ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

**NEUMANN JÁNOS
INFORMATIKAI KAR**

5. sz. melléklet

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Főző Árpád**

2021 Hallgató törzskönyvi száma: **T/005096/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Főző Árpád**
Törzskönyvi száma: T/005096/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Bűnügyi Térkép Mobil Alkalmazás
Crime Map Mobile Application

Intézményi konzulens: Rigó Ernő
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Felhőszolgáltatási technológiák és IT biztonság specializáció

A feladat

A világ egy veszélyes hely. Sok bűntényt követnek el az ország számos pontján.
Kijelenthető, hogy senki nem szeretne olyan helyre költözni vagy az értékéit, mint biciklit vagy autót olyan helyen hagyni, ahol kirívóan sok ilyen eset történik.
De mégis, hol lehet erről információt találni?
A rendőrség már választ adott erre a problémára a bűnügyi térképükkel.
Ezen a térképen, látható, melyik utcában mennyi bűncselekmény lett elkövetve.
A probléma ezen térképpel, hogy nem túl hordozható, telefonról nehezen használható.
A rendkívüli szerencse viszont, hogy a rendőrség bizonyos mértékig, bármely fejlesztő részére hozzáférhetővé tette, a térkép mögötti adatbázist.
A dolgozat célja, egy telefonos térkép alkalmazást elkészíteni, amely, felhasználva ezen adatbázist, képes megjeleníteni egy környéken elkövetett bűncselekmények számát, ezzel segítve:
- lakás vásárlás vagy albérlet választás esetén környék leellenőrzését,
- utazástervezés során biztonságos útvonal választását,
- bicikli vagy autó biztonságos helyen hagyását.

A dolgozatnak tartalmaznia kell:

- a létező vagy hasonló megoldások feltérképezését és összehasonlítását,
- a rendszerrel kapcsolatos elvárások specifikációját,
- implementációs tervet,
- az applikáció use-case és wireframe diagrammjait,
- a tesztelési szempontokat, a tesztelés folyamatának leírását és az eredmények kiértékelését,
- az applikáció implementációját, és ennek dokumentációját.

Ph.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 20..22..05..12.....

.....
hallgató aláírása

ÓBUDAI EGYETEM

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Főző Árpád
Neptun Kód: [redacted]
Tagozat: Felhő szakirány

Telefon: [redacted]
Levelezési cím:

Szakdolgozat címe magyarul:
Bűnügyi Térkép Mobil Alkalmazás

Szakdolgozat címe angolul:
Crime Map Mobile Application

Intézményi konzulens: Rigó Ernő
Külső konzulens:

Kérjük,hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.10.05	Alapötlet megbeszélése	
2.	2021.10.29	Irodalomkutatás áttekintése	
3.	2021.11.18	Tartalom áttekintése	
4.	2021.12.06	Prezentáció próbabemutatása	

A Konzultációs naplót összesen 4 alkalommal az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a „Szakdolgozat" tantárgy aláírási követelményét teljesítette.

Intézményi konzulens

Budapest, 2021.12.10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Fõzõ Árpád..... [redacted] ..... Felhõ szakirány.....

Telefon: Levelezési cím (pl: lakcím):

[redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

Bünügyi Térkép Mobil Alkalmazás.....

Szakdolgozat / Diplomamunka[2] címe angolul:

Crime Map Mobile Application.....

Intézményi konzulens: Külső konzulens:

Rigó Ernő..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.10	Szerver program architektúra egyeztetése	
2.	2022.04.05	Szakdolgozat leellenőrzése	
3.	2022.04.13	Telefon program architektúra egyeztetése	
4.	2022.04.29	Tartalom áttekintés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.12.

.....
Intézményi konzulens

1 Megfelelő aláhúzandó!
2 Megfelelő aláhúzandó!
3 Megfelelő aláhúzandó!
4 Megfelelő aláhúzandó!

Tartalom

1. Bevezetés

1.1. Probléma

A rendőrség munkájától függetlenül, bármelyik településen lehetnek helyek ahol az ember, mindenféle előítélet nélkül is elbizonytalanodhat, a terület biztonságosságát illetően. Nem tudja, hogy leteheti-e itt a biciklijét, jó ötlet-e vásárolni vagy bérelni a környéken egy lakást, vagy akár megéri-e végigmenni az adott utcán, esetleg válasszon egy másik útvonalat.

Vegyünk egy fiatalt, aki szeretne egy lakást bérelni. Végig nézi a feladott hirdetéseket, meg is talál egy az átlagnál olcsóbb rezidenciát. Felmerülhet a kérdés, mi az árkülönbség oka. Nem nagyon tud másra hagyatkozni, mint a főbérlőre, esetleg a saját benyomásaira. A probléma ott van, hogy egyik sem tükrözi feltétlenül a valóságot. A főbérlőnek az ingatlan kiadása a legfőbb célja, így lehetséges, hogy a környék negatív aspektusait nem osztja meg. Illetve az is lehetséges, hogy miközben nappal semmi kirívó problémával nem szembesül, aközben este, kis túlzással élve, ki sem mer lépni. Ez a tapasztalat nem biztos, hogy megéri a kauciót és a szerződésben megkötött időkeretet, amíg kötelezően bérelnie kell a lakást vagy házat.

1.2. Rendőrség megoldása

Erre megoldást nyújt a rendőrség bűnügyi térképe, amin megtekinthető, a Magyarországon elkövetett összes lezárt bűncselekmény, amennyiben 30 napnál régebben történt, de 60 napnál nem, térképen jelölve. Amennyiben otthoni számítógépen vagy laptopon használják a szolgáltatást, kielégítő lehet, viszont telefonról megnyitva, hagy kívánnivalót maga után a rendszer. Mobil eszközön csak böngésző használatával elérhető, érintőképernyős használata jelentős késleltetéssel működik emellett a betöltési ideje ingadozó. A hallgató egyazon eszközön tapasztalt viszonylag gyors (kb. 5mp-es) és rendkívül lassú (kb. 1-2 perces) felállást is.

1.3. Saját megoldás

A rendszer tökéletes a második bekezdésben említett problémára, csupán egy visszásságát kell kiküszöbölni. Hordozhatóvá kell tenni. A mobilitás elérése érdekében biztosan egy telefon operációs rendszere lesz az alkalmazás környezete. Az, hogy melyik, egy későbbi bekezdésben kerül tárgyalásra, a programozói környezettel egyetemben.

A telefonra fejlesztett térkép alkalmazások segítségével lehetőség nyílt egyszerűbb tájékozódásra, úttervezésre, nyaralás szervezésére, könnyebben található, a felhasználó számára szimpatikus étterem, ezek mellett nem lehet megfeledkezni arról sem, hogy a közösségi közlekedési eszköz bérlése is ilyen applikációk segítségével bonyolíthatók le. Röviden akármelyik városban vagy településen egyszerűbbé teszik az életet. Miért-ne tehetnék biztonságosabbá is az ott tartózkodást?

A hallgatónak, a szakdolgozat keretein belül, egy olyan alkalmazás elkészítése lesz a feladata, amely a rendőrség bűnügyi térképének telefonos megfelelőjeként képes funkcionálni. A terv, hogy egy interaktív térképen, egy választott pont bizonyos sugarú körében, utcánként képes legyen megjeleníteni az elkövetett bűncselekményeket, megnevezéssel és dátummal. Ehhez a szakdolgozat író, annak az adatbázisnak a publikus részét fogja felhasználni, amely, a bűnügyi térkép által megjelenített adatokat tartalmazza. A rendelkezésre álló erőforrások miatt, a cél egy, maximum két eszköz egyidejű kiszolgálása.

1.4. Eddigi megvalósítások

A manapság leghasználtabb térképes applikációknak, érhető módon nincs részletekbe menően közzétéve az architektúrájuk. A hallgatónak viszont így is sikerült egy alap képet kapnia a rendszerek felépítéséről.

A front-end-től elindulva az első rész, a kezelő felület. Itt kap helyet a térkép komponens, amelyen megjelennek a felhasználó számára szánt adatok. A térkép maga lehet saját fejlesztésű, mint a Waze, navigációs alkalmazásban használt, vagy egy más cég által kreált, mint ami az Uber alkamazásban található. Utóbbinál Google Maps térképét integrálták.[1] Erre lehetőséget biztosít a cég a Google Maps API által. Ennek próbaverziója ingyenes, viszont éles környezetben, minden egyes az API kulcs általi betöltés után, minimum 0.002 amerikai dollárt számolnak fel.[2]

Az alkalmazások adatai felhőben vannak tárolva, és feldolgozva. A felhasználó helyadatai Rest API hívások segítségével jutnak el a felhő rendszerig. A nagyszámú kéréseknek köszönhetően ahhoz, hogy az adatok egységesen kerüljenek feldolgozásra és tárolásra a felhő rendszer erőforrásai által, Load Balancert használnak. Ezzel képesek elkerülni azt, hogy a rendszer egy pontja kezeli a kérések többségét, ezzel túlterhelve azt, miközben a többi kapacitása kihasználatlan marad.[3]

Az adatok tárolására több megoldás is létezik. A fent említett alkalmazások esetén fontos szempont a bővíthetőség és a kezelhetőség. Az Uber kezdetben relációs adatbázist használt, amelyet ki kellett egészíteni nem relációs elemekkel, a nagy mennyiségű adat kezelésére. Kikötéseik, hogy több szerver hozzáadásával bővíthető legyen a kapacitás. Gyorsan lehessen írni és olvasni az adatbázisba és –ból. Emellett folyamatosan elérhető legyen.[4] Ezek a tulajdonságok tökéletesen jellemeznek egy felhőrendszert, így nem meglepő, hogy ezt a tárolási helyet választották. A két adatbázis nyelv használatával pedig vegyítették a relációs strukturáltságát és a nem relációs bővíthetőségét.

Mind az Uber[4] mind a Waze[5] az adatok feldolgozását mikroszervizekkel végzi. Ezek egy nagyobb feladatot, több kisebb független folyamatra bontanak fel, amelyet különálló szolgáltatások végeznek. Ezek a szolgáltatások többnyire API hívásokkal kommunikálnak egymással. Előnye, hogy az adatok feldolgozása párhuzamosan képes folyni, hiszen egyszerre több szolgáltatás is dolgozhat, a monolitikus kód egymásutániságával szemben, így gyorsítva azt.

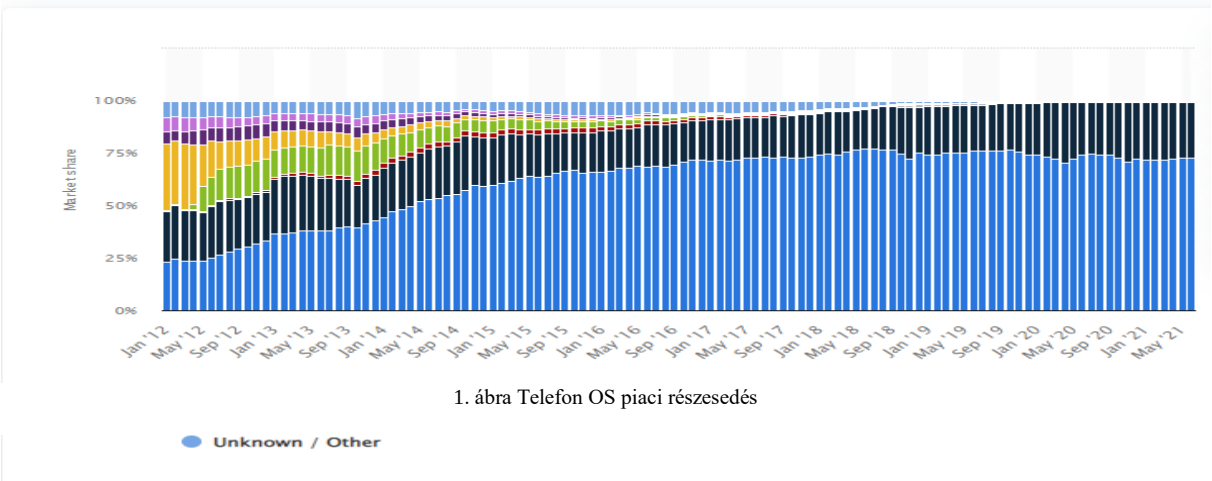
Összefoglalva, a nagy térképes applikációk fő szempontjai, adattal kapcsolatban, a gyorsaság, legyen szó adatbázisba való írásról, olvasásról, vagy a kapott adat feldolgozásáról. A rendszerük kapacitásának bővíthetősége, és maximális kihasználtságának biztosítása, az előző mondatban lévő elvárásoknak való megfeleléshez. Illetve a folyamatos elérhetőség.

