Szakdolgozat

**FELADATKIÍRÁS**

**Bíró László**

Mérnökinformatikus hallgató részére

Járműkölcsönző rendszer fejlesztése

A szakdolgozat témája egy járműkölcsönző cég teljes rendszerének a felépítése, amelyhez több modern technológiát is szükséges lesz majd felhasználni.

A rendszer fog állni egy folyamatosan futó szerverből, ami Java Spring keretrendszer felhasználásával fog történni, egy adatbázisból és egy kliensalkalmazásból, amelyhez react/react native vagy kotlin nyelvek lesznek felhasználva. A kommunikáció REST API-k, illetve JSON adatstruktúrák segítségével fog megvalósulni.

A felhasználó könnyen képes lesz a lehetőségek közül a neki legmegfelelőbb jármű kiválasztására, az útvonala nyomon követésére és a járművek egyértelmű azonosítására. A járművek azonosításánál lehetősége lesz majd bárkódok, illetve QR kódok leolvasására a gyorsabb kölcsönzés végett, de a manuális beírás is egy lehetőség lesz. A felhasználók adatainak a biztonságos kezelése érdekében többszintű validáció lesz megvalósítva (adatbázis, szerver, kliensalkalmazás).

A hallgató feladatának a következőkre kell kiterjednie:

* Ismerje meg a felhasználandó technológiák lehetőségeit, amelyek segítségével képes lesz elvégezni megoldani a feladat kihívásait.
* Tervezze meg az adatbázis szerkezetét, illetve az alkalmazáson belül adatküldésre használt JSON adatstruktúrák szerkezetét.
* Készítsen el egy működő szerver alkalmazást, amely REST API-k segítségével képes kommunikálni egy kliens alkalmazással.
* Készítsen el egy működő kliens alkalmazást, amely képes lesz majd kommunikálni a szerver alkalmazással.
* Tesztelje az elkészült alkalmazást.

**Tanszéki konzulens:** Kövesdán Gábor, tanársegéd

Budapest, 2022. október 6.

Dr. Charaf Hassan

egyetemi tanár

tanszékvezető



**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Bíró László

**Járműkölcsönző rendszer fejlesztése**

Konzulens

Kövesdán Gábor

BUDAPEST, 2022

**Tartalomjegyzék**

[Szakdolgozat 1](#_Toc120402766)

[Összefoglaló 6](#_Toc120402767)

[Abstract 7](#_Toc120402768)

[1 Köszönetnyílvánítás 8](#_Toc120402769)

[2 Bevezetés 10](#_Toc120402770)

[**2.1 Előzmények** 10](#_Toc120402771)

[**2.2 Szakdolgozatom témája** 12](#_Toc120402772)

[3 Felhasznált technológiák 15](#_Toc120402773)

[**3.1 Java** 15](#_Toc120402774)

[**3.2 React native** 15](#_Toc120402775)

[**3.3 React native hooks** 15](#_Toc120402776)

[**3.4 JSON** 15](#_Toc120402777)

[**3.5 JSON APIs** 16](#_Toc120402778)

[**3.6 Visual studio code** 16](#_Toc120402779)

[**3.7 IntelliJ IDEA** 16](#_Toc120402780)

[**3.8 Gradle** 17](#_Toc120402781)

[**3.9 Spring keretrendszer** 17](#_Toc120402782)

[3.9.1 Spring Boot 18](#_Toc120402783)

[3.9.2 Spring Security 18](#_Toc120402784)

[**3.10 GIT** 19](#_Toc120402785)

[3.10.1 Github 19](#_Toc120402786)

[3.10.2 Source tree 20](#_Toc120402787)

[**3.11 Kotlin** 20](#_Toc120402788)

[3.11.1 Retrofit2 21](#_Toc120402789)

[3.11.2 Okhttp3 21](#_Toc120402790)

[**3.12 Android studio** 22](#_Toc120402791)

[3.12.1 Android Emulátor 22](#_Toc120402792)

[3.12.2 Sdk és sdk manager 22](#_Toc120402793)

[**3.13 Node JSx** 23](#_Toc120402794)

[4 Szerver alkalmazás 24](#_Toc120402795)

[**4.1 A rendszer felépítése** 24](#_Toc120402796)

[4.1.1 A domainek 24](#_Toc120402797)

[4.1.2 A szolgáltatások 24](#_Toc120402798)

[4.1.3 A vezérlők (controllers) 24](#_Toc120402799)

[4.1.4 A biztonság 24](#_Toc120402800)

[4.1.5 A konfigurációk 24](#_Toc120402801)

[4.1.6 A validációk 24](#_Toc120402802)

[4.1.7 A tesztelés 24](#_Toc120402803)

[5 Kliens alkalmazás 25](#_Toc120402804)

[**5.1 Az applikáció felépítése** 25](#_Toc120402805)

[5.1.1 A kapcsolatfelvétel 25](#_Toc120402806)

[5.1.2 A belépés 25](#_Toc120402807)

[5.1.3 A regisztráció 25](#_Toc120402808)

[5.1.4 A módosítás 25](#_Toc120402809)

[5.1.5 A járművek listája 25](#_Toc120402810)

[5.1.6 Egy jármű tulajdonságai 25](#_Toc120402811)

[5.1.7 A szerződéskötés 25](#_Toc120402812)

[5.1.8 Az info oldal 25](#_Toc120402813)

[5.1.9 A kilépés 25](#_Toc120402814)

[5.1.10 A validációk 26](#_Toc120402815)

[5.1.11 A logolás 26](#_Toc120402816)

[6 Irodalomjegyzék 27](#_Toc120402817)

[Függelék 29](#_Toc120402818)

**Hallgatói nyilatkozat**

Alulírott **Bíró László**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 10. 24

...…………………………………………….

Bíró László

**Összefoglaló**

A szakdolgozat témája egy járműkölcsönző cég teljes rendszerének a felépítése, amelyhez több modern technológiát is szükséges lesz majd felhasználni.

A rendszer fog állni egy folyamatosan futó szerverből, Java Spring keretrendszer felhasználásával, egy adatbázisból és egy kliensalkalmazásból, amelyhez react/react native vagy kotlin nyelveket szeretnék felhasználni. A kommunikáció REST API-k, illetve JSON adatstruktúrák segítségével fog megvalósulni.

A szakdolgozat elkészítése során a felhasználandó technológiák lehetőségeit kihasználva a kliens többfajta jármű közül biztonságosan választhat. Könnyen képes lesz a lehetőségek közül a neki legmegfelelőbb jármű kiválasztására és a járművek egyértelmű azonosítására. A járművek azonosítására lehetőség lesz majd barkódok, illetve QR kódok leolvasására a gyorsabb kölcsönzés végett, de a manuális beírásra is lehetőség lesz. A felhasználók adatainak a biztonságos kezelése érdekében többszintű validáció lesz megvalósítva (adatbázis, szerver, kliensalkalmazás). A beléptetést pedig a Java Spring lehetőségeit kihasználva szeretném még biztonságosabbá tenni. (Spring Security)

Összefoglalva a szakdolgozatom témája kiterjed egy komplex rendszer minden fontosabb elemére, ezek stabil megalkotására, biztonságossá tételére és a felhasználók szemszögéből a minél egyszerűbb kezelhetőségre a legmodernebb technológiák és módszerek felhasználásával.

**Abstract**

The topic of the thesis is the construction of the entire system of a vehicle rental company, for which it will be necessary to use several modern technologies.

The system will consist of a continuously running server using the Java Spring framework, a database, and a client application for which I would like to use react/react native or kotlin languages. Communication will be implemented using REST APIs and JSON data structures.

The client can safely choose from several types of vehicles. In addition, the client will easily be able to choose the vehicle that suits him best and clearly identify the vehicles. To identify the vehicles, it will be possible to read barcodes and QR codes for faster rental, but it will also be possible to enter them manually. In order to safely manage user data, multi-level validation will be implemented (database, server, client application), and I would like to make access even more secure by taking advantage of Java Spring. (Spring Security)

In summary, the topic of my thesis covers all the most important elements of a complex system, their stable creation, making them safe and making them as simple as possible from the users' point of view using the most modern technologies and methods.

1. **Köszönetnyílvánítás**

Ebben a fejezetben szeretném köszönetemet kifejezni a szakdolgozatom elkészítése során kapott anyagi, illetve erkölcsi segítséget és támogatást. Ebben több személyt és intézményt is megszeretnék említeni, mert nélkülük valószínűleg nem jutottam volna el idáig.

Első sorban meg szeretném köszönni az édesapámnak, aki már nincsen köztünk, de számomra olyan, mintha végig mellettem lett volna, a jó és főleg a rossz időkben. A tőle kapott erkölcsi és anyagi támogatást már sosem tudom meghálálni sajnos, ezért szerettem volna külön megemlíteni.

Ezenkívül természetesen meg szeretném köszönni édesanyámnak, hogy mindig ott volt mellettem, így mindig volt, akivel megbeszéljem a dolgokat, még akkor is ha 1000 km volt a távolság és természetesen minden rokonomnak aki valamilyen módon támogatott egyetemi éveim alatt. Tudom néha erejükön felüli mértékben támogattak ezen évek során.

