamenet-azonosítóit vagy a kapcsolódó cookie-kat, így megszemélyesítve egy valós felhasználót. Védekezni ez ellen cookie hasheléssel, valamint a gyanús bejelentkezések figyelésével lehet.

2.5.2.1.9 *CSRF*

A Cross-Site Request Forgery (CSRF) olyan biztonsági sebezhetőség, amely lehetővé teszi a támadók számára, hogy jogosulatlan műveleteket hajtsanak végre egy megbízható felhasználó nevében anélkül, hogy a felhasználónak erről tudomása lenne. A támadás akkor következik be, amikor a felhasználó már bejelentkezett egy weboldalra, és egy másik, rosszindulatú weboldalon keresztül végrehajtott kérések által manipulálják az eredeti weboldal szolgáltatásait.

3 ÖNÁLLÓ MUNKA

TDK dolgozatomban egy olyan webes szolgáltatást tervezek meg, amely alkalmazásokat fejlesztő cégek számára könnyíti meg a felhasználói fiókok kezelését oly módon, hogy azokat egyszerű REST API felületen keresztül gyorsan és biztonságosan kezelhetik.

3.1 Szoftver specifikációja

3.1.1 Üzleti igények meghatározása

Az általam tervezett szoftver egy olyan alkalmazás, amely segítségével a szolgáltatást igénybe vevő (jogosultsággal rendelkező) bérlők (tenant) felhasználóinak adatai nyilvántarthatók, megtekinthetők, a REST API-t meghívó alkalmazások számára elvégzi az autentikációt.

Az elkészült szoftver egy mobiltelefonon is használható felhasználói felülettel kell rendelkezzen, valamint szintén a projekt része a kapcsolódó adatbázis felépítése és kezelése.

Az elkészült szoftver mellett a megfelelő műszaki és felhasználói dokumentáció megléte is követelmény.

3.1.1.1 Funkcionális követelmények

Az elkészült szoftvernek az alábbi fő üzleti célokat kell teljesítenie:

- Rendszerhez hozzáférő ügyfélfiókok kezelése
- Felhasználói fiókok megkülönböztetése jogosultság szempontjából
- Ügyfél által kezelt felhasználók listájának kezelése
- Ügyfél által kezelt felhasználók adatainak nyilvántartása
- Ügyfél számára REST API biztosítása a kapcsolódáshoz
- Szoftver használhatóságának biztosítása asztali és mobileszközről egyaránt
- Az ügyfelek felhasználói e-mail cím és jelszó segítségével tudnak bejelentkezni
- Az ügyfelek számára elérhető a többlépcsős hitelesítés
- Az ügyfelek számára elérhetőek harmadik fél hitelesítési szolgáltatásai (pl Facebook, Google)

3.1.1.2 Nemfunkcionális követelmények

3.1.1.2.1 *Adatvédelem*

- Anonimizált adatok használata fejlesztéshez és teszteléshez
- Adatvédelmi követelmények betartása személyes, bizalmas adatokra
- A GDPR betartásának biztosítása, meghatározott adatmegőrzési folyamatok mentén

3.1.1.2.2 *Dokumentáltság*

- Felhasználói dokumentáció
- Fejlesztői dokumentáció
- Forráskód kommentelése

3.1.1.2.3 *Teljesítmény*

- Skálázhatóság
- Nem pazarló erőforrás kihasználás
- Elvárt teljesítmény biztosítása, párhuzamos felhasználók száma alapján

3.1.1.2.4 *Alkalmazástámogatás*

- Változáskezelés
- Migrálhatóság
- Monitorozhatóság
 - o UX & technikai monitorozás
 - o E2E monitorozás támogatása (kontextuskövetés)
 - o meglévő monitoringeszközökhöz való integrálhatóság biztosítása
- rendszeres karbantartási feladatok dokumentálása

3.1.1.2.5 *Tesztelhetőség*

- Integrációs tesztesetek, tesztforgatókönyvek, valamint tesztkörnyezet megléte, dokumentálása
- Kiegészítő (automatizált) teszt eszközökhöz való integráció lehetősége
- Tesztek visszakövethetősége

3.1.1.2.6 *Biztonság*

- Tűzfalnyitás elkerülése
- OWASP-alapelvek betartása
- Legalacsonyabb jogosultság elve

- Üzembiztos alapműködés
- Standard szoftver használata
- Teljes ellenőrzés

3.1.2 ISO 15408:2 Common Criteria

A Common Criteria for Information Technology Security Evaluation - Part 2: Security Functional Components [10] dokumentum (továbbiakban: a dokumentum):

- a Nemzetközi Szabványügyi Szervezet (ISO) és a Nemzetközi Elektrotechnikai Bizottság (IEC) által kiadott ISO/IEC 15408-2:2008 szabványnak felel meg,
- az információbiztonsági termékek és rendszerek értékelésével foglalkozik.
- Biztonsági funkcionális komponensek meghatározására összpontosít, amelyeket a termékek és rendszerek értékeléséhez használnak annak biztosítása érdekében, hogy megfeleljenek a végfelhasználók biztonsági igényeinek.

A dokumentum célja, hogy egyértelmű és átfogó útmutatást nyújtson a termék- és rendszerfejlesztőknek, valamint az értékelőknek a biztonsági funkciók meghatározásában és értékelésében. Az egyes biztonsági funkcionális komponensek specifikációja magában foglalja a funkció célját, a kapcsolódó biztonsági követelményeket, valamint azokat a kritériumokat, amelyek alapján az adott funkció megvalósítása értékelhető.

Az általam tervezett szoftver esetén a dokumentumból az alábbi elvárások mérvadóak:

- funkcionális követelmények meghatározása
- erős hitelesítési funkciók alkalmazása
- titkosítás és adatvédelem
- auditálás és naplózás
- jogosultságkezelés / hozzáférésvezérlés
- rendszerbiztonság, sebezhetőségek elleni védelem
- felhasználóbarát interface

3.1.3 User storyk meghatározása

A User story-tervezés egy olyan eszköz, amely segítségével érvényesülnek az agilis alapelvek. Lényege, hogy áthelyezi a hangsúlyt a követelmények írásáról azok megvitatására, egyfajta sajátos párbeszédre. A User Story egy rövid leírása az adott funkciónak, mégpedig annak a felhasználói csoportnak a szemszögéből, amelyik használni fogja azt. Ezek a storyk nagyon tömörök, legtöbbször elférnek egy névjegykártya méretű papíron.

A User Storyk általában az alábbi séma szerint épülnek fel:

„Én, mint {felhasználó típusa}, azt szeretném, hogy {valamilyen elérendő cél}, azért, hogy {indíték}.” [11]

A User Storykat a tervezési folyamat során a kapcsolódó funkciókat Epicekbe, azaz nagyobb csoportokba osztják, ezzel garantálva azt, hogy az összetartozó felhasználói felületek, funkciók átláthatóbbak legyenek mind az ügyfél, mind pedig a fejlesztők számára.

Az alább részletezett User Storyk kerültek meghatározásra az üzleti igényben megfogalmazottak alapján.

- adminisztrációs funkciók
- fiókmenedzsment funkciók
- REST API funkciók

3.1.3.1 Adminisztrációs funkciókhoz tartozó user storyk

- Szoftver adminisztrátorként szeretnék új bérlői fiókot létrehozni, hogy a bérlő el tudja kezdeni használni a szoftvert
- Szoftver adminisztrátorként szeretném módosítani a bérlő adatait, hogy a bérlő adataiban keletkező változásokat követni lehessen.
- Szoftver adminisztrátorként szeretném megtekinteni a bérlők listáját.
- Szoftver adminisztrátorként szeretném módosítani a bérlő adatbázisának lokációját, amennyiben a bérlő ezt kéri