1.5. Kialakítás

A forrás a rendőrség adatbázisa. Itt találhatók táblánként az adatok, amelyeket megjelenítenek a Bűnügyi térképen. Használata legális az oldalukon található leírás alapján, ugyanakkor egyeztetés szükséges az üzemeltetőkkel és a rendőrséggel, nehogy bármilyen adat jogszerűtlenül legyen felhasználva és közzétéve. A rendőrség által használt adatbázishoz, REST API-n keresztül lehet hozzáférni. Erre a célra az OData (Open Data Protocol) protokollt használják. Ez később kerül majd tárgyalásra, jelen bekezdésben annyit érdemes megemlíteni, hogy egy egyszerű, autentikációt nem igénylő, HTTP GET kéréssel, hozzá lehet férni az információhoz.

Amennyiben a hallgató adatbázis szerver implementálása mellett dönt, az adatbázis az innen lekért információval lesz feltöltve. Szükséges lesz egy keret programot készíteni, amely segítségével le lehet kérdezni és módosítani az adatbázis tartalmát. A rendőrség naponta frissíti a rendszerüket, a délutáni-esti órákban, így a saját adatbázist is elég lenne naponta egyszer. Minden itt tárolandó információ publikus, emellett szintén csak Http GET kérésre válaszolna, az-az nem kell aggódni az adatbázis jogosulatlan módosítása vagy akár használata miatt, így a biztonság nem lenne fő szempont a rendszer megalkotásánál.

Az alkalmazás alapja egy interaktív telefonos térkép. Egy Magyarországi pont kiválasztásokkor lekéri az adatokat a forrásról, majd ezeket megjeleníti. Ahogy az az első ábrán látható, a leginkább elterjedt két mobil operációs rendszer, az Android és IOS. Kérdés melyikre történjen a fejlesztés. Miután sikerült döntést hozni, ki kell választani mely fejlesztői környezetet érdemes használni az alkalmazás elkészítéséhez.[6]



1. ábra Telefon OS piaci részesedés

2. Forrás Police.hu Bűnügyi Térkép

2.2.1 Kialakulás

Mielőtt elkezdődne az elkészítendő rendszer tárgyalása, érdemes áttekinteni az alap honlapot, amelynek a telefonos kiegészítését kívánja elkészíteni a hallgató.

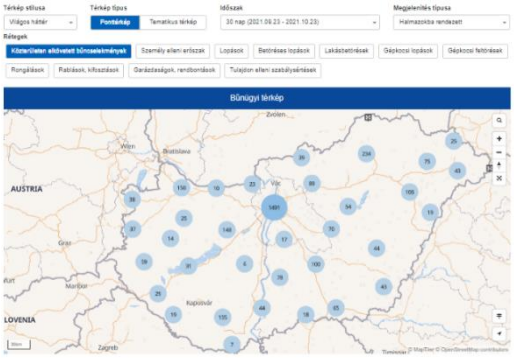
A térkép története 2002-ben kezdődött. Az Origo cikke alapján [7], ez volt az addigi legrészletesebb bűnügyi helyzetfelmérés. Az ehhez való hozzáférés 2002-ben még a Belügyminisztériumon keresztül volt lehetséges, interneten nem osztották meg a tartalmat. Nem meglepő, hiszen hazánkban, a mai adatokhoz képest csak elterjedőben volt az internet használat. Megint csak az Origo-ra hivatkozva: „A magyarországi internethasználók számát mérő kutatások igen nagy szórással dolgoznak: a szakmai vélemények, piackutatások szerint hazánkban az otthonról, munkahelyről és iskolából netezők száma összességében 1,3-1,7 millió körül van.[8]

2. ábra Bűnügyi térkép 2012

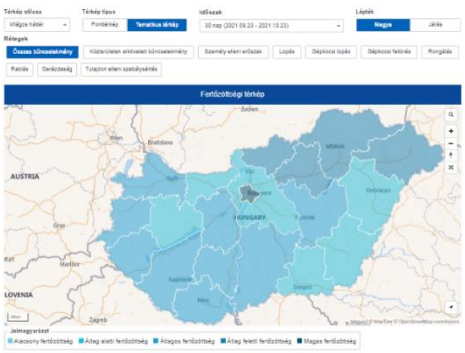
Ezek után az első nagy ugrást 2012 hozta. Ekkor tették közzé az első változatát a rendőrség Bűnügyi térképének, amely funkcionalitását tekintve szinte teljes mértékben megegyezik a maival. Látható a dátuma, a helyszíne és piktogramja az elkövetett bűnténynek és típusának. Különbség, hogy itt még minden bűntény egy térképen volt megjelenítve. Ez látható a második ábrán.[9]

2.2.2 Jelenlegi forma

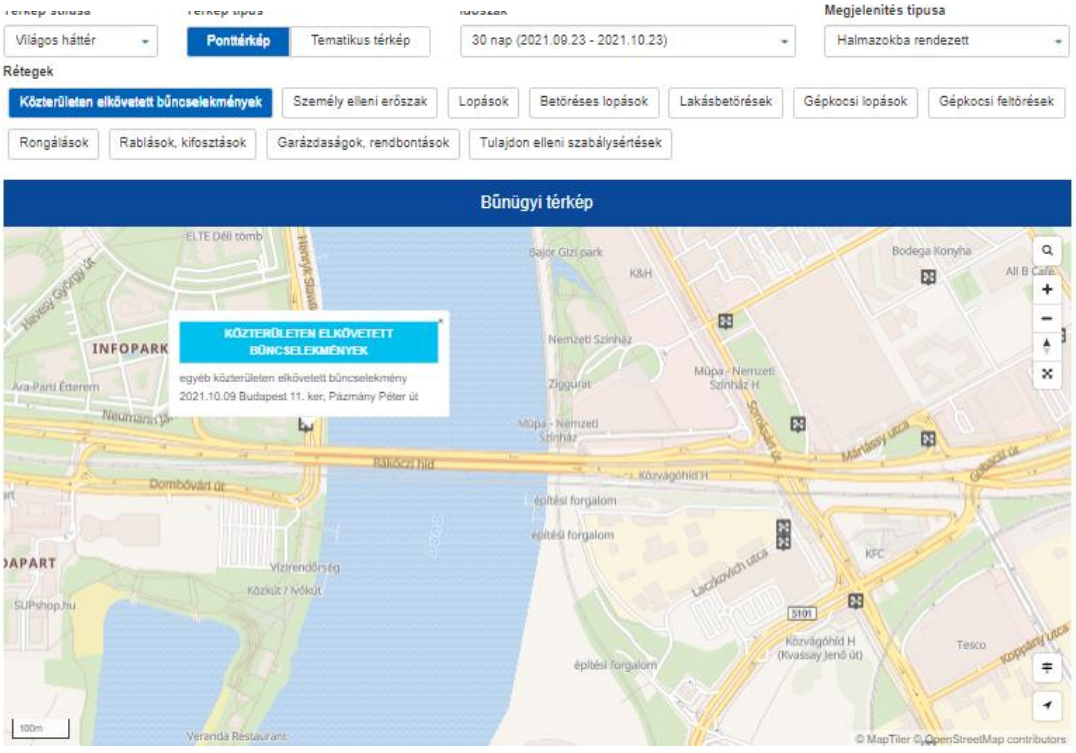
A ma elérhető változatnak két típusa van. Az egyik a klasszikus ponttérkép, amely pontokba szedve jeleníti meg az elkövetett bűncselekményeket, lásd harmadik és ötödik ábra. A másik a tematikus térkép, ahol egy-egy megye vagy járás „fertőzöttsége” tekinthető meg. A fertőzöttség 100.000 lakosra jutó bűncselekmények számát jeleníti meg. Minél sötétebb kékkel van beszínezve a terület, annál több bűneset került elkövetésre, lásd 3. ábra. A ponttérképen csak szűrt adatok jelennek meg, nincs „Összes bűncselekmény” menüpont, ahogy a tematikus térképen.



3. ábra Bűnügyi térkép pont térkép



4. ábra Bűnügyi térkép tematikus térkép



5. ábra Bűnügyi térkép bűncslekmény megjelenítése

A szakdolgozatban megvalósítani kívánt program számára ez nem lesz megfelelő, egy összesített térkép szükséges, hiszen minek elvárni a felhasználótól, hogy három menüponton keresztül jusson hozzá a keresett információhoz (pl. visszatérve a lakásbérléshez, a felhasználónak szüksége lehet a környéken elkövetett betörések, lopások és személy elleni erőszak számára.), ha azt rögtön meg tudja tekinteni egy menüpont alatt.

A hallgatónak a ponttérkép adataira lesz szüksége a feladatához. A tematikus térkép, a régiók bűnügyi statisztikáinak illusztrálására tökéletes, viszont a bevezetésben tárgyalt célhoz, az-az az utcákban elkövetett bűncselekmények megjelenítéséhez nem megfelelő.

2.2. Adatokhoz való hozzáférés

A https://terkep.police.hu/portal/bunugyi oldal tökéletes látványtervnek, hiszen ehhez hasonló telefonos alkalmazás elkészítése a cél. Legfőképpen viszont az adataira van szükség. Az adatokhoz, amelyeket a térkép megjelenít, elérési utat és egy bizonyos mértékű dokumentációt biztosít a rendőrség a „Fejlesztőknek” fül alatt.

2.2.1 RESTful API

A leírásból kiderül, hogy adatbázisukhoz Odata segítségével létrehozott RESTful web szolgáltatásokon keresztül lehet hozzáférni, autentikáció nélkül.

A REST egy nem nyelvspecifikus tervezési minta, amely interfészt biztosít a kliensnek, a szerver erőforrásaihoz, HTTP vagy TCP/IP protokollokon keresztül. A REST dizájn megalkotója Roy Fielding a következő alapelveket határozta meg: [10]

Az API (Application Programming Interface) négy alapműveletet biztosít a szerverrel való kommunikációhoz.

- **Létrehozás** – POST: Erőforrás létrehozása a szerveren.
- **Lekérés** – GET: Szerveren található erőforrás lekérése.
- **Frissítés** – PUT: Szerveren található erőforrás információjának megváltoztatása.
- **Törlés** – DELETE: Szerveren található erőforrás törlése.

Fontos kikötés, hogy egyes műveletek valóban csak azt végezzék el, amely a specifikációban olvasható. Például GET hatására ne módosuljanak a szerveren tárolt erőforrások.

Legyen állapotmentes a web szolgáltatás. A HTTP kérés törzsében kap meg minden információt a szerver, kontextus vagy előző állapot ismerete nélkül. Például, a web szolgáltatás következő oldalának információjára érkezik GET kérés. Ha ezt szerver kezeli olyan módon, hogy lekérdezi a jelenlegi oldal számát és hozzáad egyet, majd ez alapján küldi a választ, az nem jó megoldás. A kliensnek kell küldenie pontosan melyik oldalra kíváncsi.

Könyvtár szerkezetű URI-k (Unified Resource Identifier) implementálása. Cél, hogy az URI logikus felépítésével könnyen átlátható legyen a felhasználó számára. Például egy cukrászda web szolgáltatásának URI-ja, amellyel az egyes típusú (vegán, laktózmentes) tortáik adataihoz (ár, kalória) biztosítanak hozzáférést, következőképpen nézhetne ki:

www.cukibolt.hu/termekek/tortak/típus/

Egyértelmű, hogy a webszolgáltatás milyen információt szolgáltat.

A kliens és szerver közötti adattranszfer JSON vagy XML (esetleg egyszerre mindkettő) dokumentum formátumban történjen. Ezzel biztosítva van, hogy a HTTP kérés vagy válasz törzsében egyszerű, könnyen kezelhető, ember által is értelmezhető adat van.

2.2.2 OData

Az ODATA (Open Data Protocol) egy standard, amelynek célja, hogy leegyszerűsítse a lekérdezéseket és adatmegosztást különböző fajta platformok és applikációk között. Ennek az eszköznek az implementálásával viszonylag egyszerűen hozható létre REST API az elérni kívánt erőforráshoz, legyen az relációs adatbázis, fájlrendszer, vagy honlap. A honlap olvasható leírás alapján, ODATA használatával, a fejlesztőnek nem kell törődnie a kérés és válasz header-ökkel, URL konvencióval, HTTP metódusokkal és a legtöbb REST API készítésekkor felmerülő kérdéssel, így fókuszálhat a biznisz logikára.[11] Segítségével az alap CRUD műveleteken kívül, képes komplexebb lekérdezéseket natívan kezelni, emellett kérés alapján metódusokat lefuttatni a szerver gépen és URI-t generálni az erőforrás modelljéből.