Megköszönném Dr. Pataricza Andrásnak, aki a témalaboratórium, illetve önálló laboratórium tantárgyaknál konzulensem volt a korábbi tanévben, aki értékes és hasznos megjegyzéseivel, észrevételeivel nagy mértékben a segítségemre volt. Elmagyarázta hogyan lehet jól beosztani az időmet, hogyan lehet egy dokumentumot jól megszerkeszteni, egy prezentációt elkészíteni és bemutatni, illetve a szakirodalom kutatást megfelelő precizitással elvégezni. Elmagyarázta, hogy nem az a jó programozó, aki sok nyelvet ismer, hanem aki jól ismeri a módszereket, az elméleteket, és tudja milyen programozóként gondolkodni, Hiszen a nyelvek változnak, illetve mindig jönnek ki újak, de a mögöttük lévő elméleti háttér állandó.

Szeretném még megköszönni jelenlegi konzulensemnek Kövesdán Gábornak, aki bevezetett a Java szerverek és a Spring működésébe. Sok jó tanáccsal és hasznos észrevétellel látott el a projekt készítése közben és mindig tudtam válaszolt a kérdéseimre, ha valahol elakadtam. Az ő szakmai hozzáértése és hasznos megjegyzései nagy mértékben a segítségemre voltak a szakdolgozatom és a hozzá tartozó szerver-kliens alkalmazásom elkészítése során.

Köszönettel tartozok még a csíkszeredai Magic Solutions srl. munkatársainak, akik bevezettek a Java programozás és a Java szerverek világába régi és új Java technológiák segítségével megalkotott projektjeiben. Nagyon pozitívan fogadtak a munkába állásom kezdetén és a felmerülő kérdéseimre készséggel válaszoltak, illetve segítettek a kezdeti nehézségek leküzdésében.

Legvégül, de persze nem utolsó sorban, pedig szeretném megköszönni a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatika Karának és valamennyi oktatójának, dolgozójának a lelkiismeretes munkáját, erőfeszítését, amellyel a hallgatók képzését, tanulását, a versenyképes tudás megszerzését bármilyen módon is támogatják, hiszen nélkülük egyikünk sem juthatna el idáig, vagy érhetne el komolyabb eredményeket az életben szakmai téren.

1. **Bevezetés**

A szakdolgozatom

* 1. **Előzmények**

A Budapesti Műszaki egyetemen eltöltött évek során különböző programozási nyelvekkel ismerkedtem meg. Például: C, C++, Assembly, C#, Javascript, Prolog, Haskell, Answer set programming, és még sorolhatnám. Ezeket legalább alapszinten sajátottam el, van amelyiket mélyebben el tudtam sajátítani, és pénzt is tudtam keresni vele. Leginkább az Objektum Orientált (OO) szemlélet miatt kedveltem meg a Java és C++ nyelveket, de a legtöbbet a Java nyelvvel foglalkoztam.

Korábban már voltak ismereteim Java szerverekről, főleg régebbi konstrukciókról, amelyek Java ee szerveroldali programozási platform felhasználásával történtek Apache Ant projektépítő eszköz és Wildfly 10-es alkalmazásszerverrel. Fejlesztői környezetnek az Eclipse-et használtuk. Engem egy már kb 18 éve folyamatosan futó projekthez raktak, amelyen fejlesztéseket kellett végezzek, illetve szuport feladatokat láttam el. Ilyen régi technológiák megismerése után felüdülés volt számomra megismerni egy modern Java szerver oldali keretrendszert, a Spring-et. Szerintem jó dolog egy új technológiával foglalkozni, de ennek az előnyeit csak akkor tudja igazán értékelni az ember, ha tudja, hogy hogyan jutottunk el idáig, mit használtak ezelőtt a programozók a hasonló problémák megoldása során. Ezeknek a tudása elengedhetetlen egy komplex rendszer megalkotásához, a szerkezet megfelelő kialakításához és a problémák előrelátásához.

Több Java programozási nyelvvel kapcsolatba hozható tantárgyat is hallgattam. Ezek között volt a Kotlin programozási nyelvvel általánosan foglalkozó tárgy is, amely épít a Java programkönyvtár részeire, 2011 ben fejlődött ki. A Mobil- és webes szoftverek tárgy folyamán egy mobil eszközre fejlesztett alkalmazást is kellett készíteni Kotlin programozási nyelv felhasználásával. Az Alkalmazásfejlesztési környezetek tantárgyban mélyebben megismerkedtek a Java szerverek világával, pontosabban a Spring keretrendszerrel. Ezt a keretrendszert választottam a szerveralkalmazásom felépítéséhez is.

A szakdolgozat megírásától azt várom el, hogy az ismereteim nőjenek, és a szakmai tudásom gyarapodjék a Java szerver és a hozzá tartozó kliens alkalmazás architektúrán alapuló projektek terén. Ezek az ismeretek sokat segíthetnek a későbbiek során más hasonló technológiák elsajátításakor. Az fejezetek megírása közben természetesen szeretném a saját tudásomat is elmélyíteni, illetve tovább bővíteni, valamint a téma előzetes kutatása során új ismeretekkel kiegészíteni, így a szakdolgozatom teljességét biztosítani.

Az alkalmazásszerver olyan infrastruktúra-szoftver, amely végrehajtási környezetet kínál az alkalmazás-összetevők számára. A kifejezés először a webes alkalmazások területén jelent meg. Szigorúan véve az alkalmazáskiszolgáló által üzemeltetett összetevők nem egyszerű eljárások vagy szkriptek, hanem valódi szoftverkomponensek, amelyek megfelelnek egy komponensmodellnek. Az első objektumorientált alkalmazásszerver a WebObjects volt, az Apple megoldása. 2001-ig az Objective-C programozási nyelvet használta, és jelenleg tiszta Java alkalmazásszerver.

A Java az alkalmazásfejlesztés egyik legnépszerűbb és leggyakrabban használt fejlesztési nyelve és számítási platformja. Az iskolában több nyelv között a Java nyelv elsajátítása az elsődleges cél. A Java rugalmas és felhasználóbarát, számos alkalmazás fejlesztőjének kedvence – legyen szó akár közösségi médiáról, webről, játékalkalmazásokról, akár hálózatkezelésről vagy nagyvállalati alkalmazásokról. Bár a Java könnyen használható és könnyebben tanulható más nyelvekhez képest, a Java-alkalmazások létrehozásának, hibakeresésének és telepítésének bonyolultsága szédítő magasságokba emelkedett.

A Spring egy nyílt forráskódú projekt, amely leegyszerűsített, moduláris megközelítést biztosít az alkalmazások Javás létrehozásához. A Spring-projektek családja 2003-ban indult el válaszként a korai Java-fejlesztés bonyolultságára, támogatást nyújtva a Java-alkalmazások fejlesztéséhez. Maga a Spring név általában az alkalmazás keretrendszerére, esetleg a projektek vagy modulok teljes csoportjára utal.

A Spring Boot a Spring-keretrendszer kiterjesztéseként elkészült egyik modul. Ez egy keretrendszer, aminek segítségével könnyen és gyorsan írhatunk szerver oldali Java alkalmazásokat. A Spring-re épül, ami egy összetett, sok modulból álló Framework. A Spring Boot pedig már a Spring által kínált eszközökből összeállított “váz”, ami egyszerűen bővíthető.

Ezek a keretrendszerek szinte bármely esetben csökkenteni tudják a fejlesztő terhelését – akár mobil- és webalkalmazásokat fejlesztenek, akár asztali számítógépekkel és API-kkal dolgoznak. Az én esetemben éppen egy mobilalkalmazásról és egy alkalmazás szerverről van szó, amelyek REST API hívások segítségével kommunikálnak egymással és az adatokat egy adatbázisban tárolják el.

A számítógépes kommunikációra többféle szabványt dolgoztak ki és alkalmaznak. Az internet szempontjából lényeges TCP/IP protokoll-család két fontos szabványa az UDP és a TCP:

* UDP (user datagram protocol): pl. Skype, sportközvetítés, VOIP - ez gyorsabb, de csak áramlik az adat, mint a folyó (stream); ez megenged hibákat, ki-kihagy a Skype-beszélgetés stb ….
* TCP (transmission control protocol): pontosabb, de lassabb, mert van visszajelzés, ha nem érkezett meg egy csomag, akkor újraküldi; így történik a honlapok böngészése, mobilvásárlás.

A számítástechnikában a kliens-szerver egy olyan szoftverarchitektúra-modell, amely két részből áll: kliensrendszerekből és szerverrendszerekből. Ezek mindenképp valamilyen számítógépes hálózaton keresztül kommunikálnak egymással, amely lehet az interneten, de lehet akár lokálisan is.

* 1. **Szakdolgozatom témája**

A szakdolgozatom elkészítése alatt fejlesztettem egy komplex kliens-szerver rendszert, amelynek segítségével egy járműkölcsönző cég feladatait lehet menedzselni. Ehhez olyan technológiák felhasználását terveztem, amelyek modernek, de a céges világban már elég stabilnak és bejáratottnak vannak elismerve. Lényege az, hogy a felhasználó a különböző típusú járművek közül választva egy felhasználóbarát mobilalkalmazás segítségével képes elvégezni a kölcsönzést, bizonyos esetekben csak egy QR kódot beolvasva. Ezzel a módszerrel sok papírmunka spórólható meg, hiszen egy regisztrált felhasználóról már rengeteg adatot tárol az adatbázis, így a szerződéskötés egy bejelentkezett felhasználó esetén már néhány kattintás alatt elvégezhető jóformán emberi beavatkozás nélkül.