3.1.3.2 Fiókmenedzsment funkciókhoz tartozó user storyk

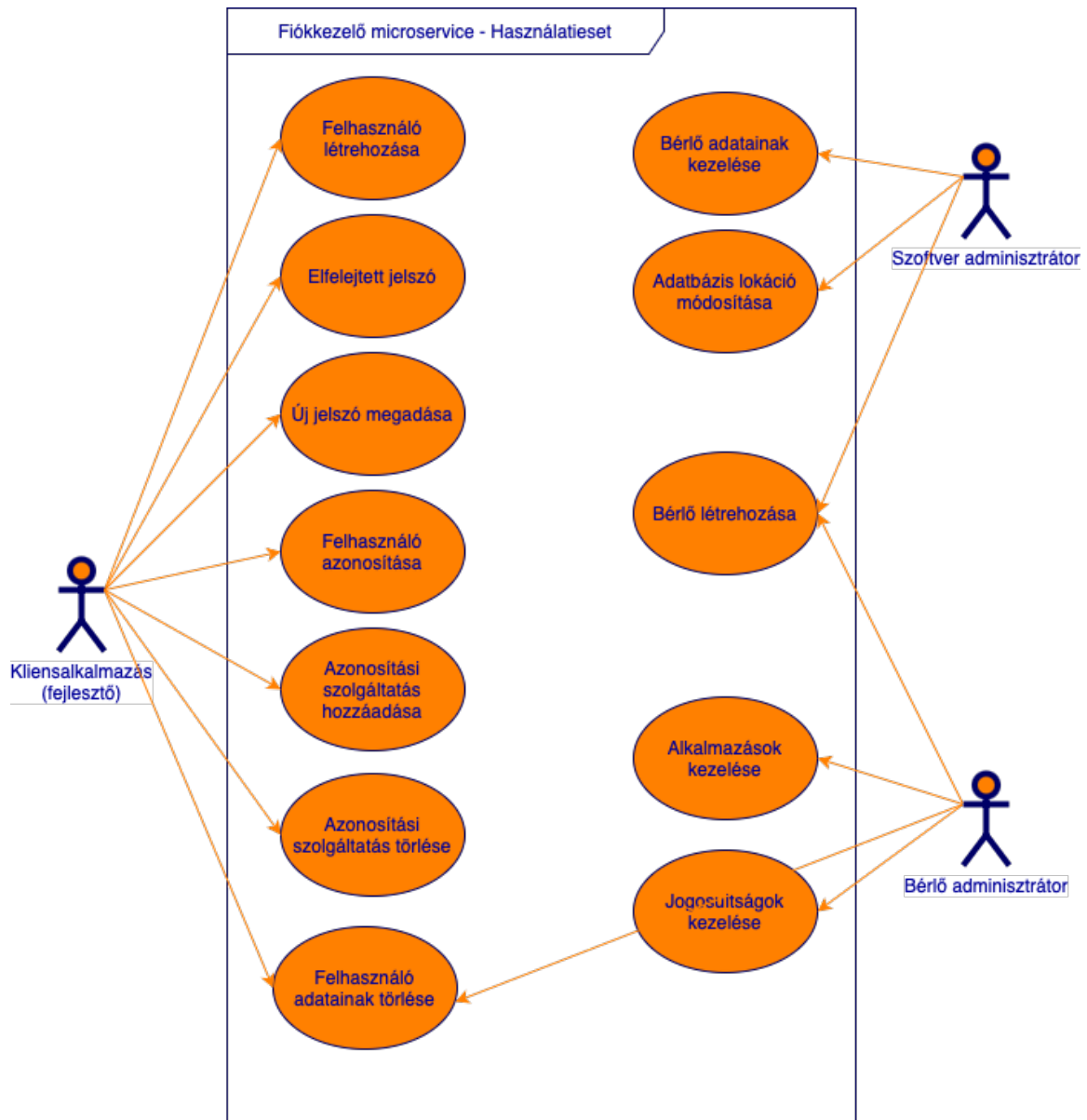
- Bérlő adminisztrátorként szeretnék új bérlői fiókot létrehozni, hogy el tudjam kezdeni használni a szoftvert

- Bérlő adminisztrátoraként szeretnék új alkalmazást lértéhozni, amelyen keresztül felhasználókat tudok regisztrálni a szoftverbe
- Bérlő adminisztrátoraként szeretném kezelni a jogosultságokat, amelyeket később hozzá tudok rendelni a felhasználókhoz
- Bérlő adminisztrátoraként szeretném módosítani a felhasználókhoz tartozó jogosultságokat, ezzel meghatározva, hogy adott felhasználó milyen jogosultságokkal rendelkezik a rendszerben
- Bérlő adminisztrátoraként szeretném törölni a felhasználót, és a felhasználóhoz kapcsolt adatokat a rendszerből

3.1.3.3 REST API funkciókhoz tartozó user storyk

- Fejlesztőként szeretnék egy funkciót, amely segítségével új felhasználót tudok regisztrálni
- Fejlesztőként szeretnék egy funkciót, amely segítségével jelszóvisszaállítási folyamatot tudok igényelni adott felhasználóhoz
- Fejlesztőként szeretnék egy funkciót, amely segítségével meg tudok győződni arról, hogy a felhasználó által megadott azonosítási adatok megfelelőek-e
- Fejlesztőként szeretnék egy funkciót, amely segítségével törölni tudom a felhasználót és a felhasználóhoz kapcsolt adatokat a rendszerből
- Fejlesztőként szeretnék egy funkciót, amely segítségével új jelszót állíthatok be a felhasználóhoz
- Fejlesztőként szeretnék egy funkciót, amely segítségével új azonosítási szolgáltatást tudok felvenni a felhasználóhoz
- Fejlesztőként szeretnék egy funkciót, amely segítségével törölni tudok meglévő azonosítási szolgáltatást a felhasználó fiókjából

3.1.4 Használatieset-diagram



3. ábra - Használatieset-diagram (Forrás: saját ábra)

3.2 A szoftver tervezése

3.2.1 Adatbázisarchitektúra

Szoftverem tervezésekor elsődleges szempontként tekintek az adatbiztonságra, illetve a megfelelő adatizolációra, ezért a tervezés során úgynevezett multi-tenant (több-bérlős) architektúra kialakítása mellett döntöttem. A multi-tenant architektúra lehetővé teszi, hogy a

szoftver egyetlen példánya egyszerre több bérlő (tenant) számára is szolgáltatást nyújtson, miközben az adatokat bérlőnként elkülönítve kezeli. A tervezett megoldásban egy központi adatbázist használok a tenantok adatainak kezelésére, és minden egyes tenant számára külön adatbázist hozok létre azok saját működési adatainak tárolására. Ez az elrendezés több előnnyel is jár:

- a külön adatbázisok használata erősebb adatelkülönítést tesz lehetővé, ami növeli az adatbiztonságot. Az egyes tenantok adatai fizikailag is elkülönülnek, így minimálisra csökkenthető a jogosulatlan adathozzáférés kockázata.
- mivel a tenantok adatbázisai külön vannak kezelve, gyakran különböző szervereken vagy akár adatközpontokban elhelyezve, ez pedig lehetővé teszi az egyedi skálázási igények könnyebb kezelését
- az adatbázisok elkülönítése javíthatja a rendszer teljesítményét, különösen nagyszámú tenant esetén, mivel a különböző tenantok adatbázis-műveletei nem befolyásolják egymást és nem közös erőforrásból dolgoznak.

A fenti megoldás csak adatbázis szintjén különíti el az egyes bérlőket, így az alkalmazást futtató kód minden bérlő esetében ugyanaz, csak a kapcsolódó adatbázisok különböznek. Erre a tervezési módra az AWS "Targeted Silo Isolation"-ként hivatkozik. [12]

3.2.2 Adatbázis modellezés

3.2.2.1 Adatmodellek

3.2.2.1.1 Központi adatbázis

Adminok (admins tábla)

- id
- name
- email
- email_verified_at
- password
- remember_token
- created_at
- updated_at

Adatbázis migrációk követése (migrations tábla)

- id
- migration
- batch

Jelszóvisszaállítási tokenek (password_reset_tokens tábla)

- email
- token
- created_at

Bérlők (tenants tábla)

- id
- company_name
- domain
- database
- address
- phone
- email
- city
- zip
- created_at
- updated_at

3.2.2.1.2 Bérlői adatbázis

Adminok (admins tábla)

- id
- name
- email
- email_verified_at
- password
- remember_token
- created_at

- updated_at

Adatbázis migrációk követése (migrations tábla)

- id
- migration
- batch

Jelszóvisszaállítási tokenek (password_reset_tokens tábla)

- email
- token
- created_at

Felhasználók (users tábla)

- id
- name
- email
- email_verified_at
- password
- remember_token
- created_at
- updated_at

Jogosultságok (permissions tábla)

- id
- name
- created_at
- updated_at

Alkalmazások (applications tábla)