2.2.3 Rendőrség API használat

A fejlesztőknek fül alatt kilenc darab ODATA segítségével létrehozott API látható. A felsorolásban szereplő szolgáltatások a következő típusú URI segítségével érhetők el:

https://terkep.police.hu/api/odata/v1/szolgáltatás

Amennyiben az adatok csak id-val kérhetők le (pl. 60 napos lopási adatok):

https://terkep.police.hu/api/odata/v1/szolgáltatás(id)

Szolgáltatások név szerint:

- **MapCompositions:** A szolgáltatáson keresztül egy vagy több publikált kompozíció legfontosabb adataihoz juthatunk hozzá. (https://terkep.police.hu/portal/open-data-api) Az itt feltüntetett adatok a „Térkép kereső” fül alatt található térképek információit teszi hozzáférhetővé.
- **MapComposition:** Ugyancsak az mondható el, mint a fentebb említett szolgáltatásnál, ugyanakkor itt több adatot lehet lekérdezni.
- **MapLayer:** A tematikus típusú térkép megjelenítéséhez szükséges táblák nevei és típusai vannak eltárolva.

- **ChartConfig:** A szolgáltatáson keresztül egy grafikon részletezett adataihoz innen kérdezhetők le. Szintén „Térkép kereső” fül alatt található térképek információit teszi hozzáférhetővé.
- **TableConfig:** A szolgáltatáson keresztül egy táblázat részletezett adatai innen kérdezhetők le. . Szintén „Térkép kereső” fül alatt található térképek információit teszi hozzáférhetővé.
- **Infections:** Az „Infection” táblák lekérdezéséhez szükséges id-k innen kérdezhetők le. Tematikus típusú térképhez.
- **Infection:** A tematikus térképen megjelenítendő adatok, innen kérdezhetők le. Tematikus típusú térképhez.
- **Accidents:** Az „Accident” táblák lekérdezéséhez szükséges id-k innen kérdezhetők le. Baleseti térképhez.
- **Crimes:** Az „Crime” táblák lekérdezéséhez szükséges id-k innen kérdezhetők le. Ponttérképhez.

A kérdés, hogy a telefonos alkalmazás számára szükséges adat honnan kérdezhető le. Sajnos a honlapon (a forráskódon kívül) erről nincs megosztva információ, a felsorolásban jelzett felhasználási hely a hallgató saját kutatásának az eredménye. A kutatás folyamata, a következő volt. Az egyes szolgáltatásoknál jelzett adatok összevetésre kerültek az oldalon megosztott összes térképpel. A térkép tárban lévő térképeknél, két adat volt a legfontosabb, a publikáció dátuma és az utolsó frissítés. Ezek alapján azonosítani lehetett, melyik sor melyik térképre értendő. Az Infection – magyarul fertőzöttség szóval a Tematikus térképnél lehet találkozni, az Accident – magyarul baleset értelemszerűen a Baleseti térképnél használatos, így kizárásos alapon maradt a Crime – Ponttérkép párosítás.

Ahogy már fent tárgyalásra került, nem a tematikus térkép adatai szükségesek, így a „MapLayer”, az „Infections” és „Infection” kizárásra került. Nem is a „Térkép tárban” található térképek adatai érdekesek a hallgató számára, ennek köszönhetően a „TableConfig”, „ChartConfig”, „MapComposition” és „MapCompositions” is figyelmeny kívül hagyható. „Accidents” és „Accident” amelyekben a balesetek adatai hozzáférhetők, is eliminálhatók, hiszen ez nem tartozik a jelen feladat hatáskörébe. Maradt a „Crimes” és „Crime” amire szükség van, hiszen itt vannak eltárolva a megjelenítendő bűncselekmények utcával és dátummal együtt.

Az itt leírt gondolatmenetet tesztelendő, a hallgató elkészített egy C# nyelven írt próbaprogramot, Visual Studio 2017-ben, amelynek célja a megjelenítendő adat lekérése. A hívások nem igényelnek semmiféle azonosítót, és a válasz mindig JSON formátumban érkezik. Az eredmény a következő.

A Crimes tábla egy HTTP GET hívással a hatodik ábrán látható adatot adja vissza. Egy megfelelő adatstruktúrával és konverterrel, ebből könnyűszerrel kinyerhető az összes szükséges ID. Fontos, az azonosítóban található szám mindig az időintervallumot jelzi.

Felhasznált URI:

https://terkep.police.hu/api/odata/v1/Crimes


```
{"@odata.context":"$metadata#Crimes","value":[{"name":"CRIME_60_BURGLARY"},
{"name":"CRIME_30_PROPERTY"},{"name":"CRIME_60_STEAL"},{"name":"CRIME_60_PROPERTY"},
{"name":"CRIME_180_BURGLARY_THEFT"},{"name":"CRIME_180_PEDESTRIAN_SERIOUS"},
{"name":"CRIME_180_BICYCLE_SERIOUS"},{"name":"CRIME_365_CAR_DEADLY"},
{"name":"CRIME_180_MOTORCYCLE_SERIOUS"},{"name":"CRIME_180_CAR_SERIOUS"},
{"name":"CRIME_180_LORRY_SERIOUS"},{"name":"CRIME_180_BUS_SERIOUS"},{"name":"CRIME_365_ABUSE"},
{"name":"CRIME_30_ABUSE"},{"name":"CRIME_60_ABUSE"},{"name":"CRIME_60_CAR_BREAK"},
{"name":"CRIME_365_BICYCLE_SERIOUS"},{"name":"CRIME_365_CAR_SERIOUS"},
{"name":"CRIME_365_PEDESTRIAN_SERIOUS"},{"name":"CRIME_60_PUBLIC_SPACE"},
{"name":"CRIME_60_ALL_CRIME"},{"name":"CRIME_60_TRUCULENCE"},
{"name":"CRIME_365_MOTORCYCLE_SERIOUS"},{"name":"CRIME_365_LORRY_SERIOUS"},
{"name":"CRIME_365_BUS_SERIOUS"},{"name":"CRIME_30_PUBLIC_SPACE"},{"name":"CRIME_180_ALL_CRIME"},
{"name":"CRIME_365_PROPERTY"},{"name":"CRIME_365_ALL_CRIME"},{"name":"CRIME_180_CAR_BREAK"},
{"name":"CRIME_365_CAR_THEFT"},{"name":"CRIME_ALL_ALL_CRIME"},{"name":"CRIME_180_ROBBERY"},
{"name":"CRIME_180_PROPERTY"},{"name":"CRIME_365_TRUCULENCE"},{"name":"CRIME_30_CAR_BREAK"},
{"name":"CRIME_180_VIOLENCE_AGAINST_PERSON"},{"name":"CRIME_365_PUBLIC_SPACE"},
{"name":"CRIME_180_CAR_THEFT"},{"name":"CRIME_30_ALL_CRIME"},{"name":"CRIME_60_BURGLARY_THEFT"},
{"name":"CRIME_365_CAR_BREAK"},{"name":"CRIME_365_BURGLARY_THEFT"},
{"name":"CRIME_60_VIOLENCE_AGAINST_PERSON"},{"name":"CRIME_30_BURGLARY"},
{"name":"CRIME_180_ABUSE"},{"name":"CRIME_60_CAR_THEFT"},{"name":"CRIME_30_BURGLARY_THEFT"},
{"name":"CRIME_365_VIOLENCE_AGAINST_PERSON"},{"name":"CRIME_365_BURGLARY"},
{"name":"CRIME_60_ROBBERY"},{"name":"CRIME_30_TRUCULENCE"},{"name":"CRIME_30_CAR_THEFT"},
{"name":"CRIME_30_ROBBERY"},{"name":"CRIME_180_PUBLIC_SPACE"},{"name":"CRIME_30_STEAL"},
{"name":"CRIME_180_STEAL"},{"name":"CRIME_180_BURGLARY"},{"name":"CRIME_180_TRUCULENCE"},
{"name":"CRIME_365_ROBBERY"},{"name":"CRIME_365_STEAL"},
{"name":"CRIME_30_VIOLENCE_AGAINST_PERSON"}]}
```


6. ábra Bűnügyi térkép API /Crimes JSON válasz

A Crime('id') táblával pedig a fent kinyert id-k segítségével, a konkrét bűnesetre készíthető lekérdezés. Erre példa a hetedik ábrán látható, ahol a 60 napos lakásbetörésekre keresett a hallgató.


```
{
  "shortInfo": "2021.10.21",
  "longInfo": "lakásbetörés\\r\\n2021.10.21\\r\\nMersevát, Hargita út\\r\\n\\r\\n",
  "lat": "17.203645",
  "lon": "47.290121"
},
```


7. ábra Bűnügyi térkép API /Crimes('id') JSON Deszerializáció után

A felhasznált URI:

https://terkep.police.hu/api/odata/v1/Crime('CRIME_60_BURGLARY')

Ezt konvertálás (tesztprogramban Newtonsoft.Json .JsonConvert.DeserializeObject) segítségével a nyolcadik ábrán látható formára lehet hozni. Még ezt szükséges lesz jobban feltördelni, de látható, hogy ahhoz az adathoz sikerült jutni, amelyre valóban szüksége lesz az alkalmazásnak.

Az ábrán látható mező a következő információkat tartalmazza:

longinfo: Bűncselekmény megnevezése, elkövetés dátuma, település (Budapest esetén a kerület is meg van adva).

shortinfo: Elkövetés dátuma.


{"@odata.context":"$metadata#Crime/$entity","name":"CRIME_60_BURGLARY","data":[{"shortInfo":"2021.08.28","long
rés\r\n2021.08.26\r\nBudapest 02. ker, Kavics út\r\n\r\n","lat":"19.033503","lon":"47.521036"},{"shortInfo":"2021.0
2021.08.27","longInfo":"lakásbetörés\r\n2021.08.27\r\nLetenye, Akácfa út\r\n\r\n","lat":"16.729679","lon":"46.4294
ngInfo":"lakásbetörés\r\n2021.10.18\r\nZalaegerszeg, Átalszegett út\r\n\r\n","lat":"16.831533","lon":"46.833804"},
s\r\n2021.10.06\r\nSzázhalombatta, Erkel Ferenc út\r\n\r\n","lat":"18.908904","lon":"47.312269"},{"shortInfo":"202
örés\r\n2021.09.06\r\nNagylóc, Petőfi Sándor út\r\n\r\n","lat":"19.577732","lon":"48.033164"},{"shortInfo":"2021.08
":"lakásbetörés\r\n2021.10.13\r\nTiszakeszi, Jókai út\r\n\r\n","lat":"20.986401","lon":"47.787042"},{"shortInfo":"20
r\n2021.09.28\r\nMezőkövesd, Teréz út\r\n\r\n","lat":"20.56842","lon":"47.815988"},{"shortInfo":"2021.09.27","long
örés\r\n2021.09.30\r\nJászkisér, Hunyadi út\r\n\r\n","lat":"20.224477","lon":"47.45477"},{"shortInfo":"2021.09.18",
13\r\nHeves, Fő út\r\n\r\n","lat":"20.272984","lon":"47.588078"},{"shortInfo":"2021.09.17","longInfo":"lakásbetörés\
szúhát út\r\n\r\n","lat":"19.801273","lon":"47.049456"},{"shortInfo":"2021.09.07","longInfo":"lakásbetörés\r\n2021.0
zi Ferenc út\r\n\r\n","lat":"22.46349","lon":"47.970333"},{"shortInfo":"2021.10.14","longInfo":"lakásbetörés\r\n2021.
Kossuth Lajos út\r\n\r\n","lat":"17.923725","lon":"46.770654"},{"shortInfo":"2021.09.13","longInfo":"lakásbetörés\r\
r\n2021.09.21\r\nBudapest 11. ker, Lágymányosi út\r\n\r\n","lat":"19.052753","lon":"47.477412"},{"shortInfo":"2021
lnfo":"lakásbetörés\r\n2021.09.28\r\nKiskunhalas, Batthyány út\r\n\r\n","lat":"19.486855","lon":"46.425645"},{"sho
örés\r\n2021.09.19\r\nBakonyoszlop, Táncsics út\r\n\r\n","lat":"17.923755","lon":"47.347557"},{"shortInfo":"2021.1
nfo":"lakásbetörés\r\n2021.10.12\r\nJászberény, Neszűr tanya 1565 hrsz. út\r\n\r\n","lat":"20.336478","lon":"47.2
10.08","longInfo":"lakásbetörés\r\n2021.10.08\r\nSzigethalom, Dunasor út\r\n\r\n","lat":"19.020012","lon":"47.299
o":"2021.10.12","longInfo":"lakásbetörés\r\n2021.10.12\r\nHajdúszoboszló, Kenézy út\r\n\r\n","lat":"21.386452","
ásbetörés\r\n2021.09.02\r\nBudapest 14. ker, Felsőbüki Nagy Pál út\r\n\r\n","lat":"19.129087","lon":"47.507067"},
nfo":"lakásbetörés\r\n2021.10.15\r\nKecskemét, Ballószög út\r\n\r\n","lat":"19.656912","lon":"46.883889"},{"shortI
r\n2021.08.31\r\nÚjkígyós, Hosszú út\r\n\r\n","lat":"21.004107","lon":"46.592021"},{"shortInfo":"2021.09.13","longI
\nPutnok, Vásártéri út\r\n\r\n","lat":"20.424697","lon":"48.295154"},{"shortInfo":"2021.09.29","longInfo":"lakásbetörés
\\nNyírkáta, Petőfi út\r\n\r\n","lat":"22.25295","lon":"47.864486"},{"shortInfo":"2021.08.27","longInfo":"lakásbetörés\\r
élyes, Petőfi Sándor út\r\n\r\n","lat":"18.611967","lon":"47.111599"},{"shortInfo":"2021.09.29","longInfo":"lakásbetör
ág út\r\n\r\n","lat":"19.133008","lon":"47.883922"},{"shortInfo":"2021.09.09","longInfo":"lakásbetörés\r\n2021.09.09
ecsend, Demjeni út\r\n\r\n","lat":"20.350321","lon":"47.812793"},{"shortInfo":"2021.09.27","longInfo":"lakásbetörés\
major út\r\n\r\n","lat":"17.836602","lon":"46.453513"},{"shortInfo":"2021.10.21","longInfo":"lakásbetörés\r\n2021.10
út\r\n\r\n","lat":"17.417508","lon":"46.579839"},{"shortInfo":"2021.09.07","longInfo":"lakásbetörés\r\n2021.09.07\r\
aki út\r\n\r\n","lat":"21.706529","lon":"47.962192"},{"shortInfo":"2021.09.27","longInfo":"lakásbetörés\r\n2021.09.27
\r\n\r\n","lat":"21.71461","lon":"47.949859"},{"shortInfo":"2021.10.20","longInfo":"lakásbetörés\r\n2021.10.20\r\nN
s király út\r\n\r\n","lat":"21.426566","lon":"47.142395"},{"shortInfo":"2021.09.11","longInfo":"lakásbetörés\r\n2021.0
lep út\r\n\r\n","lat":"20.752558","lon":"48.099484"},{"shortInfo":"2021.09.18","longInfo":"lakásbetörés\r\n2021.09.18