A mai világban már rengeteg helyen elterjedt az a módszer, hogy a hosszadalmas papírmunkával járó procedúrákat egy mobilos alkalmazás segítségével valósítjuk meg ezzel könnyítve meg az éles adatok áttekinthetőségét, a folyamat átláthatóságát és logok segítségével az esetleges problémák minél könnyebb és gyorsabb megoldását is. A technológiák közötti kommunikáció mostanság egyre hatékonyabbá válik, így még jobban kilehet használni ennek az előnyeit.

A kliens alkalmazásom egy Android rendszerre írt alkalmazás, mely Rest API-k segítségével kommunikál a szerver alkalmazással. Programozási nyelvnek a Kotlin-t választottam, amely a háttérkutatásaim alapján egy elég modern nyelv így nagyon dinamikusan és hatékonyan lehet benne fejleszteni. A nagy népszerűség és a komoly fejlesztőgárda következtében egy elég kifejlett programozási tekinthető. Fejlesztés során, ha elakadtam valahol általában elég hamar tudtam kapni megoldást az interneten, vagy egy hasonló példát, így sikeresen tudtam venni az elém kerülő akadályokat. Ami még nagyon megtetszett a Kotlin nyelvben, az az erősen típusos tulajdonsága, amely rengeteget a segít számomra a forráskód átláthatóságában és a hibakeresés folyamatában.

Az applikáció segítségével meg lehet tekinteni, hogy milyen járművek állnak a rendelkezésünkre, lehetőségünk van a járművek közötti szűrésre, illetve köthetünk egy szerződést is a céggel a kiválasztott járműre vagy járművekre. Ehhez először regisztrálni kell a felhasználónkat, amely alatt adatokat adunk meg magunkról, amely segítségével megkülönböztethetővé válunk másoktól az applikáció előtt. A regisztráció után az újonnan készült felhasználónkkal rögtön be tudunk lépni az applikációba. Ezután már dinamikusan és egyszerűen lehet úgymond böngészni a bérlendő járművek között és két három kattintás után kibérelni egy vagy akár több járművet is egyszerre. Ezenkívül lehetőségünk van a saját profilunk módosítására és bizonyos grafikonok/táblázatok/információk megtekintésére.

A regisztrációkor az adataink a szerveren tárolódnak el és teljes biztonságban vannak a Spring Security technológiáinak köszönhetően. A belépés után a szerver generál egy tokent, és ezután az adatok további lekérésekor ezt a tokent használja, mert másként nem is lehet a szerveren lévő bizonyos adatokhoz hozzáférni. Erről a szakdolgozat további részében részletesebben lehet olvasni.

Ha valamilyen gond adódik, vagy valamit nem írtunk be helyesen, akkor az applikáció egy felugró ablak segítségével értesít a megfelelő információkról, vagy figyelmeztet, ha gond adódik például a szerver kapcsolattal. Bármi is történik a kliens alkalmazás nem válik instabillá és így az adatvesztésnek, vagy a hibás adat bevitelének is nagyon kevés az esélye. Ez köszönhető a többszintű validációnak is, amelyet szintén kifejtek majd a szakdolgozatom további részében.

Az applikációból való kijelentkezés során a token törlődik a teljes kliens alkalmazásból, nyoma sem marad kliens oldalon a felhasználónak. Ez egy biztonságos működést eredményez, hiszen minden bejelentkezéskor egy teljesen új token jön létre. Így, ha valaki meg is szerezné a tokent valahogy egy újabb bejelentkezés érvénytelenné és használhatatlanná tenné a régit, ezenkívül egy tokennek van egy lejárai ideje, amely a szerverben generáláskor állítódik be.

1. **Felhasznált technológiák**

Itt szeretném összefoglalni a projektem összerakása során felhasznált technológiákat. Amennyire ismereteim engedték megpróbáltam a legmodernebb technológiákat felhasználni, persze a választásaim során a technológiák kiforrottsági szintjét is figyelembe vettem, hiszen egy régebbi technológiához szakirodalmat és bevált módszereket is könnyebben talál az ember.

* 1. **Java**

A Java általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett az 1990-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle. A Java alkalmazásokat jellemzően bájtkód formátumra alakítják, de közvetlenül natív kód is készíthető Java forráskódból. Ebből fejlődött ki később a Kotlin nyelv is.

* 1. **JSON**

A JSON egy kis méretű, szöveg alapú szabvány, ember által olvasható adatcserére. A JavaScript szkript nyelvből alakult ki egyszerű adatstruktúrák és asszociatív tömbök reprezentálására. A JavaScripttel való kapcsolata ellenére nyelvfüggetlen, több nyelvhez is van értelmezője. A React native Javascript alapú, így ehhez tökéletesen kompatibilis.

* 1. **JSON APIs**

Sajnos számunkra nem elég egy szép objektum, amiben adatok vannak ezt valahogy el is kell küldeni például valamilyen esemény bekövetkezésének hatására. Erre szolgálnak az API-k. A JSON API-modul egy megvalósítást tesz közzé adattárolókhoz és adatstruktúrákhoz, például entitástípusokhoz, kötegekhez és mezőkhöz.

* 1. **Visual studio code**

A Visual Studio Code egy ingyenes, nyílt forráskódú kódszerkesztő, melyet a Microsoft fejleszt Windows, Linux és OS X operációs rendszerekhez. Támogatja a hibakeresőket, valamint beépített Git támogatással rendelkezik, továbbá képes az intelligens kódkezelésre az IntelliSense segítségével.

A Visual Studio Code az Electron nevű keretrendszeren alapszik, amellyel asztali környezetben futtatható Node.js alkalmazások fejleszthetőek. Ugyanakkor a Visual Studio Code nem az Atom forkja, hanem a Visual Studio Online szerkesztőn alapszik (fejlesztési neve: "Monaco").

* 1. **IntelliJ IDEA**

Az IntelliJ IDEA egy Java nyelven írt integrált fejlesztői környezet (IDE) Java, Kotlin, Groovy és más JVM - alapú nyelveken írt számítógépes szoftverek fejlesztésére. A JetBrains (korábbi nevén IntelliJ) fejlesztette ki, és Apache 2 licensszel közösségi kiadásként és szabadalmaztatott kereskedelmi kiadásban is elérhető. Mindkettő felhasználható kereskedelmi fejlesztésre.

Első verziója 2001 januárjában jelent meg, és az egyik első elérhető Java IDE volt, amely fejlett kódnavigációs és kódújrafeldolgozási képességekkel rendelkezik. 2009-ben a JetBrains kiadta az IntelliJ IDEA forráskódját a nyílt forráskódú Apache License 2.0 alatt. 2014 decemberében a Google bejelentette az Android Studio 1.0-s verzióját, amely egy nyílt forráskódú IDE Android - alkalmazásokhoz, amely a nyílt forráskódú közösségi kiadáson alapul.

* 1. **Gradle**

A Gradle projektépítő eszköz, amely az Apache Ant és az Apache Maven koncepciójára épül, és bevezeti a Groovy-alapú doménspecifikus nyelvet, a projektkonfiguráció deklarációját a hagyományosabb XML forma helyett. A Gradle a feladatok futási sorrendjének meghatározására irányított körmentes gráfot használ.

Szakdolgozatomhoz tartozó projektem elkészítése során, mint a szerver, mint a kliens alkalmazás esetén ezt a projektépítő eszközt választottam. Mivel az iskolában már találkoztam ezzel nem jelentett különösebb problémát a használata.

Minden Gradle projekthez deklarált függőség egy adott hatókörre vonatkozik. Például néhány függőséget kell használni a forráskód fordításához, míg másoknak csak futás közben kell elérhetőnek lenniük. Sok Gradle beépülő modul előre meghatározott konfigurációkat ad a projekthez.

* 1. **Spring keretrendszer**

A képen asztal látható

Automatikusan generált leírásA Spring Framework átfogó programozási és konfigurációs modellt biztosít a modern Java-alapú vállalati alkalmazásokhoz – bármilyen telepítési platformon.

A fenti táblázatban látható előnyök miatt a Spring közkedveltebbé vált, könnyebben használható és gyorsabban is fejlődik. Az EJB-nek is meg vannak az előnyei, mivel az elosztott tranzakciókat jobban kezeli.

A Spring keretrendszer alapja a függőséginjektálás (Dependency Injection – DI). Függőséginjektáláskor a keretrendszer kezeli a konfigurációban megadott objektumok példányosítását, a keretrendszertől kérjük el a konkrét példányokat, a használt objektum konfigurációs beállítássá válik és nem szükséges újrafordítás sem, csak a konfigurációs fájl szerkesztése. Korábban ezt a koncepciót nevezték Inversion of Control-nak (IoC).

A Spring gerincét adják a Bean-ek, amely a keretrendszer kulcsfogalma. A Bean-ek konfigurációja történhet XML fájllal, újrafordítás sem szükséges. Ezenkívül történhet szórványos annotációkkal és konfigurációs fájlokkal is.