- id
- name
- created_at
- updated_at

Alkalmazás – Jogosultság reláció (application_permission tábla)

- application_id
- permission_id

Jogosultság – Felhasználó reláció (permission_user tábla)

- permission_id
- user_id

Azonosítási szolgáltatások (providers tábla)

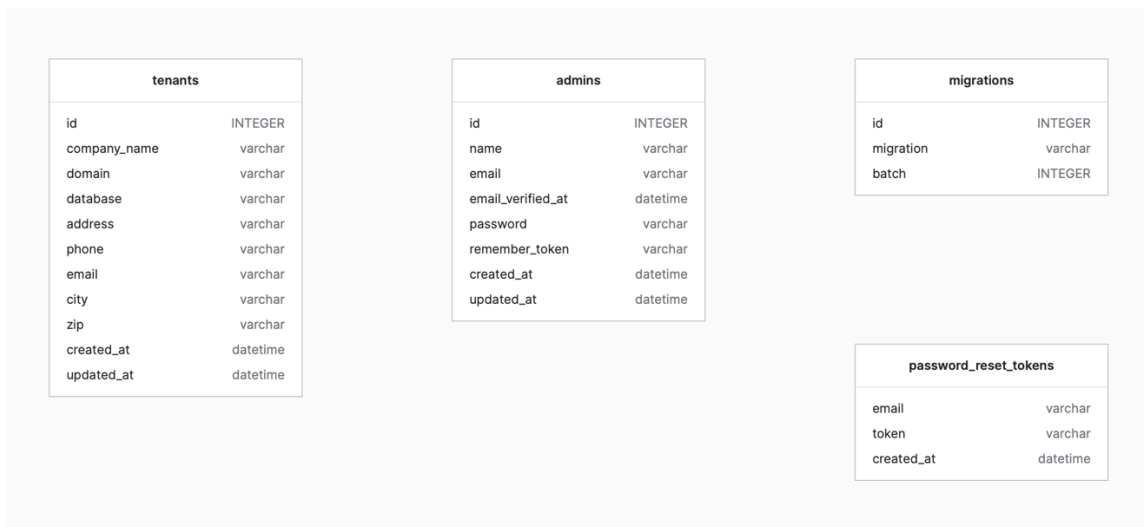
- id
- slug
- name
- service_class
- created_at
- updated_at

Azonosítási szolgáltatás – Felhasználó reláció (provider_user tábla)

- provider_id
- user_id

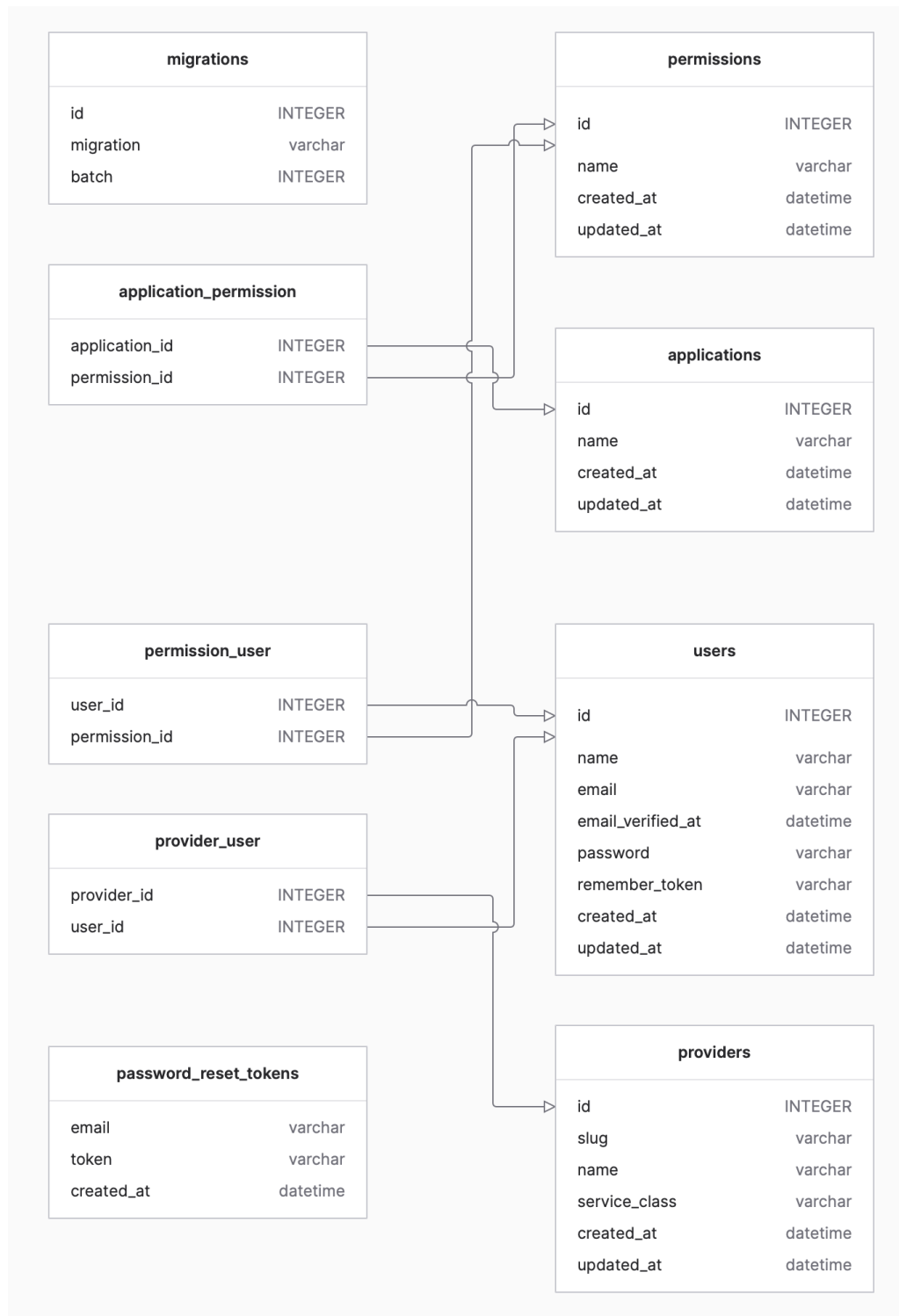
3.2.2.2 Entity Relation Diagram

A központi adatbázishoz tartozó diagramon is látható, hogy a táblák között különösebb kapcsolat nincs.



4. ábra - Központi adatbázis ER-diagramja (Forrás: saját ábra)

Az egyes bérlőkhöz kapcsolódó adatbázis ER-diagramja az alábbi ábrán látható:



5. ábra - Bérlők adatbázisának ER-diagramja (Forrás: saját ábra)

3.2.3 Szoftverfejlesztési módszertan és technológia kiválasztása

3.2.3.1 Fejlesztéshez kiválasztott technológiák

A tervezés során kiválasztott technológiai stack a szoftverfejlesztésben általánosan csak TALL-nek hívott (Tailwind, Alpine, Livewire, Laravel) [13] szolgáltatáscsomag.

A TALL stack 4 fő komponensből áll:

- Tailwind CSS – frontend dizájn
- AlpineJS – reaktív frontend Javascript segítségével
- Laravel – Fejlesztői keretrendszer PHP nyelven a backend funkciók megvalósításához
- Livewire – Összekötő réteg a backend és frontend közötti reaktivitás megvalósításához

A TALL stack egy kellően robusztus, full-stack fejlesztéshez használható alapot ad a fejlesztéshez, és rengeteg olyan komponenst tartalmaz, amely felgyorsítja a fejlesztési folyamatot.

A forráskód tárolásához, valamint a kapcsolódó CI/CD folyamatokhoz a GitHub [14] szolgáltatását választottam. Választásom a könnyű kezelhetőség, és nagy felhasználóbázis miatt esett erre a platformra.

3.2.3.2 Fejlesztési módszertan

A fejlesztési módszertan kiválasztása során fontos szempont volt egyrészt a hatékony és minőségi kód létrehozása, másrészt az alkalmazás kiadása éles üzemre a lehető legkorábban. Egy olyan módszertant választottam ki, mely a későbbiekben érkező új üzleti- és felhasználói igényeket könnyen, gyorsan és skálázhatóan képes kezelni.