8. ábra Bűnügyi térkép API /Crimes('id') JSON válasz

lat: Szélességi kör, a térképen megjelenő pont elhelyezéséhez.

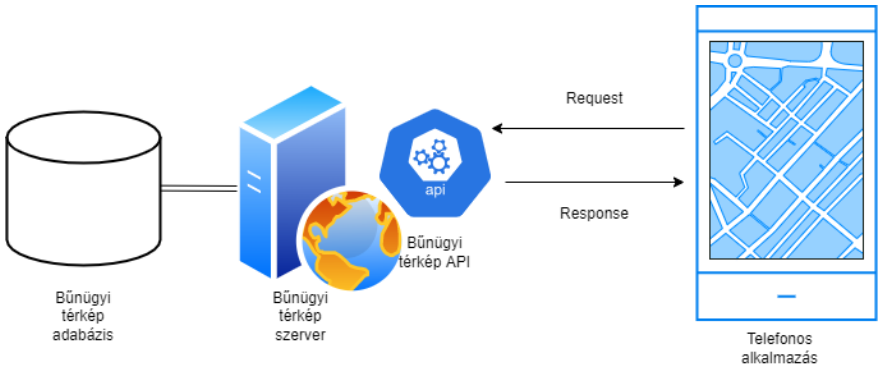
long: Magassági kör, a térképen megjelenő pont elhelyezéséhez.

A felsorolt mezők közül a legfontosabb a longinfo. Itt található minden, a szakdolgozatban létrehozandó alkalmazás számára szükséges adat. Ahogy az látható is, az adatot még formázni kell a térképes alkalmazás számára. Emellett szűrni kell a redundáns, illetve a felesleges táblákat. Az utóbbi 60 nap bűncselekményeit jeleníti majd meg a térkép, így felesleges a 180 és 360 napos adatokat lekérni és feldolgozni. Ezzel jelentősen lecsökkenthető a program futtatásához szükséges erőforrás.

3. Saját szerver

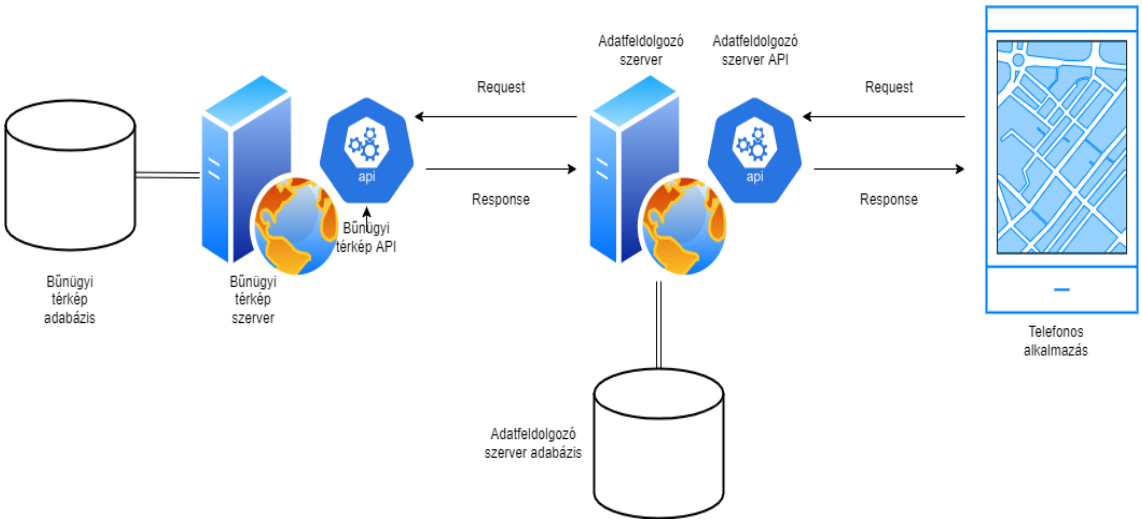
3.1. Szükséges-e szerver

Az adat eljuttatását az alkalmazáshoz, két féle képpen is meg lehet oldani. Első megoldásként egy szervergép nélküli rendszer kerül górcső alá, második megoldásként a szervergéppel rendelkező rendszer vizsgálata fog folyni. A megoldásokat a nyolcadik és kilencedik ábra illusztrálja.



9. ábra Rendszer saját szerver nélkül

Az első megoldás során a telefonos alkalmazás végzi, mind az adat lekérését mind a feldolgozását. Előnye, hogy feleslegessé teszi a szervergéppel való munkát. A telefon közvetlenül csatlakozik a rendőrség szolgáltatásához, majd az onnan kapott adatokat maga az alkalmazás formázza és kezeli, majd jeleníti meg. Hátránya, hogy az adat mennyiségének köszönhetően jelentősen megnőhet az applikáció erőforrás igénye, amely ronthat a felhasználói élménye., lassíthatja az alkamazást. A hallgató tesztprogramjának, esetenként két percbe is telt, az API-ról való adatszerzés, bármiféle más folyamat elvégzése nélkül. A bevezetőben az egyik alapprobléma az volt, hogy telefonról használva a bűnügyi térképet, lassú felállás tapasztalható. Ezt a megoldást választva sajnos fent áll a veszély, hogy egy olyan problémát sikerül átörökíteni az applikációra, amelynek pont, hogy a javítása volt a cél. Emellett hátrány a bűnügyi térkép időszakos elérhetetlensége. A hallgató többször is tapasztalta, az oldal és mellette az API elérhetetlenségét (2021.11.17 2021.11.28). Belátható, hogy az alkalmazásnál fontos a folyamatos, szünet nélküli működés, így a bűnügyi térkép API használata a telefonos alkalmazás végpontjáként erősen átgondolandó.



10. ábra Rendszer saját szerver implementálásával

A második megoldásnál a leginkább erőforrás igényes folyamatokat, amik az első megoldásnál lassulást okoznak, a szerver végzi. Az alkalmazás már csak a szűrt és formázott adattal dolgozik. Előnye, hogy ezzel sikerül tehermentesíteni az applikációt és függetleníteni a bűnügyi térkép API időszakos leállásaitól. Hátránya, hogy a rendszer kibővül egy újjabb komponenssel, a szervergéppel, amelynek köszönhetően megnő a projektbe fektetett munka mennyisége és az implementációtól függően esetleges többlet költséggel lehet számolni. Ennel a megoldásnák a szervergépnek tartalmaznia kell majd egy adatbázist és az adatbázis kezelését végző saját fejlesztésú szoftvert. Emellett egy egyszerű logoló alkalmazás telepítése is célszerű, ezzel segítve az esetlegesen felmerülő hibák javítását.

A hallgató a második megoldás mellett döntött. Szeretné ha az alkalmazás működését nem befolyásolna egy a harmadik fél részén fellépő hiba, emellett fontos, hogy optimálisabb, gyorsabb legyen a program betöltése mint a bűnügyi térképé, hiszen csak így lesz releváns opció a hivatalos oldal mellett.

3.2. Szerver Típusa

Miután eldőlt, hogy szükség lesz saját szervergépre, érdemes megvizsgálni milyen választási lehetőségek vannak a típusát illetően. Az elkövetkezendő bekezdésekben három fajta szervergép kerül tárgyalásra, majd ezek közül kerül az egyik kiválasztásra. A választási lehetőségek: az on-premise, a felhőszolgáltatótól bérelt virtuális gép és a saját virtuális gép.

3.2.1 On-premise

Az on-premise alatt a saját fizikai szervergépet kell érteni. Ennél a fajta megvalósításnál van a legnagyobb kontroll a kialakítás felett. A szerver alatt húzódó infrastruktúra minden pontja, a számítógép összes hardware komponense és a szerverre kerülő szoftverek, a kialakítást végző személy által vannak kiválasztva, így ez a legszemélyreszabhatóbb. Előnyei, a bérelt virtuális gépnél kisebb a havi fenttartási költség és nagyobb biztonság. Hátránya, hogy ennek a legmagasabb a kezdő költsége, hiszen minden komponenst be kell szerezni hozzá, a szoftvereken felül. Ezen kívül limitáltak a teljesítmény javítási lehetőségek. Csakis a számítógép hardware-én lehet fejleszteni, ami ismételten növeli a költségeket.[12]

3.2.2 Felhőben való bérlés

A felhőszolgáltatótól való bérlés, mint Amazon Web Services, Microsoft Azure Cloud vagy Google Cloud Platform kézenfekvőnek tűnhet. Lényege, hogy egy fizikai szerveren több virtuális gépet hoznak létre majd ezeket adják bérbe. Egy ilyen rendszerek az NIST modell alapján öt darab karakterisztikával rendelkezik.[13] Ezek:

- **On-demand self-service:** Az erőforrások önkiszolgáló módon érhetők el.
- **Broad network access:** Folyamatos és szélessávú internet elérés biztosított a felhőhöz.
- **Rapid elasticity:** Erőforrások szükség szerinti növelése és csökkentése.
- **Resource pooling:** A szolgáltató, „erőforrás medencével” rendelkezik. Ez foglalja magába a kiosztható erőforrásokat a felhasználók felé.
- **Measured service:** Az erőforrások használatát méri a szolgáltató és ez alapján számláz.