A Spring Annotations lehetővé teszi a függőségek konfigurálását és a függőségek beillesztésének megvalósítását Java programokon keresztül. Pl.: @Bean, @Controller, @RequestBody, @Service, @Repository, @Configuration, @Entity, @Id, @GeneratedValue(strategy = GenerationType.IDENTITY), @ManyToOne, @Email, @PreAuthorize("isAuthenticated()"), @RestController, @Override, @SpringBootApplication. Ez egy nagyon fontos eszköz, hanem a legfontosabb egy Spring alkalmazás elkészítésénél, de már sok programozási nyelvben lehet használni (Pl. Kotlin). Annotációkat meg is lehet írni, akár egy föggvényt, lehet velük velidálni, funkciót lehet adni egy osztálynak, adatbázis szerkezetet megvalósítani stb. Tehát egy nagyon sokoldalú és hasznos eszköz.

### Spring Boot

A Spring nagyon sok modult tartalmaz, a valóságban viszont kialakultak ezeknek gyakori részhalmazai: Pl.: webalkalmazások esetén nagyjából ugyanazokat a komponenseket szokás használni. Vannak részek, amelyeket szinte mindegyik Spring alkalmazás használ, Pl.: naplózás.

A Spring Boot a Spring egyfajta tovább gondolása. A Spring Boot lehetővé teszi az olyan szerverek közvetlen beágyazását, mint a Tomcat, a Jetty vagy az Undertow. A Spring Boot leegyszerűsíti az konfigurációkat azáltal, hogy egyedi formálható kezdeti függőségeket biztosít. A Spring Boot kezdőkészletekből, ún. starterekből áll. Egy-egy starter tartalmazza a tipikus, bizonyos tulajdonságú alkalmazás komponenseket. Egy program tetszőleges számú Spring Boot startert használhat. A starterek nevei kezdődhetnek a következőképpen: spring-boot-starter. Pl. a spring-boot-starter-web a webalkalmazások függőségeit tartalmazza, valamint egy beépített webszervert is, ami beállítás nélkül automatikusan elindul. Tehát a Spring Boot egy könnyű keretrendszer, amely leveszi a legtöbb munkát a Spring-alapú alkalmazások konfigurálásából.

### Spring Security

A Spring Security egy Java/Java EE keretrendszer, amely hitelesítést, engedélyezést és egyéb biztonsági funkciókat biztosít a vállalati alkalmazások számára. Egy hatékony és nagymértékben testre szabható hitelesítési és hozzáférés-felügyeleti keretrendszer. A projektet 2003 végén „Acegi Security” néven Ben Alex indította el, és 2004 márciusában az Apache License alatt nyilvánosan kiadták. A Spring Security igazi ereje abban rejlik, hogy milyen egyszerűen bővíthető, hogy megfeleljen az egyéni követelményeknek.

Védelmet biztosít az olyan támadások ellen, mint a munkamenetrögzítés, a kattintástörés (clickjacking), a webhelyek közötti kérések hamisítása stb. Servlet API integrációval rendelkezik, és opcionálisan integrálható a Spring Web MVC-vel.

Spring Security segítségével felépíthető egy biztonságos hitelesítési rendszer, amely segítségével a felhasználó akár egy felhasználónév és egy jelszó segítségével biztonságosan lekérhet adatokat a szerverről, anélkül, hogy hitelesítési adatai rossz kezekbe kerülnének. Ehhez nyújt segítséget a JWT.

A JSON Web Token (JWT) egy nyílt iparági szabvány, amelyet két entitás – általában egy kliens (például az alkalmazás frontendje) és egy szerver (az alkalmazás háttérrendszere) – közötti információmegosztásra használnak. JSON-objektumokat tartalmaznak, amelyek rendelkeznek a megosztandó információkkal.

A szerveren a megfelelő beállítások elvégzése után le lehet generálni az adott jelszóból kriptográfia titkosítási módszereket használó tokent, amelyet visszaküldve a kliens alkalmazásnak az már biztonságosan tudja használni akár személyes adatok lekérésére. Egy tokennek be lehet állítani egy időkeretet, amely alatt használható, ezen kívül pedig újbóli bejelentkezés szükséges egy újabb token legeneráláshoz. Én a projektemben Bearer tokent használtam, amelyet a szerver generál egy bejelentkezési kérelemre és a tokent a JSON válasz fejlécében küldi vissza a kliensnek.

* 1. **GIT**

A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt, így a kicsitől a nagyon nagy projektig mindent gyorsan és hatékonyan kezel. A Git könnyen megtanulható, és villámgyors teljesítménnyel rendelkezik. Felülmúlja az olyan SCM-eszközöket, mint a Subversion, CVS, Perforce és ClearCase olyan funkciókkal, mint az olcsó helyi elágazás, kényelmes állomásozási területek (staging area) és többféle munkafolyamat.

A Gitet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. A „git” nevet Linus Torvalds némi iróniával a brit angol szleng kellemetlen személyt jelentő szavából eredezteti. „Egy egoista szemétláda vagyok, és minden projektemet magam után nevezem el.Első kiadása 2005.04.07.-én készült el, és jelen állás szerint a legutolsó stabil verzió 2021.06.04-én készült el. Minden Git munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revíziókövetési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől. A Git karbantartásának felügyeletét jelenleg Junio Hamano látja el.

### Github

A GitHub egy kódtárhely a verziókezeléshez és a fejlesztők/csapatok/cégek közötti együttműködéshez. Segítségével az arra jogosultak együtt dolgozhatnak egy adott kódbázison, lekövethetik a másik csapattag munkáját, megjegyzéseket javaslatokat tehetnek a többiek számára.

A GitHub egy webalapú felület, amely a Git-et, a nyílt forráskódú verziókezelő szoftvert használja, amely lehetővé teszi, hogy egyszerre többen is módosítsák a weboldalon lévő adatokat. A módosítás történhet Command Promptból (cmd) is, de sok fejlesztő környezet is támogatja a githhub repository-t (szoftvertároló).

### Source tree

A Sourcetree egy ingyenes grafikus felhasználói felület (GUI) asztali kliens eszköz, amely leegyszerűsíti a Git-tárolókkal való interakciót, így teljes mértékben a kódolásra lehet koncentrálni. Ez a grafikus felhasználói felület megkönnyíti a tárolók megjelenítését és kezelését. Elég sok funkciója van, amelyek segítségével könnyen lehet branch-eket létrehozni Minden új fejlesztés külön branch-ben történik, úgy a visszakövethetőség elég magas szinten megvalósítható. Ha valami módosítás kell, akkor pedig meg lehet keresni azt a régi branchet, amelyen elkezdődött az adott fejlesztés, majd az azon való módosítás után összemergelhető a fő branchel, amelyről lebuildelődik az adott projekt.

* 1. **Postman**

A Postman egy API-kliens, amely megkönnyíti a fejlesztők számára az API-k létrehozását, megosztását, tesztelését és dokumentálását. Ezzel a nyílt forráskódú megoldással a felhasználók egyszerű és összetett HTTP/s kéréseket hozhatnak létre és menthetnek, valamint elolvashatják válaszaikat. 2022 áprilisában a Postman több mint 20 millió regisztrált felhasználóval és 75 000 nyitott API-val rendelkezik, amely a világ legnagyobb nyilvános API-központja.

A Postman 2012-ben indult Abhinav Asthana szoftvermérnök mellékprojektjeként, aki az API tesztelését akarta egyszerűsíteni, miközben a Yahoo Bangalore-nál dolgozott. Ingyenes alkalmazásként elindította a Postmant a Chrome Internetes áruházban. Az alkalmazás használatának növekedésével Abhinav korábbi kollégáit, Ankit Sobtit és Abhijit Kane-t toborozta, hogy segítsenek létrehozni a Postman Inc.-t. A három társalapító vezeti ma a céget, Abhinav vezérigazgatóként, Sobti pedig technológiai igazgatóként dolgozik.

* 1. **Thunder client**

A Thunder Client egy könnyű Rest API-kliens-bővítmény a Visual Studio Code -hoz, amelyet Ranga Vadhineni készített kézzel, egyszerű és letisztult dizájnnal.

Nem kell elfelejteni, hogy a Postman robusztusabb, és az iparági szabványoknak megfelelő szolgáltatások szélesebb választékával rendelkezik. Lehetővé teszi a fejlesztők közössége számára az API-k, munkaterületek és gyűjtemények legnagyobb hálózatának felfedezését a világ minden tájáról.

Ennek ellenére elég gyakran használtam ezt a bővítményt, mivel a szerver fejlesztését Visual Studio Code-ban végeztem el. Számomra minden szükséges ellenőrzést el tudtam végezni ebben a környezetben is, és nem kellett hozzá se bejelentkezés, se egy külön program elindítása, amely nagy előny volt számomra, mert az Android studio és főleg a hozzá tartozó android emulátor elég jól leterhelte a gépemet. Bár ez egy nagyon praktikus bővítmény, azonban a Thunder Client nem teszi lehetővé a Postman által kínált összes optimalizálást (proxykonfiguráció stb.), és nem teszi lehetővé változó szkriptek létrehozását sem.