A fejlesztéshez a Scrum módszertant választottam, míg a terméktervezés esetén a lean alapelvei mellett tettem le a voksomat. A célom az Eric Ries által használt MVP (Minimum Viable Product - Minimálisan működőképes termék) kifejlesztése volt a cél. [15]

A Scrum lényege – a végletekig lecsupaszítva – az egyszerűsítés. Ez a módszertan minden bonyolult folyamatot addig darabol, míg az egy egyszerű, önálló blokkként meg nem jelenik. Mindezek mellett a Scrum kiváltképp rugalmas is: minden résztvevő elfogadja, hogy történnek változások, de ezekre egyikük sem akadályként tekint, hanem ellenőrizendő és ki nem kerülhető feladatként. A folyamat során kis feladatokká darabolják a nagy egészet, ezt hívják product backlognak. A backlog nemcsak a célokat, de a prioritásokat is tartalmazza. Az ügyfél

folyamatos visszajelzései ezzel párhuzamosan érkeznek, és hajtódnak végre. Az ügyfél szinte a csapat része, folyamatos visszajelzéseivel lesz a termék magas minőségű, az elkészüléséhez vezető folyamat pedig költséghatékony. A Scrum módszertan a nagyon gyakori visszaellenőrzésekkel szinte teljesen megszünteti a kockázatot. Személyes kedvencem a tervezés során használt „Scrum-póker” névre keresztelt eljárás, amivel a fejlesztők közös megegyezéssel, vitákkal döntenek el hogy az adott feladatra mennyi időt kell majd szánni.

3.2.4 A „The twelve-factor app”-módszertan

Az Adam Wiggins által megalkotott, “The twelve-factor app”-módszertan lényege, hogy az alábbi 12 pontban foglalja össze egy webalkalmazás (vagy egyéb tetszőleges SaaS-alkalmazás) fejlesztésének alapjait:

1. Központi kódbázis, verziókezelés
2. Függőségek kezelése
3. Konfigurálhatóság, környezetek szétválasztása
4. Szolgáltatások, mint csatolt erőforrások
5. Build - Release - Run stage-ek
6. Állapotmentes folyamatok
7. Portkötés
8. Skálázhatóság
9. Folyamatok eldobhatósága
10. Fejlesztői és éles környezet közötti paritás
11. Monitorozhatóság
12. Adminfolyamatok

A fenti irányelvek egyrészt az üzleti igényekben megfogalmazott nem funkcionális követelmények alkalmazásoldali részeit is lefedik, másrészt megvalósítják az alábbi továbbfejlesztés szempontjából fontos elveket:

- átlátható forráskód
- új fejlesztő gyors és hatékony bevonása
- könnyű konfiguráció
- fejlesztés az éles környezethez nagyon közel álló (a legtöbb helyen megegyező) architektúra és folyamatok [16]

3.3 Külső integráció tervezése

Szoftvert felhasználó ügyfelek felé mindenképpen szükséges integrációs lehetőségek kialakítása, hiszen ez a szolgáltatás önmagában kevésbé értelmezhető.

3.3.1 REST API tervezése

A REST API egy szabványosított interfész a különböző informatikai rendszerek közötti kommunikációhoz. A REST API-k protokollja a HTTP, amely lehetővé teszi az erőforrások (adatok vagy funkciók) egyszerű elérését URL-ek segítségével. Az erőforrásokat általában JSON vagy XML formátumban adják vissza, amelyek könnyen feldolgozhatók különböző, az adott felhasználási célnak megfelelő programozási nyelvekben.

A REST alapelvei közé tartozik az állapotmentesség, amely azt jelenti, hogy minden kérés független, és tartalmaznia kell minden szükséges információt a kérés feldolgozásához. A REST API-kat úgy tervezték, hogy skálázhatók, egyszerűek és könnyen használhatók legyenek.

A REST API műveletei az alábbi HTTP-metódusokra épülnek:

- GET - meglévő erőforrás adatainak lekérdezése
- POST - új erőforrás létrehozása
- PUT - meglévő erőforrás frissítése
- PATCH - meglévő erőforrás adatainak részleges frissítése
- DELETE - meglévő erőforrás törlése

A REST API-k előnyei közé tartozik a technológiafüggetlenség, a könnyű integrálhatóság, és az, hogy a fejlesztők számára az előző két tulajdonságából adódóan egyszerűen érthető és használható. A fenti fő tulajdonságok miatt esett a választásom a szoftver tervezésekor a REST API-ra, ezzel biztosítható, hogy az első kiadástól kezdve, technológiafüggetlenül lehessen használni az általam tervezett szoftver funkcióit.

3.3.1.1 Fogalmak

- tenant_id - a bérlő egyedi azonosítója
- client_id - a bérlő által definiált kliensalkalmazás azonosítója
- client_secret - a kliensalkalmazáshoz kapcsolódó titkos kulcs
- provider - az azonosítási módszer neve (pl: password; google; facebook)
- payload - a bérlő alkalmazása által küldött adat, mely API-végpontként változhat
- user_id - a bérlő alkalmazásában tárolt felhasználó azonosítója

3.3.1.2 Ügyfélazonosítás

Minden API-végpont hívásához alapvető azonosításra van szükség annak érdekében, hogy a szoftver azonosítani tudja a bérlőt, és a kapcsolódó alkalmazását. Ezeket az adatokat kötelezően fejléc (header) információként kell elküldeni, az alábbi nevekkel:

- X-TENANT-ID
- X-CLIENT-ID
- X-CLIENT-SECRET

3.3.1.3 API-végpontok listája

3.3.1.3.1 *Felhasználó regisztrálása*

- Végpont URL: /user/register
- HTTP-metódus: POST
- Kötelező adattartalom:
 - o email: felhasználó e-mail címe
 - o provider: azonosítási szolgáltatás neve
 - o payload: providerhez tartozó extra adatok, melyek szolgáltatásonként eltérnek

Válasz: Siker esetén a végpont visszaadja az újonnan létrehozott felhasználó azonosítóját.

3.3.1.3.2 *Jelszóvisszaállítási folyamat indítása felhasználóhoz*

- Végpont URL: /user/{user_id}/forgot-password
- HTTP-metódus: POST
- Kötelező adattartalom:
 - o email: felhasználó e-mail címe
 - o new_password_form: URL, amely az új jelszó beállításához szükséges űrlapot tartalmazza

Válasz: Siker esetén a végpont visszaad egy tokent, amellyel kliensoldalon azonosítható, hogy valóban a felhasználó próbálja meg kitölteni az új jelszó beállításához szükséges űrlapot

3.3.1.3.3 *Jelszó frissítése*

- Végpont URL: /user/{user_id}/password
- HTTP-metódus: POST
- Kötelező adattartalom:
 - o old_password_hash - SHA512 algoritmussal hashelt régi jelszó
 - o new_password - SHA512 algoritmussal hashelt új jelszó

Válasz: Siker esetén a végpont üres adattartalommal tér vissza.

3.3.1.3.4 *Új azonosítási szolgáltatás felvétele felhasználóhoz*

- Végpont URL: /user/{user_id}/providers
- HTTP-metódus: POST
- Kötelező adattartalom:
 - o provider: azonosítási szolgáltatás neve
 - o payload: providerhez tartozó extra adatok, melyek szolgáltatásonként eltérnek

Válasz: Siker esetén a végpont üres adattartalommal tér vissza.

3.3.1.3.5 *Azonosítási szolgáltatás törlése*

- Végpont URL: /user/{user_id}/providers/{provider}
- HTTP-metódus: DELETE

Válasz: Siker esetén a végpont üres adattartalommal tér vissza.

3.3.1.3.6 *Felhasználóhoz tartozó azonosítási szolgáltatások listája*

- Végpont URL: /user/{user_id}/providers
- HTTP-metódus: GET

Válasz: Siker esetén a végpont a kért listával tér vissza.

3.3.1.3.7 Felhasználó törlése

- Végpont URL: /user/{user_id}
- HTTP-metódus: DELETE

Válasz: Siker esetén a végpont üres adattartalommal tér vissza.

3.4 Hosting

Az alkalmazás hostolására az AWS Cloud felhőszolgáltatásait választottam serverless alapokon. A javasolt cloudsolgáltatások konfigurációját a fejlesztés során, a megfelelő szakértelemmel, az üzleti igényekre szabva szükséges elvégezni.