Előnye, hogy megnövekedett igénybevétel esetén automatikusan bővíti a szerver erőforrásait. Biztosítva van, hogy mindig csak azért az erőforrásért fizet a berlő amelyet valóban használ. Egyszerűen, pár kattintással bérelhető egy szerver, így üzembehelyezése viszonylag egyszerű, csupán a saját szoftverek, adatbázisok, és szükséges komponensek telepítése a feladat. A bérelt rendszer magától készít biztonsági mentéseket, így garantált, hogy komoly hiba esetén sem történik nagymértékű adatvesztés. Itt meg kell említeni, hogy az ilyen szolgáltatások rendkívül magas elérhetőségi százalékkal működnek, például az AWS-t 99.9% availability-vel reklámozzák. Ez nincs háromnegyed óra leállás éves szinten.[14]

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	Network performance
t2.nano	$0.0067	1	0.5 GiB	EBS Only	Low

11. ábra t2.nano árazása

Hátránya, hogy nem a legbiztonságosabb választás, az ilyen rendszerek ki vannak téve a szomszédsági támadásoknak. Tekintve, hogy egy fizikai szerveren több virtuális gép került kialakításra, fent áll a veszély, hogy az egyik ott tárolt gépről indítanak egy sikeres támadást a saját, egyezon gépen elhelyezkedő virtuálsi gép ellen.[15] Költségeket tekintve ez jóval kedvezőbb mint egy on-premise kialakítás, ellenben így sem a legpénztárcabarátabb. Ismételten Amazon példával élve a legolcsóbb EC2 (AWS Elastic Computing) árazása és specifikációja a tizedik ábrán látható.[16]

Átszámolva éves költségre, ez a mai árfolyamon számolva 18.901,35 forint. Rendkívül jól hangzik, ugyanakkor a teljesítménye elégtelen. A hallgató tesztporgramja közel 1GB memóriát képes felhasználni futási időben, így a fél GB nem megfelelő. Az 1GB csak az API hívás során felmerülő teljesítmény igény, erre még érdemes rászámolni.

m6g.medium	$0.046	1	4 GiB	EBS Only	Up to 10 Gigabit

12. ábra m6g.medium árazása

A tizenegyedik ábrán látható virtuális gép már kielégítő lehet, de ez többletköltséget eredményez. Így az éves ár már 129.770,45 Ft.

3.2.3 Saját virtuális gép

A saját virtuális gép kialakítása jár a legkevesebb önköltséggel. Hypervisor segítségével készíthető és szabható személyre a szerverként funkcionáló virtuális gép. Ez egy olyan szoftver melyenek segítségével emulálható egy vagy több virtuális gép a fizikai gépen. Amennyiben a saját fizikai számítógépe amelyen helyet kap a virtuális gép, elég erőforrással rendelkezik és egy ingyenes Hypervisor szoftvert választ hallgató, megoldható akár ingyen is, nem számolva az áramfogyasztást és internet díjját. Hátrány, hogy több munkát igényel a kialakítása mint a bérlés, hiszen az erőforrások kiosztásától, operációs rendszer és az üzemeléshez szükséges szoftverek telepítéséig, mindent saját magának kell csinálnia a szakdolgozónak. A folyamatos elérhetőség sem annyira biztos mint például az AWS esetében, mivel egy áramkimaradás esetén megszűnik a szolgáltatás.

3.2.4 Választás

Költség és implementáció nehézsége. Ez alapján a két tulajdonság alapján kerül kiválasztásra a szervergép típusa. Költség tekintetében egyértelmű választás a saját kialakítású virtuális gép. A hallgató rendelkezik virtuális gép futtatására alkalmas hardware-rel, emellett ingyenes Hypervisor is rendelkezésére áll már. Implementáció nehézsége a bérelt környezetnél a legalacsonyabb, ugyanakkor a hallgató már kellő tapasztalattal rendelkezik, hogy ehhez a rendszerhez magától is el tudja készíteni a környezetet, így felesleges terhelnie magát a bérlés költségeivel. Az on-premise megoldás is megfontolásra került. A hallgató saját számítógépét használná szervergépnek Hypervisor nélkül, de a már telepített programok befolyásolnák a zavartalan működést, így ez az ötlet elvetésre került. A választás így a saját virtuális környezet kialakítására esett.

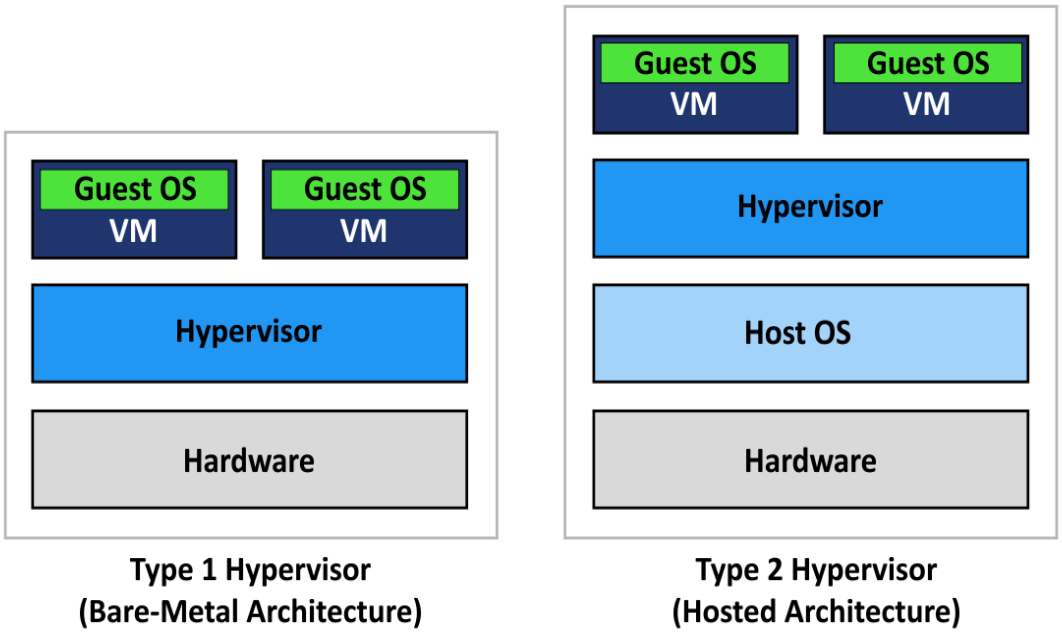
A rendszer védelmére nem kell túl nagy hangsúlyt helyezni, ugyanis hacsak nem a magára a szerver gépre törnek be, nem valószínű, hogy sikerülne akármilyen adatot megváltoztatni, törölni, vagy hozzáadni, hiszen az adatbázis kezelő rendszer (angolul DBMS) csupán HTTP GET paranccsal szerzi-meg az amúgy is publikus információt, majd ugyan ilyen módon adja tovább. Egyéb támadási pontot jelenthet a rendőrség adatbázisa, de ez kívül esik a projekt hatáskörén.

Ugyan úgy a skálázhatóság sem fontos szempont, hiszen a rendszer működésének a teszteléséhez, elég egy mobiltelefon, jelen esetben nem kell nagyszámú felhasználó kiszolgálását biztosítani.

Bővíthetőség akkor lenne szempont, ha a nagyközönség számára is elérhetővé kéne tenni az alkalmazást. Ez jelenleg nem a terv része. Egyelőre, csupán kis méretben kerül implementálásra a rendszer, saját használatra nem szükséges scalibility-vel törődni.

3.3. Hypervisor

Hypervisor segítségével hozható létre és konfigurálható virtuális gép. Egyfajta számítógépeket emuláló szoftverként lehet rá tekinteni.[17] A virtuális gépek konfigurációja és képfájlokként való léterhozása, majd ezen képfájlok futtatása az alapfeladata. Ez a program egy plusz réteget képez, az ebben az esetben fizikai gép és virtuális gép között. Erre illusztráció a a tizenkettedik ábra. A hallgató tanulmányai során két Hypervisor használatát sajátította el, így ezek közül választana. Ezek a VirtualBox és



13. ábra Hypervisor architektúrális helye számítógépen

a VMWare. Az előbbi egy ingyenes open-source szoftver, míg az utóbbinak szintén van ingyenes változata a VMware Workstation Player, de ezen kívül rendelkezik fizetős változatokkal is. Ezesetben az ingyenes változatok közül történik majd a választás. A projekt szempontjából egy fontos különbség említhető meg. A VirtualBox esetében, készíthető snapshot, míg VMWare esetén ez egy fizetős szolgáltatás. Ugyan a szerveren található összes adat el van tárolva a rendőrségnél is, de a konfigurációról nem árt ha készül biztonsági mentés, így a hallgató a VirtualBox-ot tartja a jobb megoldásnak, ennek köszönhetően, ez lesz használva a rendszer implementálásához.

3.4. Operációs rendszer

Az operációs rendszer kiválasztása egyszerűbb, mint a szervergép típusáról dönteni. Feladatai közé tartozik, hogy futtassa a hallgató által készített alkalmazást és a továbbítsa szerverhez érkező kéréseket, majd a válaszokat, ezáltal végpontot biztosítson a telefonos applikációhoz. A hallgató a szakdolgozat írásáig három szerver operációs rendszerrel dolgozott, ezek a CentOS, a Windows Server és az Ubuntu Server. Tekintve, hogy az OS-sel szemben nincs támasztva semmilyen speciális elvárás a fent meghatározottakan kívül, ebből következik, hogy ezek közül bármelyik elégséges lehet.

3.4.1 Windows Server

A legfrissebb (2019-es) Windows Server lenne a legkényelmesebb választás. A felülete hasonló bármelyik Windows operációs rendszérehez, így kezelése könnyen tanulható. Minimális hardware igénye, 1,4 GHz 64 bites processzor, 512 MB RAM és 32 GB szabad tárterület.[18] Egyedül a tártelüret igénye okozhat problémát. Van lehetőség ingyenes próbaverzió letöltésére amelyet 180 napig lehet használni. Mivel a szakdolgozat megvalósítása, ideális esetben nem vesz ennyi időt igénybe, ez elég is lehet.[19]

3.4.2 CentOS

Következő a CentOS. Ennek a használatához 10 GB tárhelyre, 768 MB RAM-ra, és egy minimum 1.1 GHz-es processzorra van szükség.[20] Ingyenesen elérhető, vizuális felülettel rendelkező operációs rendszer, amelynek legnagyobb előnyei közé sorolják a stabilitását és biztonságát. Hátránya, hogy a fejlesztő cég, Red Hat, az IBM általi felvásárlása után bejelentette a támogatás megszüntetését. A CentOS 8-nál 2021-el, a CentOS 7-nél 2024-el kezdődik meg az End of Life időszak.[21]

3.4.3 Ubuntu Server

Az Ubuntu Server, kínálja a legszemélyreszabhatóbb rendszert. Rendszer követelményei 512 MB RAM, 1GhZ-es processzor, 1,75 GB tárhely. Az ezzel való munka nehézkes lehet, mivel nincs grafikus felülete (habár telepíthető) , csupán szöveges. Ugyanakkor személyreszabhatóságának köszönhetően, a legtöbb feladatra, így a szakdolgozat rendszerének futtatására is konfigurálható.[22]

3.4.4 Választás

A választás az Ubuntu szerverre esett. Kis mérete, folyamatos támogatottsága és személyreszabhatósága tökéletes választássa teszi. Emellett nem kell aggódni a próbaverzió lejárta és támogatás megszűnéséből adódó problémák miatt.

3.5. Web szerver alkalmazás

A szervernek tudnia kell fogadni az API-hoz érkező HTTP kéréseket, ezt továbbítani magának a programnak, majd elküldeni a program válaszát. E célból egy web szervert kell telepíteni. A két legelterjedtebb web server alkalmazás amelynek ajánlják telepítését

Ubuntu környzetere, az Nginx és az Apache[23], így a hallgató is ezek közül fog választani. Mindkettő

A legnagyobb különbség a kettő között, a kérések kezelésének architektúrája. Az Apache minden kéréshez új folyamatot hoz létre, ez nagy forgalom esetén jelentős teljesítmény csökkenéshez vezet. Az Nginx aszinkron, kezeli a kéréseket, így nem kell minden kapcsolathoz külön folyamatot rendelni. Ennek köszönhetően kevesebb memóriát fogyaszt és gyorsabban kezeli a requesteket.[24]

Ugyan a szakdolgozatban megvalósítandó rendszernek nem kell majd nagyszámú kéréseket kezelnie, de egy esetleges jövőbeni közzététel esetén ez egy fontos pont, ezért az Nginx-re esett a választás.[24]

3.6. Specifikáció

Hypervisor: Oracle VM Virtualbox

Operációs rendszer: Ubuntu Server 20.10 – Legfrissebb.

RAM: 4 GB – Kezdeti érték, igény szerint változhat a rendszer tesztelése közben.

Processzor: 2 mag - Kezdeti érték, igény szerint változhat a rendszer tesztelése közben.

Hálózat: NAT – A host gép IP címének használata teszteléshez.