* 1. **Kotlin**

A Kotlin erősen típusos programozási nyelv, amely Java virtuális gépre és JavaScript kódra is lefordítható. Ez egy modern programozási nyelv, amelyet a professzionális Android-fejlesztők több mint 60%-a használja, és amely segít a termelékenység, a fejlesztői elégedettség és a kódbiztonság növelésében .

A Kotlin programnyelvet 2011 júliusában hozták nyilvánoságra, amit akkor már egy éve fejlesztettek. A fordítót és a hozzá tartozó programokat 2012 februárjában adták ki nyílt forráskódú szoftverként Apache 2.0 licenc alatt. A fő fejlesztői a JetBrains szentpétervári csapata, a nyelv a Szentpétervár közelében található Kotlin-szigetről kapta a nevét. A Jetbrains bevallott motivációja az új nyelv fejlesztésében az, hogy az növelje az IDEA fejlesztőeszköz eladásait.

A Kotlin jobban támogatja az Android fejlesztését, mint a Java, ezenkívül az erősen típusos tulajdonságát is nagyon előnyösnek találom, ezáltal a hibakeresés és a kód olvashatósága is javul. Ezért egy Android kliens alkalmazás elkészítéséhez megfelelőnek találtam.

### Retrofit2

A Retrofit egy típusbiztos REST kliens Androidra, Java-ra és Kotlinra, amelyet a Square fejlesztett ki. A könyvtár hatékony keretrendszert biztosít az API-k hitelesítéséhez és interakciójához, valamint hálózati kérések küldéséhez az OkHttp-vel. Amellett, hogy elegáns szintaxist biztosít, könnyen beilleszthető különböző könyvtárakba.

A Retrofit 2.0-ás verziója előtt, ha a lekért választ nem sikerült elemezni a meghatározott objektumban, a rendszer a hibát hívja meg. A Retrofit 2.0-ban, függetlenül attól, hogy a válasz értelmezhető-e vagy sem, az onResponse mindig meghívásra kerül. Abban az esetben viszont, ha az eredmény nem elemezhető az objektumba, akkor a response.body() nullként tér vissza. Bár az el kell mondjam, hogy miközben ezt használtam, volt olyan eset is, hogy egyszerűen kilépett a függvényből. Ezt külön le kellett kezeljem a kliens alkalmazásomnál.

### Okhttp3

A HTTP a modern alkalmazások hálózati kommunikációjának a módja. Így cserélünk adatokat és médiát. A HTTP hatékony végrehajtása gyorsabbá teszi a dolgok betöltését, és sávszélességet takarít meg.

Az OkHttp egy HTTP-kliens, amely alapértelmezés szerint hatékony:

* HTTP/2 támogatás lehetővé teszi, hogy minden kérés ugyanahhoz a gazdagéphez osszon meg egy socketet.
* A kapcsolatkészlet (Connection pooling) csökkenti a kérés késését (ha a HTTP/2 nem érhető el).
* A GZIP csökkenti a letöltési méreteket.
* Végül pedig a válaszok gyorsítótárazása (cache) teljesen elkerüli a hálózatot az ismétlődő kérések esetén.

Ha a hálózat problémás, az OkHttp csendben helyreáll az általános csatlakozási problémákból. Ha a szolgáltatásnak több IP-címe van, az OkHttp megpróbál alternatív címeket adni, ha az első csatlakozás sikertelen.

Használata egyszerű, request/response API-ja gördülékeny építőkkel és stabilitással készült. Támogatja a szinkron hívásokat és az aszinkron hívásokat is a visszahívásokkal (callbacks).

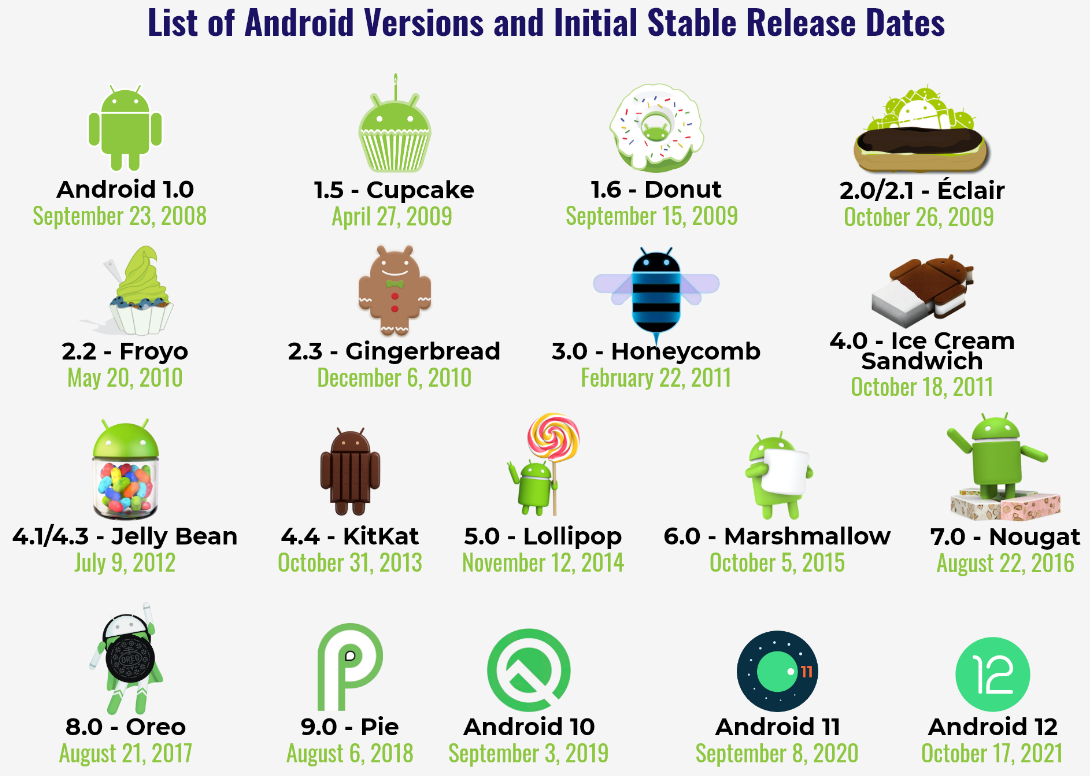
* 1. **Android**

Az Android egy teljes és modern operációs rendszer. Ez egy linux kernelt használó mobil operációs rendszer, amelyet eredetileg mobil eszközökhöz fejlesztettek ki, de manapság már a legkülönbözőbb helyeken használják fel. Például felhasználják, okostelefenokhoz, táblagépeghez, okos TV – khez, okos órákhoz, háztartási berendezésekhez, autókhoz stb. Az Andoidod kifejezetten alkalmazásokhoz fejlesztették ki.

A Google keze alatt lévő Android op. rendszer egy nyílt forráskódú rendszer, ennek köszönhetem azt, hogy létre tudtam hozni egy kliens-t android telefonra, anélkül, hogy fizetnem kellett volna valamilyen szoftverért. Rengeteg okostelefonon elérhető és jelenleg a piac több mint 72% - át uralja az Android operációs rendszer. Ez egy teljesen nyílt platform fejlesztők, felhasználók és az ipar számára.

### Android történelem

Az Androidot Inc.-et 2003 októberében Andy Rubin, Rich Miner, Nick Sears és Chris White alapította Palo Altoban, Kaliforniában, az Egyesült Államokban. 2005-ben a Google felvásárolta, majd a fejlesztését az Open Handset Alliance folytatta 2007-ben. A fejlesztők alapértelmezetten Java nyelven írhatnak rá forráskódot, az eszközt pedig a Google által fejlesztett Java programkönyvtárak vezérlik.

Az Open Handset Alliance (OHA) a cégek üzleti szövetsége a mobileszközök nyílt szabványának kidolgozására. Céljuk olyan technológiák kifejlesztése, amelyek jelentősen csökkentik a mobileszközök és -szolgáltatások fejlesztésének és terjesztésének költségeit. Ahogy a fenti képen látható ez több vállalatot foglal magába.

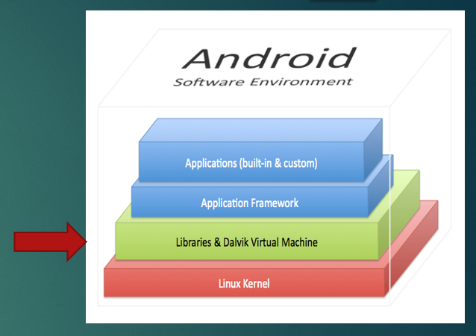
Ahogy a fenti ábrán is látszik az Android operációs rendszernek elég sok stabil verziója jelent meg az első verzió 2008-as bejelentése óta. Habár amikor megjelent egy szűk fanatikusokból álló rétegen kívül - nem nyerte el igazán az átlagfelhasználók tetszését, manapság ez a legelterjedtebb mobil (hordozható) operációs rendszer.

### 5Android architektúra

Az Android operációs rendszer szoftverösszetevők halmaza, amely nagyjából öt részre és négy fő rétegre van felosztva, amint a fenti architektúra diagramon látható.

A képen szöveg, képernyőkép, névjegykártya látható

Automatikusan generált leírásA Linux kernel magába foglalja a biztonságot, illetve a memória és folyamatkezelést. Ez egy bevált driver modell. Hatékony számítási kapacitással és erőforrás kezeléssel rendelkezik. Stabil és bevált operációs rendszer mobil platformhoz.