Az alábbiakban részletezem a szoftver működtetéséhez kiválasztott szolgáltatásokat.

3.4.1.1 Lambda

Alapvetően egyszerű függvények futtatásához hívta életre az Amazon ezt a szolgáltatást, mára azonban egy komplett alkalmazásfuttatási környezetté nőtte ki magát. A Lambda legnagyobb előnye, hogy bármekkora forgalmat képes kiszolgálni. A bejövő forgalom irányítását úgy oldja meg, hogy a háttérben annyi virtuális példányt futtat az adott alkalmazásból, amennyire éppen szükség van. Amennyiben egy alkalmazásra nem érkezik forgalom, a Lambda leállítja a háttérben üzemelő példányokat, így csak azért a forgalomért és erőforrásokért kell fizetni, amit valóban felhasznál az alkalmazás.

Egyetlen hátránya az úgynevezett “hidegindítás” vagy eredeti nevén a cold start. Ez akkor fordul elő, ha egy kérést egy még el nem indított példányra irányítja. Ebben az esetben válaszütem további 1-2 másodperccel megnő. Azonban ezeket a példányokat az AWS Lambda szolgáltatása további 15-20 percig nem állítja le, így az időközben érkező, esetleges újabb kérések már villámgyorsan kerülnek kiszolgálásra. [17]

3.4.1.2 RDS Aurora Serverless

Az RDS az Amazon menedzselt adatbázis-szolgáltatása. Ennek egyik része az Aurora Serverless szolgáltatás, amely segítségével könnyedén megvalósítható gyorsan skálázódó adatbázis-szolgáltatás, ami a forgalom függvényében allokál erőforrásokat felhasználásra.

A Lambda szolgáltatáshoz hasonlóan gyakorlatilag korlátlanul skálázható és szintén képes a nem használt szolgáltatás szüneteltetésére. Ez azonban éles üzem esetén nem javasolt. A

szolgáltatás automatikus altatásának előnye inkább a tesztkörnyezetek használata esetén hasznos. [18]

3.4.1.3 Simple Storage Service (S3)

Az Amazon egyik alapszolgáltatása, mely a kezdetektől fogva része az AWS infrastruktúrájának. Fájlok tárolására szolgál és előnye, hogy közvetlen összeköttetésben van az opcionálisan használható CloudFront CDN szolgáltatással, ezzel megkönnyítve a fájlok tárolását a felhőben, valamint eljuttatását a felhasználókhoz.[19]

3.4.1.4 API Gateway

Az API Gateway szolgáltatás segítségével köthetjük össze a beérkező nyilvános webes forgalmat a kapcsolódó Lambda szolgáltatással. Ezzel megoldható hogy a megfelelő domainhívás esetén a kapcsolódó mikroszolgáltatás lépjen működésbe.[20]

3.4.1.5 CloudFront

Az AWS CloudFront szolgáltatás az Amazon saját CDN-szolgáltatása, mely segítségével az alkalmazás által használt, felhasználói felület számára publikált statikus fájlokat szolgáljuk ki. Jelen alkalmazás vonatkozásában ezek például a letölthető szoftverek, valamint a felhasználói felület által használt grafikai elemek, CSS- és JavaScript-fájlok.

Előnye, hogy a fájlok gyorsítótárazását az Amazon által menedzselt CDN szolgáltatás végzi, amely garantálja, hogy minden esetben a felhasználó számára elérhető legközelebbi lokációból töltődjenek be a fájlok, azért, hogy az alkalmazás gyorsan és megbízhatóan működjön. [21]

3.4.1.6 WAF

Az AWS WAF szolgáltatás egy menedzselt, szoftveres tűzfalszolgáltatás kifejezetten webalkalmazások számára. A szolgáltatáson keresztül számos IT biztonsági beállítás megvalósítható, ezáltal védelmet biztosíthatunk a rosszindulatú hozzáféréssel szemben.

A WAF segítségével rate limitereket, forgalomszűrést és monitorozást valósíthatunk meg.[22]

3.4.1.7 Simple Email Service

Ritka azoknak a szoftvereknek a száma, amelyek valamilyen formában nem küldenek e-maileket. Legyen szó értesítő- vagy egyéb marketingjellegű levelekről.

Az e-mailek programozott küldését manapság már minden alkalmazás tudja, azonban a kapcsolódó szerveroldali és domainszintű konfigurálások rengeteg időt és szakértelmet

igényelnek. E-mail küldése esetén figyelni kell a küldő IP-címére, a domain rekordjainak megfelelő beállítására, a kiküldendő e-mail formai és fejlécszintű megjelenési követelményeire is.

Az AWS SES e-mailek küldésére használható menedzselt szolgáltatás. Előnye, hogy nem kell saját e-mailszerver konfigurálásával és üzemeltetésével foglalkozni, a fentebb részletezett szerveroldali beállításokat az Amazon biztosítja, a domain szintű beállításokhoz pedig egy könnyen használható adminisztrációs felületet biztosít.[23]

3.4.1.8 Simple Queue Service

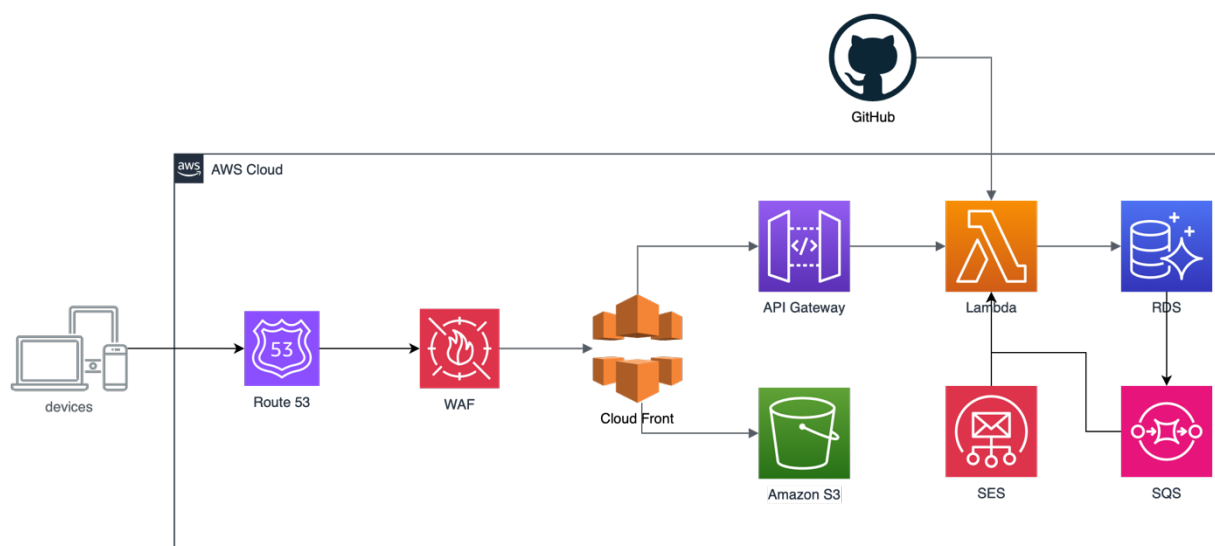
Az AWS SQS szolgáltatása egy menedzselt üzenetszolgáltatás, mely segítségével a különböző mikroszolgáltatások kommunikálnak egymással. Azokat az időigényes háttérfeladatokat, amelyek a felhasználói felület szempontjából nem fontosak vagy kritikusak, az SQS szolgáltatáson keresztül a háttérben dolgozzuk fel.[24]

3.4.1.9 Route 53

Az AWS Route 53 egy rugalmas és skálázható DNS (Domain Name System) szolgáltatás, amely nagy rendelkezésreállással rendelkezik, valamint beépítetten támogatja az AWS-en elérhető összes szolgáltatást. [25]

3.4.2 Cloud architektúra diagram

A fentebb részletezett AWS cloud alapú hosting az alábbi szolgáltatásdiagram szerint írható le egy tipikus felhasználói kérés szemszögéből:



6. ábra - Cloud architektúra diagram (Forrás: saját ábra)

3.5 IT-biztonság

Az informatikai biztonságot sokszor elhanyagolják a tervezés és fejlesztés során, mondván, hogy ez a terület olyan nemfunkcionális követelmények sokaságát tartalmazza, amelyre nincs dedikált költségvetés, és minden aspektusának megvalósítása sokszor a projektre áldozott összeg jelentős részét felemésztene.