Tárhely: 20 GB

3.7. Adatbázis

3.7.1 Adatmennyiség

A 2020-as statisztikai adatokra hivatkozva,[25] 162416 bűncselekmény került elkövetésre. Ebből az következik, hogy kb. 456 bűncselekmény történik naponta. A hallgató semmilyen adatot nem szeretne megjeleníteni, amely a rendőrség térképén, nem szerepel, így az alkalmazásban is a 60 napnál nem régebben elkövetett bűncselekmények lesznek láthatóak. Ez körülbelül 27360 rekord, amelyeket még 3 részre kell bontani, bűntett megnevezésére, településre, utcára és dátumra. Eszerint a számítás szerint az adatbázisban egyidejűleg kb. 109440 rekord lesz eltárolva, beleszámolva a sorokhoz tartozó azonosítókat.

3.7.2 Felépítés

A hallgató egy darab táblában tervezi eltárolni az összes kigyűjtött bűnesetet. Szét lehetne szedni bűnelkövetés azonosítójaként egy-egy táblára, de ebben az esetben fent áll a veszély, ha új megnevezést vezet be a rendőrség az nem fog megjelenni. Biztosabb, ha a rendszer ömlesztve gyűjti ezeket az adatokat, az azonosítóban szereplő szám alapján. (Ezesetben az a szám 60, ahogy a forrás tárgyalásánál már említésre került.)

Az adatbázisban 4 oszlopra lesz szükség:

- **Id:** Sima szám típusú mezők oszlopa. Minden rekordhoz rendelve lesz, egy INT szám az adatbázisban való tájékozódás végett.
- **Bűncselekmény megnevezése:** Szöveges mező. Például „lakásbetörés”, vagy „tulajdon elleni szabálysértés”, nem a rendőrség által használt Id mint „CRIME_60_BURGLARY”, így ez megjeleníthető konverzió nélkül a telefonos alkalmazásban.
- **Cím:** Szöveges mező. Itt kerül eltárolásra a település neve és a cím. Budapesti cím esetén a kerületet is tartalmazni fogja.
- **Dátum:** Bűnelkövetés időpontja.

Majd ezen, 4 oszlop adatait kell megosztani API segítségével.

3.7.3 Típus

Fő kérdés az SQL és NoSQL közötti választás. Felmerülhetne az in-memory tárolás is, az-az az adatok nem merevlezemzre íródnak, hanem maradnak a memóriában-ban, ami gyorsabbá tenné az írás-olvasás műveleteket, viszont a megnövekedett teljesítmény igény miatt nem éri meg.

SQL

Az SQL nyelv segítségével egy relációs adatbázisban tárolhatók el az adatok. Ez tökéletes, jól struktúrált információval való munkához, és annak kezeléséhez, ugyanakkor rugalmatlan megoldás. Amennyiben változás történik a tárolandó információban, pl. típusa megváltozik dátumról, számra vagy egy új oszlopot kell felvenni, akkor a teljes táblát módosítani kell, rossz esetben minden ott tárolt értékkel egyetemben.[26]

NoSQL

A NoSQL sokkal dinamikusabb, könnyebben változtatható fájlrendszerbe írja az adatokat. Relációs adatbázis helyett „dokumentumokban”, pl. JSON-ben vagy XML-ben tárol. Amennyiben az adatbázis változtatására van szükség, úgy könnyűszerrel implementálható , az addig tárolt adatok módosítása, esetleg átstruktúrálása nélkül.[27]

Választás

A térkép adatbázisa nem kell, hogy rugalmas legyen. Az oszlopainak típusai nem fognak változni és a benne tárolt adatok struktúrája sem. Az adatbázisban való keresésnek nagy szerepe lesz, hiszen az adott utcában elkövetett bűncselekményeknek kell megjelennie. Ennek köszönhetően az SQL lesz a helyes választás, hiszen a keresés jóval gyorsabb egy ilyen típusú adatbázisban, mint, hogy újra be kelljen olvasni az eltárolt XML vagy JSON dokumentum egészét majd ezután elvégezni a keresést.

3.8. API program

A hallgató az API rendszer megvalósításához a Visual Studio 2019-et fogja használni, ebben C# nyelven fog dolgozni. A keretrendszer ASP.NET. Rendkívül fontos, hogy olyan környezettel dolgozzon, amelyben már van programozói tapasztalata, ezáltal elkerülve, egy másik programnyelv elsajátításakkor felmerülő idő és produktivitás veszteséget. Tanulmányai során, ebben a programozási környezetben és ezen a nyelven fejlesztett adatbázis kezelő alkalmazást és ismerkedett meg API program fejlesztéssel, a munkája kapcsán pedig mind API lekérdezés mind, kész API program telepítése, a feladatai közé tartozott. A szakdolgozó ismeretei, ezzel a környezettel kapcsolatban a legbővebbek, ezért esett erre a választás.

3.8.1 Feladatai

A program feladatai. Naponta egyszer kérje le az adatot a bűnügyi térkép API-járól. Ezt az adatot formázza és mentse el adatbázisban. Majd GET kérés érkezésekor, a paraméter alapján adja vissza, azokat a sorokat ahol eggyezést talált.

3.8.2 Adat lekérés

Az adat lekérése már tárgyalásra került, ez nem több két metódusnál. Ehhez szükség lesz egy közbenső osztályra melynek adatstruktúrája megegyezik a kapott JSON dokumentumáéval. A következő konverziónak ez lesz a kiinduló típusa.

Napi lekérdezés

A lekérő metódus, napi egyszeri futtatásánál a logika, hogy meg kell adni egy időpontot, mikor hajtsa végre a metódusokat. Ha ez az időpont egyezik a jelenlegi időponttal lefut. A lefutás logolásával, kivédhető, bármilyen hiba miatti kimaradása a lekérdezésnek. Ha a logban szereplő dátum több mint egy nappal kevesebb az aznapi dátumnál, automatikusan fusson le újra. A log lehet egy sima txt fájl amelyet a program meghatározott időnként ellenőrizhet (pl. óránként).

Formázás

Az adat formázásának a célja, a forrásból kapott információ feltördelése a telefonos alkalmazás számára használható formára.

lakásbetörés\r\n2021.10.12\r\nBékéscsaba\r\nBékéscsaba, Kastély út

garázdaság, rendbontás\r\n2021.10.30\r\nBudapest 10. ker, Liget Út\r\n

Az API-ról érkezett JSON formátumú adat deszerializációja után kapott string itt látható. Amennyiben a string „\r\n” karaktersoronként kerül feltördelésre (Split használatával), úgy megkapható a három darab szükséges cella. Az összes cella szöveg típusú kivéve a második cellát, amelyet még át kell konvertálni dátum típusúra.

Eltárolás és kezelés

A kapott adat eltárolásához, egy Visual Studio-ban létrehozható Service-based Database kerül felhasználásra. Ezzel egy .mdf fájlt kreál a program. Ebben vannak eltárolva az adatbázis adatai. Ezek után, kézileg SQL nyelven kell a táblát létrehozni, ahova mentésre kerülnek az adatok.[28]

Miután a tábla készen van, össze kell kapcsolni a rendszert kezelő programkóddal. Ehhez az ADO.NET Entity Data Model-t fogja használni a hallgató.[29] Ennek segítségével, egy modellként vagy osztályként lehet kezelni az adatbázis egyes tábláit, ezáltal egyszerűbb az adatok menedzselése. E célból a CRUD metódusok implementálása a következő feladat, az-az

- **Create** – Létrehozás.
- **Read** – Kinyerés, kiolvasás.
- **Update** – Frissítés, módosítás.
- **Delete** - Törlés.

Ezek bármelyik adatbázis négy darab alapműveletei. Amely forrástól lekért bűncselekmények még nem szerepelnek az adatbázisban azokat Create-tel kell hozzáadni, amelyek már nem szerepelnek azokat Delete-tel törölni.

API interfész biztosítás

A szerveren futó alkalmazás API-ja, csupán egy fajta kérésre fog válaszolni, ez pedig egy paraméteres GET kérés. A paraméter egy lista, amely a felhasználó által kiválasztott pont, bizonyos sugarán belül lévő címeket tartalmazza. Ügyelni kell rá, hogy a telefonos applikáció által küldött adat, ugyan olyan formátumú legyen, mint az adatbázisban eltárolté. A paraméterként kapott címekkel lefut egy keresés az adatbázisban és egyezés esetén, a válasz tartalmazni fogja a címhez köthető összes információt, amit majd meg kell jeleníteni a térképen.

Az API létrehozására, a hallgató az ODATA használata mellett döntött. A térképet működtető rendszernek és annak infrastruktúrájának elkészítése, meglehetősen nagy energiabefektetést igényel, így az interfész kialakításának leegyszerűsítése, vonzó választássá teszi. ASP.NET keretrendszerben a következők az implementálásának lépései:

ODATA

Első lépés az ODATA-hoz tartozó NuGet csomag letöltése.

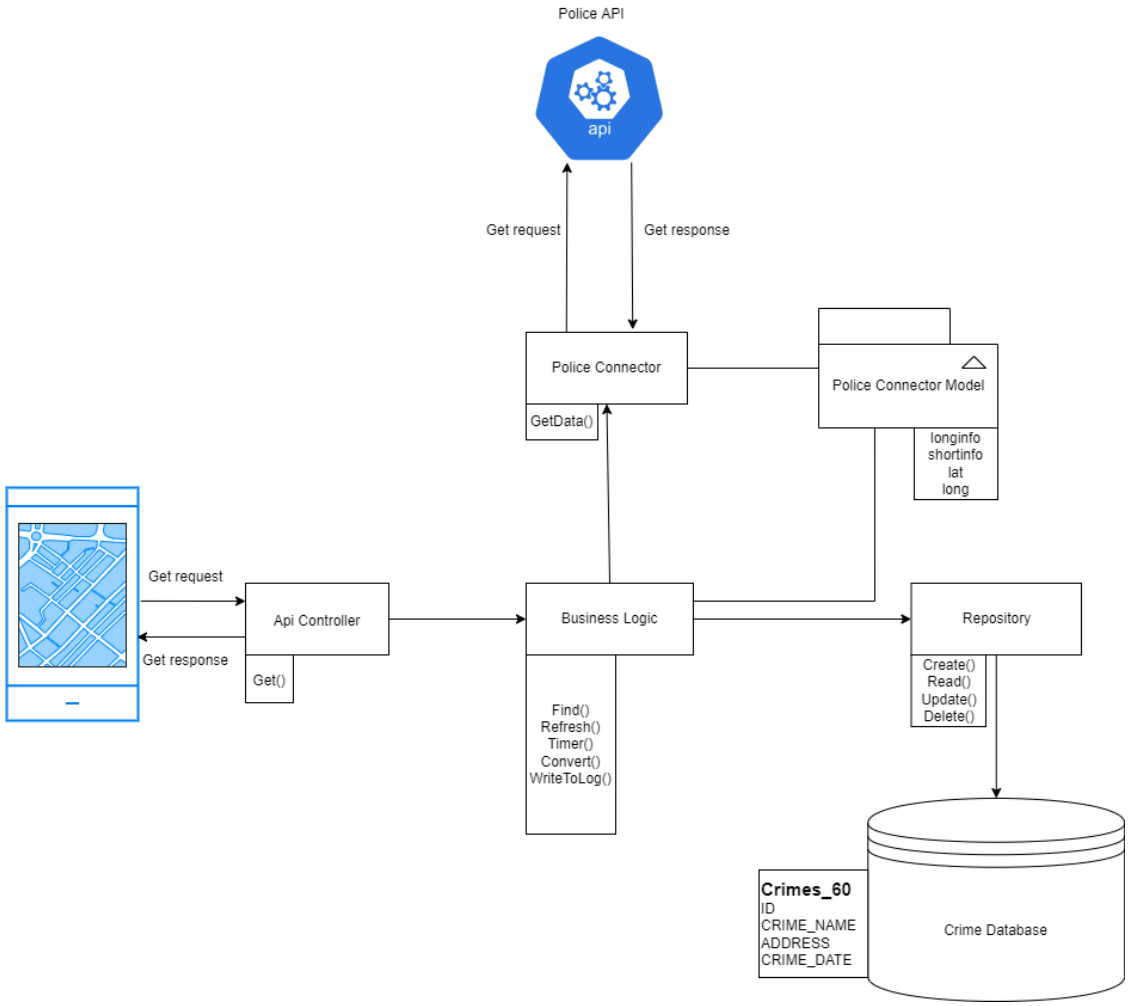
Miután ez sikerült, definiálni kell a modell osztályokat. Minden modell osztálynak ajánlatos tartalmaznia egy [Key] tulajdonságú mezőt, amely a kulcs, és egy [Required] tulajdonságú mezőt, amely a kulcshoz kötelezően megadandó érték. Ebben az esetben az osztály minden mezője rendelkezni fog ezzel az attribútummal.