A Könyvtárak a Linux kernel fölött találhatóak. Ez a réteg különféle C/C++ alapkönyvtárakat és Java alapú könyvtárakat tartalmaz, például Graphics, Media, OpenGL, Surface Manager stb. Az SQLite adatbázis támogatáshoz van. Az SSL (Secure Sockets Layer) egy biztonsági technológia. Az SGL és az OpenGL használható 2D és 3D számítógépes grafikákhoz.

A képen szöveg, névjegykártya, képernyőkép látható

Automatikusan generált leírásAz alkalmazás keretrendszere az az eszközkészlet, amelyet minden alkalmazás használ és Android-alkalmazások létrehozására használunk. Ez teljes egészében Java programozási nyelven van megírva. Tartalmaz különböző típusú szolgáltatási tevékenység menedzsereket, értesítéskezelőket, nézetrendszereket, csomagkezelőket stb., amelyek segítik az alkalmazásunk fejlesztését.

Ezeken kívül van az Android Runtime környezet, amely az Android egyik legfontosabb része. Olyan összetevőket tartalmaz, mint a core libraries és a Dalvik virtual machine (DVM). Főleg az alkalmazás keretrendszer alapját adja, és core libraries segítségével működteti az alkalmazásunkat. A Java virtuális géphez (JVM) hasonlóan a Dalvik Virtual Machine (DVM) is egy regiszter alapú virtuális gép, amelyet kifejezetten androidra terveztek és optimalizáltak annak biztosítására, hogy egy eszköz több példányt is hatékonyan tudjon futtatni.

Az Android keretrendszer tetején alkalmazások találhatók. Minden alkalmazás, mint például az otthoni, beállítások, játékok, böngészők, android keretrendszert használ, amely android futási környezetet és könyvtárakat használ. Az Android futtatókörnyezete és a natív könyvtárak Linux kernelt használnak.

Az Android operációs rendszer fontosabb jellemzői a következők:

* Az Android képes több alkalmazást is futtatni egyszerre.
* Támogatja az optimalizált VGA, 2D és 3D grafikát is.
* Lehetővé teszi a komponensek újra felhasználását és cseréjét.
* Az Android támogatja a Java alkalmazásokat.
* Alapértelmezetten mobileszközökre van optimalizálva.
  1. **Android studio**

Az Android Studio Android-alkalmazásokra optimalizált integrált fejlesztői környezetet (IDE) biztosít az alkalmazáskészítőknek az Android platformra való fejlesztéshez. Alkalmazások készíthetőek itt Android-telefonokhoz, -táblagépekhez, Android okosórákhoz, Android TV-hez és Android Auto-hoz. A strukturált kódmodulok lehetővé teszik, hogy a projektet funkcionalitási egységekre lehessen felosztani, amelyeket önállóan lehet építeni, tesztelni és hibakeresést végezni.

### Android Emulátor

Az emulátor egy olyan számítógépes program vagy hardver, ami más programoknak vagy eszközöknek a környezetét (vagy annak részét) „szimulálja”, vagyis lehetővé teszik az adott rendszerrel nem kompatibilis programok (vagy operációs rendszerek) vagy számítógépek futtatását.

### Sdk és android sdk manager

A szoftverfejlesztő készlet (SDK) olyan eszközkészlet, amelyet (általában) egy hardverplatform, operációs rendszer (OS) vagy programozási nyelv gyártója biztosít. Az SDK-k segítenek a szoftverfejlesztőknek alkalmazásokat létrehozni az adott platformhoz, rendszerhez vagy programozási nyelvhez. Úgy kell elképzelni, mint egy szerszámkészletet, vagy a szerszámokat tartalmazó műanyag zacskót egy szekrényhez, amelyet saját magunknak össze tudunk szerelni, csak ebben az esetben alkalmazásfejlesztéshez vannak. Megvannak a munka elvégzéséhez szükséges építőelemek – vagy fejlesztőeszközök –, és a készlet tartalma az egyes gyártóknál eltérő.

Alapjáraton egy SDK tartalmaz fordítót hibakeresőt és alkalmazásprogramozási felületeket. Egy jó SDK minden olyan összetevőt biztosít, amelyet a fejlesztő szükségesnek találhat, amikor új alkalmazásokat hoz létre az adott termékhez és ökoszisztémájához.

Az sdk manager egy eszköz, amely lehetővé teszi az Android SDK csomagjainak megtekintését, telepítését, frissítését és eltávolítását. Lehet használni paracssorból, illetve Integrált fejlesztői környezetből (IDE). A csomagok letöltését követően mindegyik csomag megtalálható az Android SDK-kezelőjében beállított könyvtárban, vagy megtekinthető az fejlesztői környezetben is. Az utóbbiban az is látható, ha valamelyik eszköz frissítésre szorul (egy kötőjel(-) jelenik meg az

* 1. **Rest API**

A REST API (más néven RESTful API) egy alkalmazásprogramozási felület (API vagy webes API), amely megfelel a REST architektúra stílusának korlátainak, és lehetővé teszi a RESTful webszolgáltatásokkal való interakciót. A REST a reprezentatív állapottranszfer rövidítése, és Roy Fielding informatikus vezette be és definiálta 2000-ben a doktori disszertációjában. Azokat a rendszereket, amelyek eleget tesznek a REST megszorításainak, "RESTful"-nak nevezik. Tehát ez egy szoftver architektúra típus, amely lehetővé teszi skálázható, nagy teljesítményű elosztott hálózati alkalmazások fejlesztésére.

A REST megszorítások a következők:

* kliens-szerver modell,
* egységes interfész,
* állapotmentes kommunikáció,
* réteges felépítés,
* gyorsítótárazhatóság
* kiegészíthetőség (code on demand)
  1. **Bearer token**

A Bearer token, egyfajta biztonsági token az OAuth-ban, amely hozzáférést biztosít a hordozójához. Magyarra fordítva „Hordozó (vivő) hitelesítés” - nek lehet értelmezni, amely elnevezés úgy értelmezhető, hogy „hozzáférést biztosít a token hordozójának”.

A képen szöveg látható

Automatikusan generált leírásA bearer token egy titkosított karakterlánc, amelyet általában a kiszolgáló generál. Jelen esetemben a szerver generálja kérésre és ez is mentődik el jelszó helyett az adatbázisban, mivel visszafejteni is van lehetőség szerver szinten. Az ügyfélnek el kell küldenie ezt a tokent a Authorization fejlécben, amikor kéréseket intéz a védett erőforrásokhoz.

A bearer hitelesítési sémát eredetileg az OAuth 2.0 részeként hozták létre az RFC 6750-ben, de néha önmagában is használják.

* 1. **OAuth 2.0**

Az OAuth 2.0 egy engedélyezési protokoll, amely egy API-kliens számára korlátozott hozzáférést biztosít a webszerveren lévő felhasználói adatokhoz. Az OAuth hitelesítési forgatókönyvre támaszkodik, amely lehetővé teszi az erőforrás-tulajdonos számára, hogy megossza a védett tartalmakat az erőforrás-kiszolgálóról anélkül, hogy megosztaná hitelesítő adatait. Az OAuth 2.0 a kliensfejlesztők egyszerűségére összpontosít, miközben speciális engedélyezési folyamatokat biztosít a webalkalmazások, asztali alkalmazások, mobiltelefonok és okos otthonban lévő eszközök számára.

Ezt a technológiát nagyon sok nagynevű cég előszeretettel használja, közöttük a Google, a GitHub, a Facebook stb. Az OAuth 2.1 egy folyamatban lévő erőfeszítés az OAuth 2.0 és számos gyakori bővítmény új néven való egyesítésére.

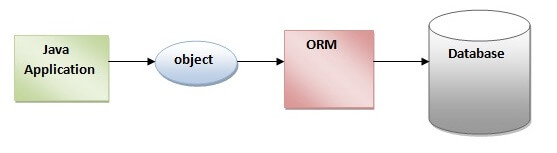
* 1. **JPA**

A Java Persistence API, vagy röviden JPA, egy keretrendszer a Java programozási nyelvhez, melynek fő feladata a relációs adatok kezelése a Java Platform Standard és a Enterprise Editiont használó alkalmazásokban.

Legtöbb esetben az adat nagy része főként relációs adatbázisokban tárolódik, melyekhez sokféle módon hozzáférhetünk a Java programból – ezen módok közül egy a JPA. A JPA tulajdonképpen egy interfészt ad, melyet implementálni lehet. Ugyanakkor léteznek már hozzá mind nyílt forráskódú, mind kereskedelmi forgalomban kapható implementációk is, melyek közül szintén választhatunk.

* 1. **ORM**

Az objektum-relációs leképzés egy programozási technika adatok konvertálására nem kompatibilis típusos rendszerek és objektumorientált programozási nyelvek között. Lényegében egy „virtuális objektum-adatbázist” hoz létre, amit a programozási nyelvben használhatunk. Összehasonlítva a hagyományos módszerrel, az ORM gyakran lecsökkenti a megírandó kód mennyiségét, objektumorientált nyelv és relációs adatbázis közötti adatcsere esetén.