Szerencsére manapság, mint ahogyan sokminden mást, az IT-biztonság kapcsolódó funkcióit is igénybe lehet venni úgynevezett “aas” (as-a-service) modellben, szolgáltatásként. Így sok esetben a szolgáltató rengeteg terhet levesz az üzemeltetők és a fejlesztők válláról.

Másrészről alkalmazásfejlesztésnek ma már ritkán látnak neki úgy a fejlesztők, hogy teljesen üres “vászonra” dolgoznának. Minden programnyelvnek, platformnak megvan a maga keretrendszere, amely minden esetben rendelkezik az alapvető, fejlesztésoldali rendszerbiztonsághoz szükséges építőelemekkel.

TDK dolgozatom témáját szolgáltató, általam tervezett szoftver megalkotása során 2 fő csoportot határoztam meg biztonsági szempontból az alkalmazás fejlesztésétől a minőségbiztosításig, melyek segítségével elkerülhető a napjainkban előforduló incidensek jelentős többsége:

- Alkalmazásoldali biztonság
- WAF-védelem

Az IT-biztonsági tervezés során kiemelt figyelmet szenteltem az OWASP (Open Web Application Security Project) 10 legfontosabb, webalkalmazásokat érintő sérülékenységi listájának. [26] A tervezésem alapját ez határozta meg.

3.5.1 Alkalmazásoldali biztonság

A 2.5.2-es fejezetben felsorolt főbb támadási faktorok ellen terveim szerint az alábbi funkciókkal lehet védekezni:

3.5.1.1 Rate limit – kérések korlátozása

Alapvető célja, hogy korlátozza a felhasználók vagy a kapcsolódó rendszerek által egy adott időszak alatt küldött kéréseket. Ezzel egyrészt megakadályozza a szolgáltatás túlterhelését és védi a kapcsolódó erőforrásokat is. A korlátozások beállítása és finomhangolása hosszas folyamat, és gyakran előfordul, hogy valós felhasználó is kizárásra kerül, ezért nagyon fontos a

folyamatos monitorozás és finomhangolás. Egy jól beállított rendszer az alábbi támadásoknak képes ellenállni: DDoS, Brute Force, Dictionary Attack

3.5.1.2 Gyanús bejelentkezések szűrése

A rendszer tanulmányozza a felhasználók bejelentkezési mintáit, beleértve az időpontokat, a lokációt, az eszközöket és a böngészőket, amelyeket rendszerint használnak. Ez létrehoz egy "normál" viselkedési profilt minden egyes felhasználó számára. Amennyiben a rendszer a normáltól eltérő viselkedési mintát talál, újra azonosítja a felhasználót, ezzel is megelőzve számos gyakori támadási formát, mint például a Phishing, Credential Stuffing vagy Session Hijacking.

3.5.1.3 Potenciális lehallgatás megakadályozása

Az SSL Pinning, egy olyan technika, amely kifejezetten arra szolgál, hogy növelje az adatátvitel biztonságát a kliens alkalmazások és a szerverek között. Ezáltal a kliensalkalmazás előre rögzíti a szerver tanúsítványának egy vagy több jellemzőjét, mint például a közvetlenül kiadott tanúsítvány publikus kulcsát, vagy a tanúsítványláncban lévő megbízható kiadó CA (Certificate Authority) publikus kulcsát. Ezután, amikor a kliens csatlakozik a szerverhez, ellenőrzi, hogy a kapott tanúsítvány megfelel-e a rögzített publikus kulcsnak vagy tanúsítványsornak. Ha nem, a kapcsolatot megszakítja, így védekezve a potenciális Man-in-the-Middle támadások ellen.[27]

3.5.2 Cloudalapú védelem (WAF)

A WAF a Web Application Firewall rövidítése. A funkcióját tekintve képes monitorozni, a HTTP és HTTPS forgalmat. Így mielőtt még elérné a kiszolgálót a kérés, szűrhetővé válik. A szűrés tárgya lehet egyszerűen GEO, IP reputation vagy akár összetettebb, még nem javított sebezhetőség kivédése.

Számos szolgáltató kínál WAF-védelmet, a TDK dolgozatomban írása során az Amazon AWS WAF [22] szolgáltatás kínálatával terveztem.

Az elsődleges működési logikáját tekintve 4 típusú lehet:

- kizárólista
- megengedőlista
- számoló
- CAPTCHA

Mind arra épít, hogy valamiféle előre definiált logika szerint kiválasztott kérésekre milyen visszatérési értéket vár.

A kizárólista blokkol mindent ami a listán van. Tehát alapból minden kérés kiszolgálásra kerül, ami nem szerepel a listán.

A megengedő lista az előző fordítottja, csak azokat a kéréseket szolgálja ki, amelyek a listán szerepelnek.

A számoló logika nem blokkol, csak számolja hogy adott kritériumnak mennyi kérés felelt meg. Nagyon hasznos lehet erőforrás-tervezés során.

A CAPTCHA szintén nem blokkol, de kötelezi a felhasználót egy feladat végrehajtására, mielőtt még továbbengedné a kérést, ezzel bizonyítékot szerezve arra, hogy a felhasználó valójában egy ember és nem egy szoftver.

A listán szereplő kritériumok sokfélék lehetnek. Vannak előre megírt, folyton frissülő csomagok. Ezek verziói dokumentáltak, ingyenesen, előzetes konfiguráció nélkül használhatók, de saját magunk is írhatunk ilyen kritériumokat. Így elérhetjük, hogy például csak bizonyos IP-címekről lehessen megnyitni az alkalmazást, vagy például csak adott országból lehessen hozzáférni egy szolgáltatáshoz.

Az előre megírt kritériumcsoportokkal ki lehet védeni olyan támadást is, amire még nincs megoldás a kiszolgálónk oldalán.

Főbb elérhető csoportok az AWS kínálatában:

- PHP application managed rule group
- Windows operating system managed rule group
- Linux operating system managed rule group
- SQL database managed rule group
- WordPress application managed rule group
- IP reputation rule groups
- Anonymous IP list managed rule group

3.5.3 Kódminőség

A fejlesztés és élesítés során a modern szoftverfejlesztési technológiák számos olyan ellenőrzési, auditálási lehetőséget biztosítanak, amellyel már a fejlesztés korai fázisában fény

derülhet olyan biztonságilag kritikus hiányosságokra, melyek később, az üzemeltetés során rengeteg problémát, sérülékenységet okozhatnak.

Az alábbi technológiák használata azért fontos, mert ezek szinte teljesen automatizálhatóak, az esetek többségében egyetlen alkalommal kell őket konfigurálni, és alkalmazásukkal folyamatosan garantálható a kódminőség.

Az általam megtervezett szoftver fejlesztése és szállítása során a továbbiakban részletezett technológiák alkalmazásával tervezem a megfelelő kódminőséget biztosítani.

3.5.4 CI/CD-rendszer használata

A CI/CD (Continuous Integration/Continuous Delivery) rendszerek nagyban növelik a fejlesztőcsapatok hatékonyságát, és csökkentik a szoftverben lévő hibák számát. Lényegük, hogy használatukkal teljesen automatizáltá válnak a szoftver buildelési és automata tesztelési folyamatai.

A korábban kiválasztott GitHub platform beépített CI/CD szolgáltatással rendelkezik, melynek neve GitHub Actions. Az automatizált folyamatok beállítása egyszerű, a forráskód mellé .github/workflows mappában kell definiálnunk azokat a lépéseket, melyeket minden kódmódosítás beküldésekor szeretnénk futtatni.

A tervezett szoftver esetében a lépések a következők:

- Statikus kódanalízis
- Függőségek telepítése (build)
- Frontend assetek összeállítása (bundle)
- Unit tesztek futtatása
- Integrációs tesztek futtatása
- End-to-End tesztek futtatása
- Szoftver kiadása (release / delivery)

Amennyiben a fenti lépések közül valamelyik hibára fut, a fejlesztőcsapat azonnal értesül róla, és azonnal megkezdődhet a hiba javítása, ráadásul a fenti folyamat garantálja azt is, hogy egy fejlesztői hiba következtében rossz verzió ne kerülhessen kiadásra.