Következőnek, a modell osztályból, vagy osztályokból, egy vagy több adatforrást kell kreálni. A honlapjukon elérhető oktatóanyagban, egy listát töltenek fel a saját modell osztályuk, futásidőben generált elemeivel. Saját szerver esetében Entity Framework-ön keresztül lesz csatlakoztatva az adatbázis, ez fogja szolgáltatni a modellt.

A Kontroller osztályokban kell meghatározni, mely kérésekre, milyen válaszokat küldjön a szerver. Az oktatóanyagban GET került definiálásra, amely a fent létrehozott adatforrást adja vissza. Saját programnál a GET metódusnak szöveges paraméterként lesznek átadva a keresendő utcák. Publikus adattal folyik a munka, így a hívás URI-ja tartalmazhatja a címek listáját, nem szükséges elrejteni. A paraméterek alapján lefut a keresés, és ennek az eredménye lesz válaszként visszaküldve JSON formátumban.

Utolsó pont az Endpoint konfigurálás, ahol az interfész URI-ja kerül meghatározásra. A rendszer teszteléséhez http://localhost:[portNumber]/Crimes/lista elérési út elégséges.[30]

3.8.3 Architektúraterv:



14. *ábra Szerveren futó API kezelő program felépítése*

3.9. Adatbázist kezelő szoftver

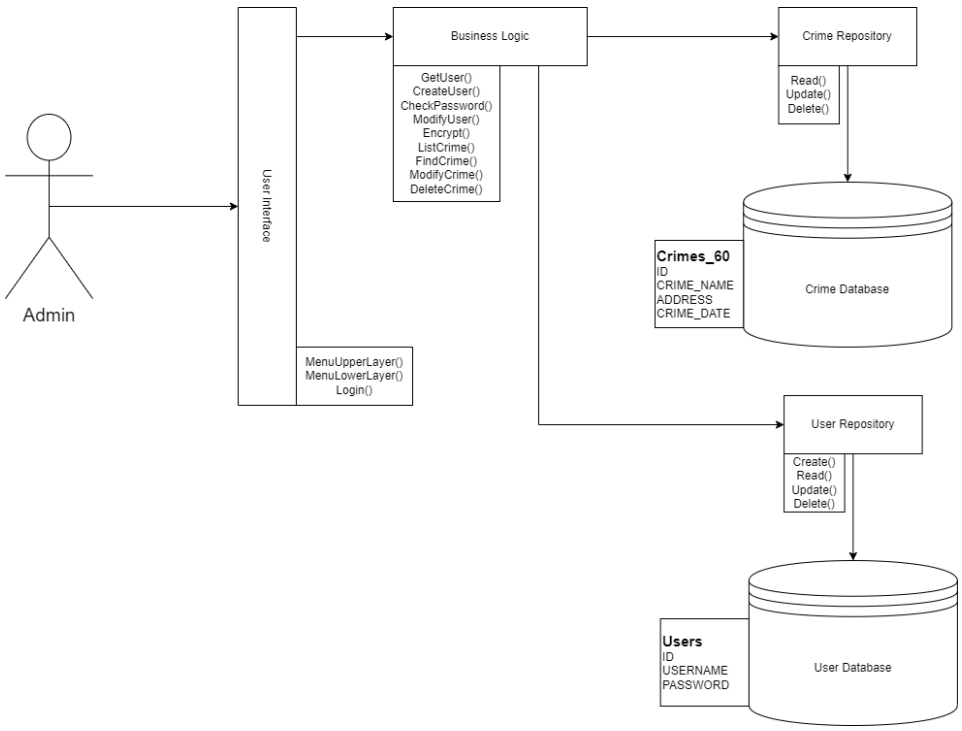
A tárgyalt, szerveren futó program, így már képes, a telefonos alkalmazás végpontjaként üzemelni. Viszont ha valami hiba történik, például rossz adat kerül eltárolásra az adatbázisban, azt nem lehet könnyűszerrel kijavítani. A fenti program autonóm működésre lett tervezve, felhasználói kezelése nem implementált. Ehhez egy másik programra lesz szükség, amellyel kézileg módosítható a rendszer bűnügyi adatbázisa. Az operációs rendszer natív felhasználói interfésze szöveges, így ez a program is konzollal lesz kezelhető.

Ezen alkalmazást csakis egy megfelelő jogkörrel rendelkező személy használhatja, emiatt felhasználó kezelésre lesz szükség. E célból konzolos bejelentkezési ablak lesz adva a programhoz, amely felhasználói név és jelszó nélkül nem engedélyez hozzáférést. A jelszavakat, nem szabad sima szövegként eltárolni, az adatbázis biztonsága érdekében, csakis titkosítva.

Bejelentkezéshez egy újabb Service-based Database lesz létrehozva, ID, USERNAME és PASSWORD oszlopokkal. Titkosításra az SHA 256 algoritmus implementálása mellett döntött a hallgató, mivel borzasztóan kicsi az esélye, hogy két azonos kulcsot generáljon, két különböző jelszónak. A jelszó begépelésekkor a rendszer rögtön átkonvertálja azt, majd összehasonlítja az adatbázisban található string-gel. Amennyiben megegyeznek, a felhasználó hozzáférhet a DBMS-hez (Database Management System).

Mindenképpen valós adatoknak kell megjelenni a térképen, így az új adatbázis menedzselő szoftverben nem kerül implementálásra a Create() metódus, ezzel is csökkentve bármilyen valótlan információ, bűnügyi adatbázisba kerülését.

3.9.1 Architektúraterv:



15. ábra Szerveren futó adatbázis kezelő program felépítése

4. Térkép Alkalmazás

A mobil applikációval kapcsolatban az első kérdés, hogy melyik operációs rendszerre történjen a fejlesztés. Ezt nagyban befolyásolja, hogy az applikáció elkészítése milyen szoftver segítségével és milyen nyelv használatával lehetséges, így ezek egyszerre kerülnek tárgyalásra. Ahogy az már leszögezésre került a két választási lehetőség az IOS és az Android. Más operációs rendszerek is vannak még használatban, de a piaci részesedésük olyannyira kicsi, hogy fejlesztési szempontból elhanyagolhatók.

4.1. Feladat

Az alkalmazás feladata, hogy miután a felhasználó kiválasztott egy adott pontot Magyarország térképén, a pont köré rajzol, egy a ponttól és kijelzőtől függő, sugarú kört. A kör sugarának állíthatónak kell lennie, a felhasználó határozza meg, mekkora területre kíváncsi.

A körön belül lévő címeket elmenti, majd GET kérés paramétereként küldi az applikáció szerverének. A kérésre kapott információt fogja megjelenni a térképen. Minden új pont kijelölésekkor szükséges request indítása, így az applikáció használata folyamatos internetkapcsolatot igényel.

A bűncselekmények megjelenítéséhez egy ikon lesz elhelyezve, az utcának valamelyik körön belüli pontján. Erre nyomva, egy, az x-edik ábrán látható táblában tekinthető meg az ott elkövetett bűncselekmények megnevezése és időpontja.

A térképen való közlekedésnek, illetve nagyításnak és kicsinyítésnek két módja lesz. Az egyik az ujjal való mozgatás, a más térképes alkalmazásokon megszokott mozdulatokkal, a másik a térkép jobb alsó sarkán található kezelővel.

A térkép implementációjához Google Map API kulcsra lesz szükség. Ez egy kisebb projekt számára ingyenesen is igényelhető kulcs, amelyet el kell helyezni az alkalmazás kódjában.[31]

4.2. Megjelenítő felület

Az alkalmazás három ablakkal fog rendelkezni. A fő ablak maga a térkép lesz. A másik két ablakot a bal felső sarokban helyet kapó, menügombbal lehet megnyitni. Ezek a kredit ablak, melyben a rendőrség által kikötött forrásmegjelölés kap helyet, illetve a segítség ablak, amiben egy bűncselekmény bejelentéséhez szükséges telefonszámok olvashatók.

4.3. Környezet

4.3.1 Android

Az első vizsgálandó rendszer az Android. Ez egy Linux alapú, open-source mobil operációs rendszer, amelyet a Google dobott piacra 2007-ben.[32] A két OS közül, ennek jóval nagyobb a piaci részese, 2021-ben a telefonok 83,8% működik Android-dal. Ez a Google üzletpolitikájának köszönhető, hiszen akármelyik gyártónak engedélyezi a rendszerük használatát, emiatt sokkal szélesebb vásárlói kört tudhat magáénak.[33] A nyílt forráskód hátránya, hogy az erre publikált applikációkat, könnyen törik fel, így a vetélytársával szemben sokkal nagyobb probléma a kalózkodás. Különbség még az applikációk haszonszerzés módja. Androidon főleg ingyenes programok találhatók, melyek reklámokkal válnak profitábilissá.

4.3.2 Fejlesztés

Az Androidra történő fejlesztés is tükrözi a platform könnyű hozzáférhetőségét. Egy alkalmazás megosztása Google Play-en, egy egyszeri 25$-os összeg befizetéséhez kötött, feltéve, hogy a fejlesztő betartotta a cég irányelveit.[34][35] A fejlesztéshez, a leginkább ajánlott eszköz az ingyenesen letölthető Android Studio. Ebben a fejlesztői környezetben 2017-ig csupán Java-ban, azóta Kotlin nyelven is fejleszthető alkalmazás, 2019-ben pedig bejelentették, hogy az utóbbi a preferált nyelv. A fejlesztői környezet a legtöbb operációs rendszerre telepíthető.

4.3.3 IOS

Az IOS-t csupán az Apple eszközei használják és használhatják. Az operációs rendszert 2007-ben tárták a nagyközönség elé, az első generációs iPhone megjelenésével. Jóval zártabb rendszer, mint az Android, forráskódja nincs közzétéve, ennek köszönhetően biztonságosabb, autentikálatlan szoftvert csupán a telefon feltörésével lehet telepíteni, ezért kevesen használnak kalóz szoftvert.

4.3.4 Fejlesztés

Az erre való fejlesztés abból a szempontból könnyebb, hogy kevesebb eszköz specifikációit kell figyelembe venni. Miközben az Android esetében mára már több mint 24.000 típusú eszköz van használatban,[36] ami közel lehetetlenné teszi, hogy az applikáció mindegyiken tökéletesen nézzen ki és működjön, úgy az IOS 14 esetén ez a szám körülbelül 31.[37] Az alkalmazás optimalizációjának lehetősége ennek köszönhetően IOS-en jóval nagyobb. Hivatalos fejlesztői környezete az XCode. Ebben Swift nyelven lehet alkalmazást fejleszteni. Az XCode legnagyobb hátránya, hogy tekintve Apple termék, csak a saját rendszerükön, a macOS-en használható. A hallgatónak két lehetősége lenne ennek a telepítésére, vagy vesz egy Apple eszközt amely macOS-t használ, ami jelentősen megnövelné a költségeit vagy virtuális gépre telepíti az operációs rendszert, és erre szerzi be a szoftvert.

4.4. Választás

4.4.1 XCode

Az XCode használatát zárta ki először a hallgató. Szeretné a saját telefonján tesztelni alkalmazását és mivel nem rendelkezik semmilyen erre alkalmas eszközzel, vásárolni pedig nincs módjában így ezen környezet választásával fellépő tetemes többletköltségek miatt eliminálásra került.

4.4.2 Android Studio

Az Android Studio vonzó az ingyenes mivolta miatt, emellett a szakdolgozó rendelkezik Android-os készülékkel így a két IOS esetén felmerült probléma itt nincs jelen. Nagyobb probléma a Java nyelv alapszintű és a Kotlin nyelv ismeretének teljes hiánya. Felmerülhet a kérdés, hogy ezesetben van-e lehetőség egy már jobban ismert programnyelv használatára.