Az ORM hátrányos azokon a területeken, amelyeken az adatbázis specifikus technikák jelentősen optimalizáltak. A legtöbb ORM eszköz nem teljesít jól nagy mennyiségű adat törlése esetén, vagy adattáblák összekapcsolásában. A tárolt eljárások esetleg jobban teljesítenek, de nem hordozhatóak. Az ORM eszközök használata gyakran vezethet rosszul tervezett adatbázisok készítéséhez.

* 1. **Hibernate**

A Hibernate egy nyílt forráskódú objektumrelációs leképezési (ORM) eszköz, amely keretet biztosít az objektumorientált tartománymodellek relációs adatbázisokhoz való leképezéséhez webalkalmazások számára. Ez a technológia Java platformra van kifejlesztve. A Hibernate a JPA (Java Persistence API) specifikációit valósítja meg az adatok megmaradása érdekében.

1. **A projektről általánosan**

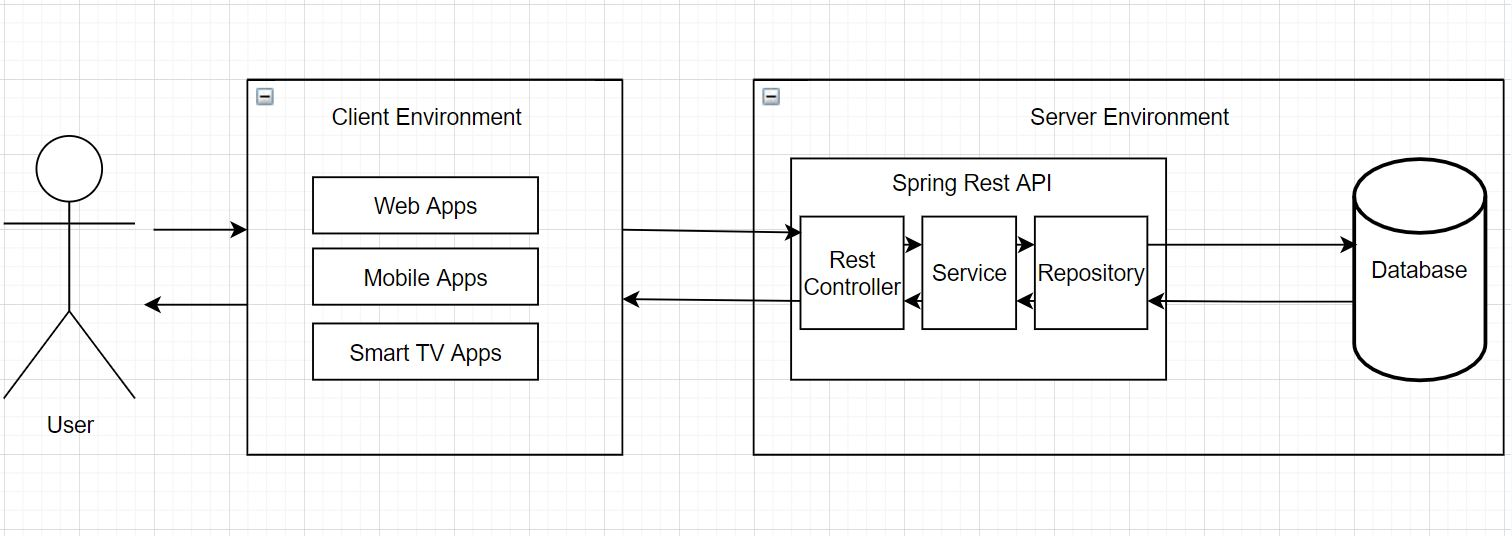
Ez a projekt egy kliens-szerver alkalmazás. Áll egy folyamatosan működő szerver alkalmazásból, egy adatbázisból, és egy kliens alkalmazásból. A szerver kommunikál az adatbázissal és műveletek a Controller osztályokban vannak definiálva.

Ezeken keresztül lehet nagyon szerte ágazóan utasításokat adni a szervernek valamilyen kliens által. Minden felhasználó rendelkezik személyre szabott objektumok fölött (pl. Contract), és vannak közös objektumok is, amelyeket mindenki lát. Ezek fölött nem rendelkeznek teljhatalmú uralommal. Vannak szerepkörök (role-ok) definiálva: admin, user. Bizonyos műveleteket csakis az admin szerepkörrel rendelkező felhasználó képes elvégezn. Ilyen például a POST /createadmin (új admin felhasználó létrehozása) vagy a POST /vehicle (új járművet lehet beszúrni a táblába). Az egyszerű felhasználók csak használni tudják ezeket az adatokat.

A kliens jelen esetben egy android alkalmazás, amelyben minden fontos funkció elérését implementáltam, így a szerver által nyújtott szolgáltatások nagy hányadát lefedi. A bejelentkezés biztonságos módon van megoldva, a jelszavak már lekódolva vannak elmentve egy Bearer token-ben és bejelentkezés után minden művelet-kor ennek a token-nek a szerverre való felküldése szükséges a műveletek hitelesítéséhez. A regisztrálás (/register), a bejelentkezés (/login) és a szerverrel való kapcsolat ellenőrzése (/welcome) céljából létrehozott rest kéréseken kívül mindig szükséges a fejlécben küldött bearer tokennek a használata. Igazából a kliens a /login kérés sikeres lefutása után kapja meg ezt a tokent visszatérítve szintén a headerben. A token minden sikeres /login esetében újra generálódik szerver szinten.

A felület felhasználóbarát módon van megalkotva, minden fontosabb eseménykor felugró ablak értesíti a felhasználót az esemény sikeres, avagy sikertelen megtörténtéről. A reszponzivitás görgetősávokkal és felülethez alkalmazkodó beimprtált elemekkel van megoldva, így egy kisebb képernyővel rendelkező telefononon, vagy nagyobb képernyővel rendelkező tableten is használhatók az alkalmazás funkciói.

A validációk több szinten is meg vannak valósítva különböző beépített és saját kezűleg megírt függvények segítségével. A kliens szinten vannak előleges validációk, ha ezen túljutunk, akkor a szerver is le ellenőrzi egy függvénnyel, kivételek dobásával vagy annotációkkal. Az adatbázis validációkat és a tranzakciókezelést a spring boot keretrendszer biztosítja.

A java szerver és az adatbázis között automatikus leképezés van Object-Relational Mapping (ORM) segítségével. Ennek az implementálásához hibernate-t használtam. A hibernate fő előnye az Objektum Orientált (OO) szemlélet, amely során a relációs adatbázis elérése lehetségessé válik JDBC vagy SQL kód írása nélkül.

A kliens-szerver (magyarul: ügyfél-kiszolgáló) kifejezést először az 1980-as években használták olyan számítógépekre (PC-kre) amelyek hálózatban működtek. A ma ismert modell a 80-as évek végén vált elfogadottá. Az ügyfél (kliens) valamilyen kéréssel fordul a kiszolgáló felé, amely a kért szolgáltatást nyújtja.

A programozásban a főprogram-alprogram viszonynak feleltethető meg. A kommunikációt mindig az ügyfél kezdeményezi, sohasem a kiszolgáló, vagyis a szerver szolgáltatást nyújt a munkaállomások (kliensek) részére. Egy erős hardverrel és jó konfigurációval rendelkező szerver nagyon sok kliens-t ki tud szolgálni egyszerre, amelyek akár többfélék lehetnek: webböngésző, telefon, okos tv, okos óra, hűtő stb.

A kliens-szerver architektúra legalapvetőbb formájában mindössze kétfajta csomópont (node) van, a kliens és a szerver. Ezt az egyszerű architektúrát két szintűnek (angolul two-tier) hívják.

Bonyolultabb architektúrák is léteznek, amelyek 3 különböző típusú csomópontból állnak: kliensből, alkalmazás szerverből (application server) valamint adatbázis szerverből (database server). A háromszintű kiépítésben az alkalmazásszerverek azok, amelyek kiszolgálják a kliensek kéréseit, és az adatbázisszerverek az alkalmazásszervereket szolgálják ki adatokkal. Ennek a rendszernek nagy előnye a bővíthetőség. A rendszer így képes lesz egyensúlyozni és elosztani a feldolgozásra váró adatmennyiséget és munkát a több és gyakran redundáns, specializált csomópont között. Hátránya, hogy nagyobb az adatátviteli forgalom a hálózaton és hogy nehezebben programozható, illetve tesztelhető egy kétszintű architektúránál, mert több eszközt kell összehangolni a kliensek kéréseinek kiszolgálásához.