A GitHub actions rugalmas infrastruktúrája lehetőséget ad az általánosan elfogadott Linux alapú futatókörnyezetek mellett Windows vagy MacOS alapú futtatást is, amely hosszú távon segítheti a tervezett szoftver multi-platformmá válását is.[28]

3.5.5 Függőség ellenőrzés (Dependency-checking)

A szoftver fejlesztéséhez kiválasztott Laravel framework backend (PHP) és frontend (Javascript) oldalon is számos függőséget hoz magával. Ráadásul a szoftver fejlesztése során ezek a külső függőségek folyamatosan bővülnek egy-egy újabb funkció bevezetésével. Ha egy fejlesztőcsapat nem tartja karban ezeket a függőségeket, akkor előfordulhat az az eset, hogy habár a megírt forráskód nem tartalmaz biztonsági réseket, azonban az alkalmazás mégis sebezhetővé válik egy elavult, nem frissített függőség következtében. Éppen ezért IT biztonság szempontjából kiemelt szerepe van a folyamatos ellenőrzésnek, és frissítésnek.

PHP nyelven a legnépszerűbb csomagkezelő a Composer, Javascript esetén pedig az NPM (Node Package Manager). A GitHub Actions egy kiegészítő ún. Dependabot szolgáltatása képes a függőségeket leíró fájlok elemezésére, és javaslatot tenni a kapcsolódó csomagok frissítésére úgy, hogy azok továbbra is kompatibilisek maradjanak az eredeti alkalmazással. A szolgáltatás beállítása – hasonlóan a CI/CD konfigurációjához – a `.github/workflows` mappában történik egy `dependabot.yml` konfigurációs fájl feltöltésével. [29]

A függőségek folyamatos karbantartása nagyban növeli az alkalmazás biztonságát.

3.5.6 Statikus kódanalízálás

A statikus kódanalízis egy olyan eljárás, mely során a forráskódot átvizsgálja egy vagy több szolgáltatás anélkül, hogy azt lefuttatná. A statikus analízálásnak több szintje is van, és sokszor a fejlesztőkörnyezet már beépítetten tartalmaz ilyen eszközöket, ezzel is hatékonyabbá téve a fejlesztési folyamatot. Ezek az ellenőrzések a korábban részletezett CI/CD folyamat első lépéséhez tartoznak.

Az alább felsorolt minőségi problémák kiküszöbölésére terveztem statikus kódanalízist. A problémák mellett zárójelben tüntettem fel az általam kiválasztott szolgáltatás nevét is:

- forráskód egységes formázása (Prettier [30], Laravel Pint [31], ESLint [32]),
- redundáns kódrészletek kiszűrése (SonarLint [33]),
- felesleges komplexitás felismerése (SonarQube [34]),
- biztonsági sebezhetőségek azonosítása (SonarQube).

3.5.7 Automatizált tesztek futtatása

A fentebb felsorolt eszközök és folyamatok kellő stabilitást adnak a szoftver hatékony fejlesztéséhez és kiadásához, azonban jól megírt automatizált tesztek nélkül a szoftver teljesítménye és felhasználói élménye rossz lehet. Ráadásul megfelelő tesztek hiányában újabb funkciók fejlesztése is nehézkessé válik, valamint a fejlesztők cseréje sem egyszerű folyamat.

A tesztek megtervezésekor 3 fő szintet választottam: unit tesztek, integrációs tesztek és end-to-end (E2E) tesztek.

3.5.7.1 Unit tesztek

A unit tesztek a legkisebb kódegységekre, például függvényekre vagy metódusokra összpontosítanak. Izoláltan tesztelhetők a modellek, kontrollerek és egyéb osztályok. Ezek a tesztek gyorsan lefutnak és segítenek biztosítani, hogy az egyes komponensek a várt módon működjenek izolált környezetben. A unit tesztek írása során fontos cél hogy tiszta, karbantartható és könnyen érthető tesztek jöjjenek létre, amelyek pontosan tükrözik a tesztelt funkciókat.

3.5.7.2 Integrációs tesztek

Az integrációs tesztek arra szolgálnak, hogy ellenőrizzék, hogyan működnek együtt az alkalmazás különböző részei. Ezek a tesztek vizsgálják a komponensek közötti interakciókat, például hogy a kontrollerek helyesen kommunikálnak-e az adatbázissal, vagy hogy a backend és frontend komponensek integrációja megfelelő-e. Laravel esetében az integrációs tesztek írásához számos tesztelési funkciót biztosít a keretrendszer, aminek segítségével realisztikus környezetben tesztelhető az alkalmazás komponenseinek együttműködése.

3.5.7.3 E2E tesztek

Az E2E tesztek az alkalmazás teljes működését vizsgálják a felhasználó szemszögéből, egy böngésző vagy egy böngésző-szimulációs eszköz használatával. Ezek a tesztek segítenek azonosítani a felhasználói interakciók és a felhasználói élmény problémáit, amelyek az alacsonyabb szintű tesztek során nem feltétlenül válnak nyilvánvalóvá. Az E2E teszteléshez használhatóak a Cypress vagy a Laravel Dusk eszközök, amelyek szimulálják a felhasználói interakciókat, és ellenőrzik az alkalmazás működését a valós felhasználói forgatókönyvek alapján.

3.6 Összefoglalás

TDK dolgozatomban egy olyan webes szolgáltatást terveztem meg, amely alkalmazásokat fejlesztő cégek számára könnyíti meg a felhasználói fiókok kezelését oly módon, hogy azokat egyszerű API-felületen keresztül gyorsan és biztonságosan kezelhetik. Ezen túlmenően az általam tervezett szoftver olyan modern funkcionalitással rendelkezik, mind felhasználói mind IT-biztonsági oldalról, amely a kor szellemének megfelel. A szolgáltatásom egy REST API-on keresztül kommunikál bármely megrendelő külső rendszerével.

TDK dolgozatomban a megvalósításra ajánlottam módszertant, technológiai stacket, definiáltam a hostingszolgáltatást, az adatbázismodelleket, a használatieset-diagramokat és a user storykat. Kitértem továbbá az IT-biztonsági követelményekre és a leggyakoribb sebezhetőségek elleni védelemre. Megterveztem a szoftverfejlesztéshez elengedhetetlen kiegészítő eszközkészleteket is, amellyel a magas szintű kódminőség biztosítható, továbbá a folyamatos és gyors funkcionális fejlesztés kivitelezhető. A munkámban felvázolt webes alkalmazás minimum viable product jelleggel a leadott TDK dolgozat alapján megvalósítható. A TDK dolgozat befejezését követően azonban néhány fontos tanulságot vontam le a bővíthetőséggel és a megvalósítással kapcsolatban.

További fejlesztési lehetőségek:

- SDK kiadása a népszerűbb programozási nyelveken, amelynek segítségével még könnyebben és még gyorsabban integrálódhatna az ügyfelek alkalmazásaihoz.
- Hardveres biztonsági kulcs (például yubikey), mint azonosítási szolgáltató bevezetése.
- OAuth2 szerverfunkciók biztosítása az ügyfelek számára.
- Kiegészítő, csak rendszeradminisztrátorok számára elérhető funkciók fejlesztése.

A megvalósítás legfőbb tanulságai:

- Miként lehet védekezni a leggyakoribb kibertámadási módszerek ellen.
- Mennyire változatos egy szoftver tervezésének folyamata.
- Milyen előnyei vannak egy cloudhosting-szolgáltatónak.
- A scrum mint fejlesztési módszertan pontos alkalmazásának előnyei.
- Fejlesztői eszközkészlet (CI/CD, statikus kódanalízi, megfelelő fejlesztőkörnyezet) meghatározó szerepe.