1. **Szerver alkalmazás**

Jellemzői:

* Passzív, a kliensektől várja a kéréseket
* A kéréseket, lekérdezéseket feldolgozza, majd visszaküldi a választ
* Általában nagyszámú klienshez kapcsolódik egyszerre
* Általában nem áll közvetlen kapcsolatban a felhasználóval
  1. **A rendszer felépítése**

A diploma

### A domainek

A

### A szolgáltatások (Services)

A

### A vezérlők (Controllers)

A

### A biztonság

A

### A konfigurációk (Configurations)

A

### A validációk

A

### A tesztelés

A

### A logolás

as

1. **Kliens alkalmazás**

E

* 1. **Az applikáció felépítése**

A diploma

### A kapcsolatfelvétel

A

### A belépés

A

### A regisztráció

A

### A módosítás

A

### A járművek listája

A

### Egy jármű tulajdonságai

A

### A szerződéskötés

A

### A kijelentkezés

A

### A validációk

A

### A logolás

A

1. **Irodalomjegyzék**

|  |  |
| --- | --- |
| [1] | Git branches, [Online]. Elérhető: https://www.atlassian.com/git/tutorials/using-branches. [Hozzáférés dátuma: 19 október 2022]. |
| [2] | JSon API [Online]. Elérhető: http://jsonapi.org/. [Hozzáférés dátuma: 20 október 2022]. |
| [3] | Visual Studio Code [Online]. Elérhető: http://hu.wikipedia.org/wiki/Visual Studio Code. [Hozzáférés dátuma: 30 október 2022]. |
| [4] | Java [Online]. Elérhető: https://www.java.com/en/download/help/whatis\_java.html. [Hozzáférés dátuma: 31 október 2022]. |
| [5] | Gradle [Online]. Elérhető: https://docs.gradle.org/, [Hozzáférés dátuma: 05 november 2022] |
| [6] | Spring [Online]. Elérhető: https://spring.io/, [Hozzáférés dátuma: 05 november 2022] |
| [7] | Spring boot [Online]. Elérhető: https://spring.io/projects/spring-boot, [Hozzáférés dátuma: 05 november 2022] |
| [8] | Android studio [Online]. Elérhető: https://en.wikipedia.org/wiki/Android\_Studio, [Hozzáférés dátuma: 12 november 2022] |
| [9] | Sdk [Online]. Elérhető: https://www.redhat.com/en/topics/cloud-native-apps/what-is-SDK, [Hozzáférés dátuma: 13 november 2022] |
| [10] | Sdk manager [Online]. Elérhető: https://developer.android.com/studio/command-line/sdkmanager, [Hozzáférés dátuma: 13 november 2022] |
| [11] | Emulator [Online]. Elérhető: https://en.wikipedia.org/wiki/Emulator, [Hozzáférés dátuma: 13 november 2022] |
| [12] | Github [Online]. Elérhető: https://github.com/about, [Hozzáférés dátuma: 13 november 2022] |
| [13] | OkHttp [Online]. Elérhető: https://square.github.io/okhttp/, [Hozzáférés dátuma: 14 november 2022] |
| [14] | Postman [Online]. Elérhető: https://en.wikipedia.org/wiki/Postman\_(software), [Hozzáférés dátuma: 15 november 2022] |
| [15] | Bearer token [Online]. Elérhető: https://swagger.io/docs/specification/authentication/bearer-authentication/, [Hozzáférés dátuma: 25 november 2022] |
| [16] | Hibernate, ORM, JPA [Online]. Elérhető: https://www.javatpoint.com/hibernate-tutorial, [Hozzáférés dátuma: 26 november 2022] |
| [17] | Android [Online]. Elérhető: https://www.javatpoint.com/android-software-stack, [Hozzáférés dátuma: 26 november 2022] |
| [18] | Alkalmazásfejlesztési környezetek (VIAUAC04), 2022 tavaszi előadások, Tárgy holnapja: https://portal.vik.bme.hu/kepzes/targyak/VIAUAC04 |

**Függelék**

**Build-elés** – az a folyamat, amely során a számítógép a forráskódot valamilyen futtatható állományba tömöríti össze, amelyet később fel tudunk használni akármikor, vagy amikor direkt feltelepíti a programot egy céleszközre.

**Git Branch** – hatékonyan mutatják a változtatások pillanatképét. A fájlok könyvtárból könyvtárba másolása helyett a Git egy branchet tárol a véglegesítésre való hivatkozásként. Ebben az értelemben egy branch a véglegesítések sorozatának csúcsát jelenti – ez nem a véglegesítések tárolója.

**Git Merge** – A mergelés a Git módszere arra, hogy újra összerakja az elágazó brancheket. A git merge parancs lehetővé teszi a git branch által létrehozott független fejlesztési vonalak felvételét és egyetlen ágba való integrálását.

**Fetch** - A git fetch parancs egy távoli tárolóból tölti le a véglegesítéseket, fájlokat és hivatkozásokat a helyi tárhelyre. A lekérés az, amit akkor csinál, ha látni szeretné, min dolgozott már mindenki. Ez teszi a lekérést biztonságossá, mielőtt integrálná a véglegesítéseket a helyi tárolóba.

**Pull** - A git pull parancs a tartalom távoli tárolóból való lekérésére és letöltésére szolgál, valamint a helyi lerakat azonnali frissítésére, hogy megfeleljen a tartalomnak.

**Commit** – A git commit parancs pillanatképet készít a projekt jelenleg végrehajtott módosításairól.

**Push** - A git push parancsot a helyi lerakat tartalmának távoli tárolóba való feltöltésére használják. A leküldés az a mód, ahogyan átviheti a commitokat a helyi lerakatból egy távoli tárolóba.

**Debug folyamat** – hibakeresési folyamat, ami lehet statikus (fejlesztői eszköz általában elvégzi), és dinamikus, amelynek során a program lefuttatásával ellenőrizzük, hogy az megfelelően működik-e. Ez általában magával vonja annak ellenőrzését, hogy megadott bemeneti értékek esetén a program az elvárt kimenetet produkálja-e. Sok esetben megállási pontokat helyezünk el a kód azon részén, ahol szerintünk végig fog haladni, aminek hatására ott kicsit szünetel a futás és meg tudjuk nézni a pillanatnyi változók értékét.

**SDK** – Szoftverfejlesztői csomag (Software development kit). Szoftverfejlesztő eszközök gyűjteménye egyetlen telepíthető csomagban.

**IDE** - Integrált fejlesztői környezet (Integrated development environment). Az integrált fejlesztői környezet egy olyan szoftveralkalmazás, amely átfogó létesítményeket biztosít a számítógépes programozók számára a szoftverfejlesztéshez. Az IDE általában legalább egy forráskódszerkesztőből, automatizálási eszközökből és hibakeresőből áll. Pl: Eclipse, NetBeans, Visual Studio, Android Studio, IntelliJ IDEA...

**CMD** - A Command Prompt, más néven cmd.exe vagy cmd, az OS/2, eComStation, ArcaOS, Microsoft Windows és ReactOS operációs rendszerek alapértelmezett parancssori értelmezője, amely 1987-ben jelent meg először. A cmd.exe parancssori felületen keresztül kommunikál a felhasználóval. Megvalósítása operációs rendszerenként eltérő, de a viselkedés és a parancsok alapkészlete konzisztens.

**Repository (REPO)** - A szoftvertár vagy angolul repository a szoftvercsomagok tárolási helye. Gyakran egy tartalomjegyzéket is tárolnak a metaadatokkal együtt. A szoftvertárakat általában forrásvezérlők vagy lerakatkezelők kezelik. A csomagkezelők lehetővé teszik a tárolók automatikus telepítését és frissítését.

**HTTP (HyperText Transfer Protocol)** - egy információátviteli protokoll elosztott, kollaboratív, hipermédiás, információs rendszerekhez. A HTTP fejlesztését a World Wide Web Consortium és az Internet Engineering Task Force koordinálta RFC-k formájában.

**RFC** – ez egy olyan dokumentum, mely egy új Internet-szabvány beiktatásakor adnak közre. Az új szabvány első tervezete saját számmal kerül a nyilvánosság elé, egy adott időtartamon belül bárki hozzászólhat.

**Connection pool** - A szoftverfejlesztésben a kapcsolatkészlet az adatbázis-kapcsolatok gyorsítótárát jelenti, amelyet úgy karbantartanak, hogy a kapcsolatok újra felhasználhatók legyenek, amikor az adatbázishoz jövőbeli kérésekre van szükség. A kapcsolati készletek a parancsok végrehajtásának teljesítményét javítják az adatbázison.

**Cache (gyorsítótár)** - a számítástechnikában az átmeneti információtároló elemeket jelenti, melyek célja az információ-hozzáférés gyorsítása. A gyorsítás egyszerűen azon alapul, hogy a gyorsítótár gyorsabb tárolóelem, mint a hozzá kapcsolt, gyorsítandó működésű elemek.

**Node (csomópont)** – Ez a hálózatban nem más, mint egy kapcsolódási pont. Lehet egy újraelosztási pont (adattovábbítás) vagy pedig az adatátvitel végpontja. Általában a csomópontok úgy vannak felállítva, hogy képesek felismerni és továbbítani az adatokat más csomópontok felé is.

**Biztonsági token** – Lehet egy fizikai eszköz, amit egy számítógépes szolgáltatás autorizált felhasználója kap, az autentikáció segítése érdekében, de a mi esetünkben egy generált szöveg. Egy felhasználó hozzáférési tokenje tartalmazhatja a felhasználó nevét, azokat a csoportokat, amelyekhez a felhasználó tartozik, a felhasználó biztonsági azonosítóját és az összes olyan csoport biztonsági azonosítóját, amelyekhez a felhasználó tartozik.

**Responzív design** - A reszponzív design egy olyan kialakítása felhasználói felületnek, amely rugalmasan alkalmazkodik a különböző eszközökhöz, böngészőkhöz, képernyőméretekhez annak érdekében, hogy optimális megjelenést biztosítson a felhasználónak minden alkalmas eszközön. Létrejöttének oka az egyre többféle méretű kijelző megjelenése.