4 IRODALOMJEGYZÉK

[1] *A magyar nyelv értelmező szótára - Jelszó* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://mek.oszk.hu/adatbazis/magyar-nyelv-ertelmezo-szotara/kereses.php?kereses=Jelsz%C3%B3>

[2] *A magyar nyelv értelmező szótára - Azonosít* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://mek.oszk.hu/adatbazis/magyar-nyelv-ertelmezo-szotara/kereses.php?csakcimben=&szo=AZONOS%C3%8DT&offset=0&kereses=AZONOS%C3%8DT>

[3] *A magyar nyelv értelmező szótára - Rejtjel* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://mek.oszk.hu/adatbazis/magyar-nyelv-ertelmezo-szotara/kereses.php?csakcimben=&szo=REJTJEL&offset=0&kereses=rejtjel+>

[4] NÁNDOR Kovács. A jelszó világnapja - A jelszavak rövid története, az első jelszószivárgás és hasznos tippek biztonságos jelszavakhoz. *Hydroxy.hu* [online]. 5. május 2022 [Letöltés ideje: 2024-04-08]. Elérhető: <https://hydroxy.hu/2022/05/05/a-jelszo-vilagnapja-a-jelszavak-rovid-tortenete/>

[5] WALLEN, Dave. OAuth 2.0: What Is It and How Does It Work? *Spanning* [online]. 12. február 2020 [Letöltés ideje: 2024-04-06]. Elérhető: <https://spanning.com/blog/oauth-2-what-is-it-how-does-it-work/>

[6] MOHAMMED, Alzahraa és Ali adil YASSIN. *Efficient and Flexible Multi-Factor Authentication Protocol Based on Fuzzy Extractor of Administrator's Fingerprint and Smart Mobile Device* [online]. B.m.: Computer Science Department, Education College for Pure Sciences, University of Basrah. Elérhető: https://www.researchgate.net/publication/335709093_Efficient_and_Flexible_Multi-Factor_Authentication_Protocol_Based_on_Fuzzy_Extractor_of_Administrator's_Fingerprint_and_Smart_Mobile_Device

[7] VÁCI Krisztián. Microservice architektúra. *Stylers Group* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://stylersgroup.hu/digitalis-transzformacio-archive/microservice-architektura/>

[8] On-Premises vs Cloud Computing: Pros, Cons, and Cost Comparison. *Intellias* [online]. [Letöltés ideje: 2024-04-07]. Elérhető: <https://intellias.com/cloud-computing-vs-on-premises->

comparison-guide/

[9] Serve your hash with Salt and Pepper for Stronger Account Security. *DEV Community* [online]. 30. június 2023 [Letöltés ideje: 2024-04-06]. Elérhető: https://dev.to/0xog_pg/serve-your-hash-with-salt-and-pepper-for-stronger-account-security-587k

[10] COMMON CRITERIA. *Common Criteria for Information Technology Security Evaluation - Part 2: Security functional components*. 2022

[11] GÁBOR, Halmosi. Hogyan használjuk a User Story-kat a fejlesztések során? *Nitrowise* [online]. 8. november 2021 [Letöltés ideje: 2024-04-06]. Elérhető: <https://hu.nitrowise.com/hogyan-hasznaljuk-a-user-story-kat-a-fejlesztések-során/>

[12] AMAZON WEB SERVICES, INC. és Tod GOLDING. *SaaS Tenant Isolation Strategies* [online]. B.m.: Amazon Web Services, Inc. 2020 [Letöltés ideje: 2024-04-07]. Elérhető: <https://d1.awsstatic.com/whitepapers/saas-tenant-isolation-strategies.pdf>

[13] TALL Stack: Tailwind CSS, Alpine.js, Laravel, and Livewire. One Happy Family. *TALL stack* [online]. [Letöltés ideje: 2024-04-07]. Elérhető: <https://tallstack.dev>

[14] Build software better, together. *GitHub* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://github.com>

[15] RIES, Eric. *Lean Startup - Hogyan tegyük ötleteinket sikeressé és fenntarthatóvá?* B.m.: HVG Könyvek, 2019. ISBN 978-963-304-889-4.

[16] *The Twelve-Factor App* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://12factor.net/>

[17] *AWS Lambda* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/pm/lambda/>

[18] Serverless Database - Amazon Aurora Serverless - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-07]. Elérhető: <https://aws.amazon.com/rds/aurora/serverless/>

[19] Cloud Object Storage - Amazon S3 - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/s3/>

[20] API Management - Amazon API Gateway - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/api-gateway/>

[21] Content Delivery Network - Amazon CloudFront - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-09]. Elérhető: <https://aws.amazon.com/cloudfront/>

- [22] Web Application Firewall, Web API Protection - AWS WAF - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/waf/>
- [23] Bulk Cloud Email Service - Amazon Simple Email Service - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/ses/>
- [24] Message Queuing Service - Amazon Simple Queue Service - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://aws.amazon.com/sqs/>
- [25] DNS Service - Amazon Route 53 - AWS. *Amazon Web Services, Inc.* [online]. [Letöltés ideje: 2024-04-09]. Elérhető: <https://aws.amazon.com/route53/>
- [26] *OWASP Top Ten | OWASP Foundation* [online]. [Letöltés ideje: 2024-04-07]. Elérhető: <https://owasp.org/www-project-top-ten/>
- [27] RAI, Anuj. SSL Pinning. *Medium* [online]. 30. március 2020 [Letöltés ideje: 2024-04-06]. Elérhető: <https://medium.com/@anuj.rai2489/ssl-pinning-254fa8ca2109>
- [28] GitHub Actions documentation. *GitHub Docs* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://docs.github.com/en/actions>
- [29] Dependabot quickstart guide. *GitHub Docs* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide>
- [30] *Prettier · Opinionated Code Formatter* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://prettier.io/index.html>
- [31] *Laravel Pint - Laravel 11.x - The PHP Framework For Web Artisans* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://laravel.com/docs/11.x/pint>
- [32] *Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter* [online]. 5. április 2024 [Letöltés ideje: 2024-04-06]. Elérhető: <https://eslint.org/>
- [33] *Linter IDE Tool & Real-Time Software for Code | Sonar* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://www.sonarsource.com/products/sonarlint/>
- [34] *Code Quality Tool & Secure Analysis with SonarQube* [online]. [Letöltés ideje: 2024-04-06]. Elérhető: <https://www.sonarsource.com/products/sonarqube/>

5 ABSZTRAKT

TDK dolgozatomban egy olyan webes szolgáltatást tervezek meg, amely szoftverfejlesztők számára könnyíti meg a felhasználói fiókok kezelését oly módon, hogy azokat egyszerű REST API-felületen keresztül gyorsan és biztonságosan kezelhetik ezeket a fiókokat. Ezen túlmenően az általam tervezett szoftver olyan modern funkcionalitással rendelkezik, mind felhasználói, mind IT-biztonsági oldalról, amely a kor szellemének megfelel. A szolgáltatásom egy REST API-on keresztül kommunikál bármely megrendelő külső rendszerével.

TDK dolgozatomban a megvalósításra ajánlok módszertant, technológiai stacket, definiálom a hostingszolgáltatást, az adatbázismodelleket, a használatieset-diagramokat és a user storykat. Kitérek továbbá az IT-biztonsági követelményekre és a leggyakoribb sebezhetőségek elleni védelemre. Megtervezem a szoftverfejlesztéshez elengedhetetlen kiegészítő eszközkészleteket is, amellyel a magas szintű kódminőség biztosítható, valamint a folyamatos és gyors funkcionális fejlesztés kivitelezhető.

Kulcsszavak: scrum, cloudhosting, CI/CD, LAMP, Lambda, unit-testing, user story, adatbázismodellezés, UML, használati eset, REST API, IT-biztonság, AWS, microservice-architektúra

6 ABSTRACT

In my thesis, I design a web service that facilitates the management of user accounts for software developers, allowing them to manage these accounts quickly and securely through a simple REST API. In addition, I design software that is cutting-edge and has modern functionality both from a user and IT security point of view. My service communicates with any client's external system through a REST API. In the thesis, I recommend an implementation methodology, define the technology stack, hosting service, database models, use case diagrams and user stories. I also address IT security requirements and protection against the most common vulnerabilities. I design an essential complementary software development toolset that ensures high code quality and enables continuous and rapid functional development.

Keywords: scrum, cloud hosting, CI/CD, LAMP, Lambda, unit testing, user story, database modeling, UML, use case, REST API, IT security, AWS, microservice architecture

7 ÁBRAJEGYZÉK

1. ábra - OAuth alapú azonosítás működése [5].....	9
2. ábra - Többlépcsős azonosítás működése [6]	10
3. ábra - Használatieset-diagram (Forrás: saját ábra)	24
4. ábra - Központi adatbázis ER-diagramja (Forrás: saját ábra)	28
5. ábra - Bérlok adatbázisának ER-diagramja (Forrás: saját ábra).....	29
6. ábra - Cloud architektúra diagram (Forrás: saját ábra).....	37