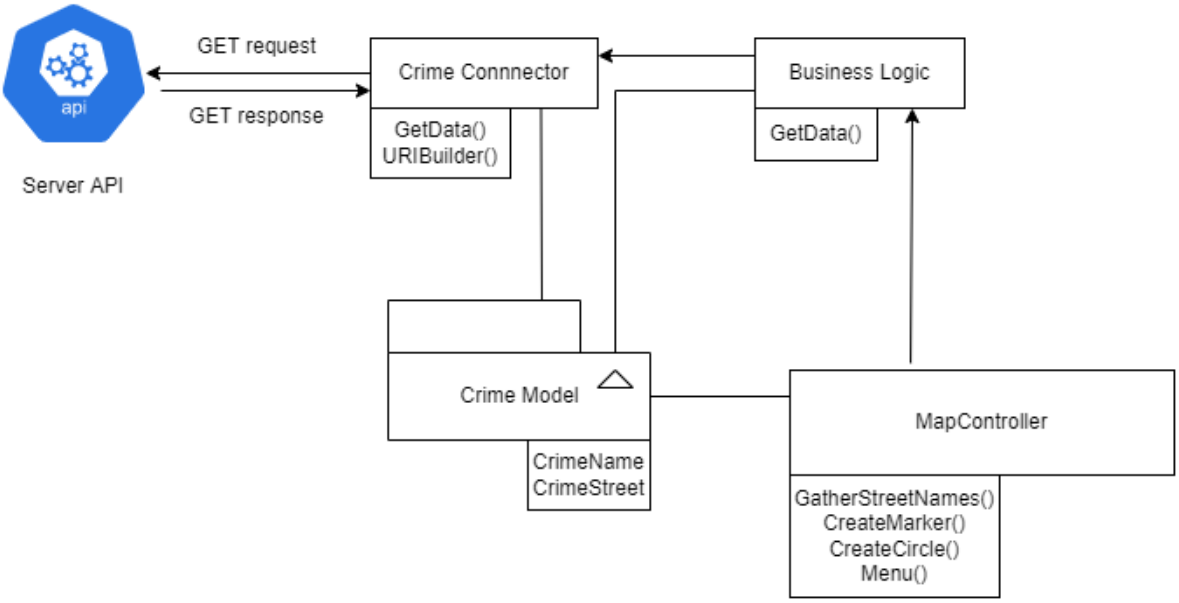
4.4.3 Xamarin

Van, a hallgató által vizsgált lehetőség a Xamarin, amely a Microsoft cross-platform megoldása. Használatával Visual Studio-ban lehet fejleszteni mind Androidra mind IOS-re, C# nyelven. Ugyan 2022 novemberében megszűnik a támogatása és a .NET MAUI veszi át a helyét, de az azonos keretrendszernek köszönhetően az alkalmazás migrációja nem ütközik nagy problémákba. A térkép alkalmazáshoz szükséges komponensekkel, mint Google Maps template, térképen való tájékozódáshoz és közlekedéshez kellő kezelőkkel mind el van látva.[38][39][40][41]

Hátrányai, hogy limitált a hozzá található dokumentációk száma, a platformokra érkező frissítések, csak bizonyos késéssel kerülnek implementálásra, illetve egy bonyolultabb felhasználói interfész elkészítése nehéz és nem kecsegtet szép eredménnyel.[42][43][44][45]

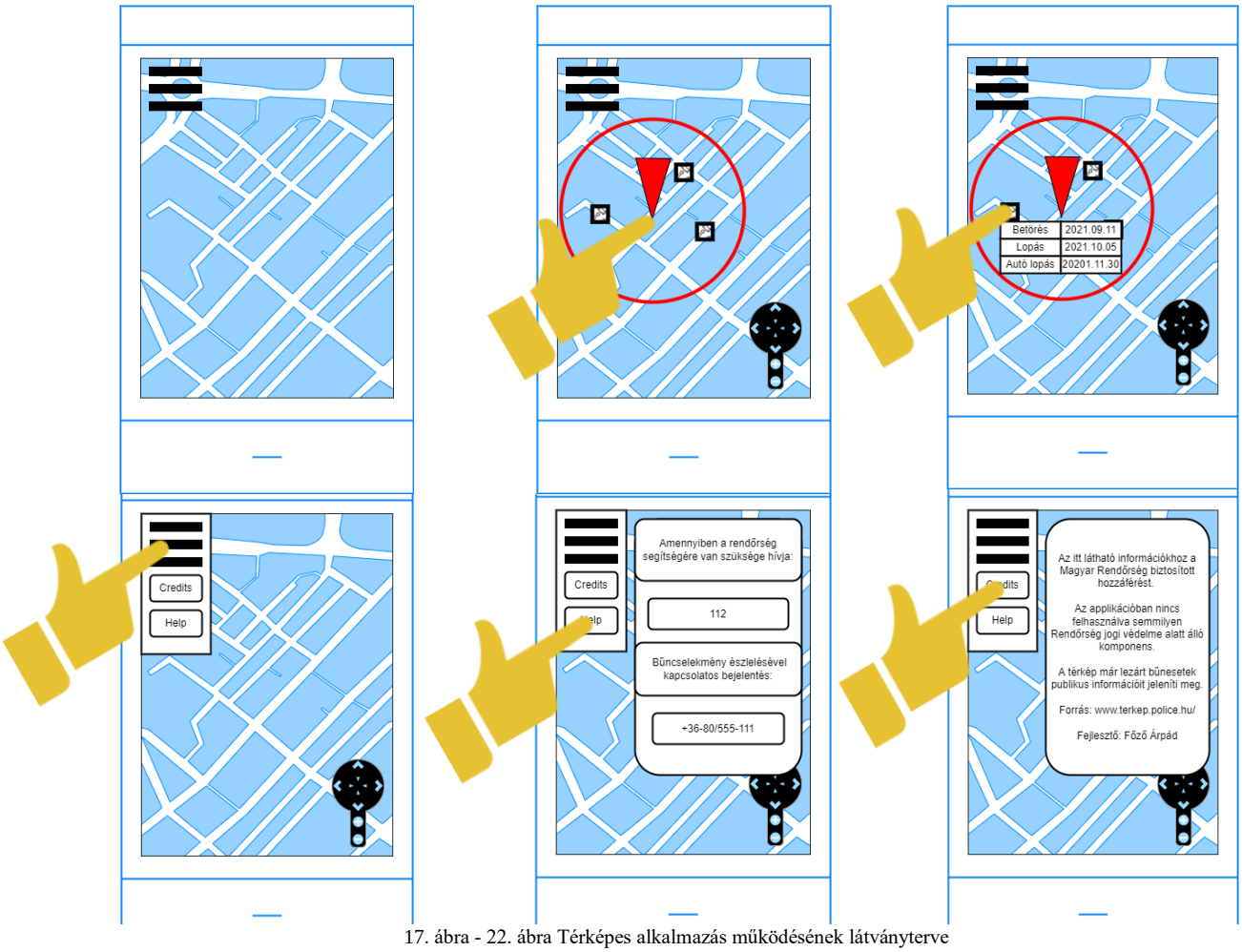
A térképes alkalmazás nem áll sok elemből, így a UI kialakítását érintő hátrányok nem nagy befolyásoló tényezők. Legfőbb komponens a térkép, amelynek elemeihez elég leírás található. Tesztrendszer révén a frissítések késése másodlagos. A nyelv ismerete ugyanakkor, olyannyira megkönnyíti a munkát, hogy erre esett a választás. Androidra fog történni a fejlesztés a rendszer saját mobilon való éles teszteléséhez.

4.5. Architektúraterv



16. ábra Térkép alkalmazás felépítése

4.6. WireFrame



17. ábra - 22. ábra Térképes alkalmazás működésének látványterve

5. Rendszer specifikációi:

- Hypervisor:

  Az ingyenes mivolta és a biztonsági mentés funkciójának köszönhetően az **Oracle Virtualbox**-ra esett a választás. A honlapjukon https://www.virtualbox.org/wiki/Downloads, olvasható, hogy a legfrissebb változat már nem támogat 32 bites hostokat. A munkaállomásként használt számítógép 32GB RAM-mal rendelkezik, így ez nem okoz problémát.

- Operációs rendszer:

  A kis méreténe és az alacsony rendszerkövetelményei miatt az **Ubuntu Server 20.10**-re esett a választás. Probléma ezzel kapcsolatban, hogy nincs grafikus felülete, viszont tekintve, hogy a telepítendő applikációt, konzolon keresztül lehet majd kezelni, ez nem szempont.

- RAM:

  Tekintve, hogy csak 64 bites hostokkal folyhat a munka minimum **4 GB** RAM-ot kell rendelni a virtuális géphez.

- Processzor:

  A végrehalytandó folyamat, sokkalta inkább memória mint processzot igényes így a virtuális gép létrhozásakkor **2 mag** áll majd a rendelkezésére. A hallgató rendelkezik annyi erőforrással, hogy szükség esetén bővíteni lehessen.

- Hálózat:

  A hálózattal különösebb feladat nincs, így lehet a gazdagép **NAT**-olt interfészét használni.

- Tárhely:

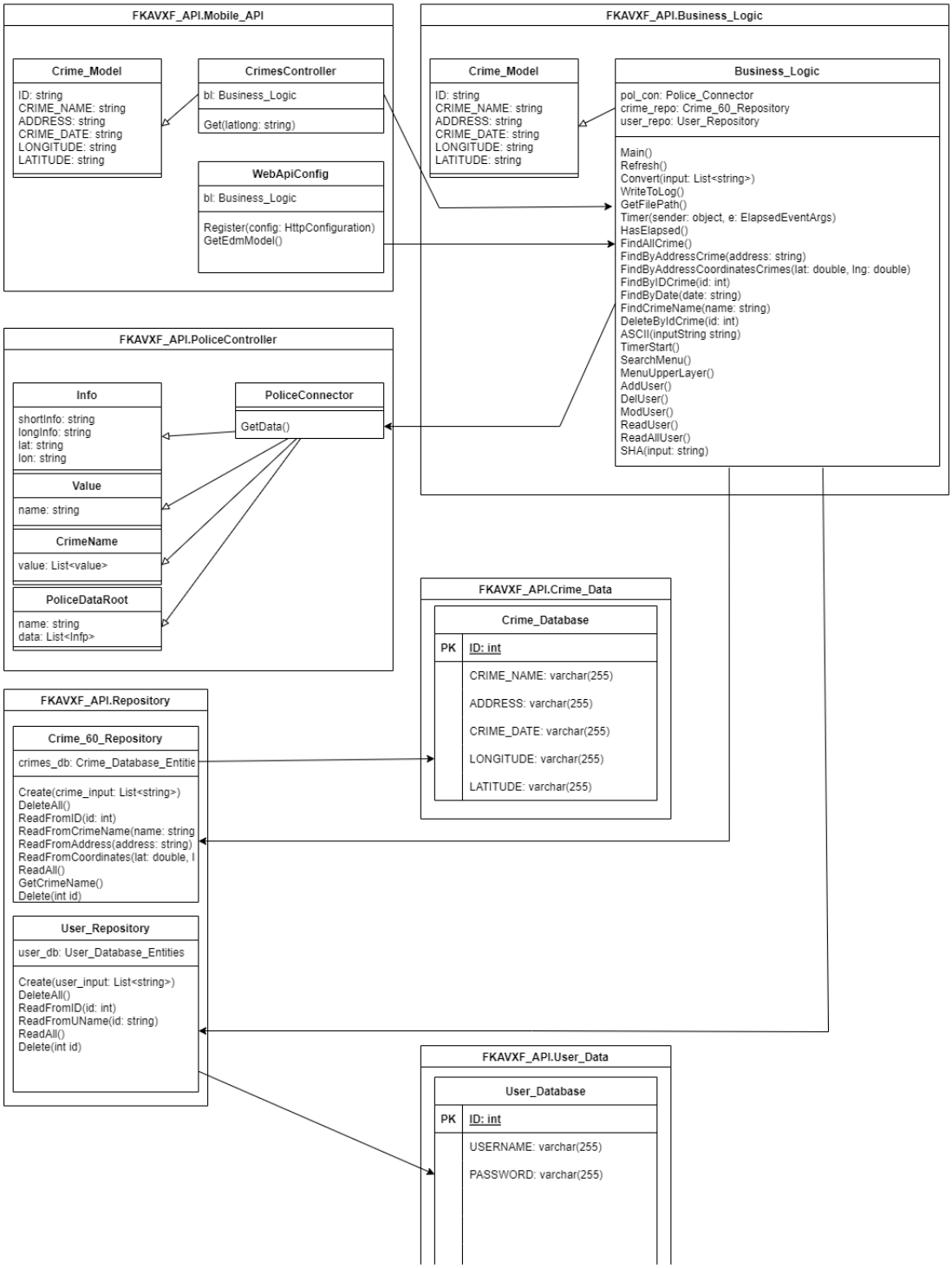
  Tekintve, hogy a hostra nem lesz telepítve, semmilyen nagy tárhelyigényű csomag vagy program, illetve maga az operációs rendszer mérete is kicsi így a **20 GB** kielégítő választás.

- IDE és nyelv:

  A hallgató **Visual Studio 2019 és C#** használata mellett döntött. Ahogy az már tárgyalásra került, a szakdolgozat írónak ezekben van a legnagyobb tapasztalata és nincs idő, új nyelvet megtanulni a projekthez. Ezen kívül a Xamarin bővítménnyel, készíthető Android alkalmazás C# nyelven, így ez tökéletes választássá teszi.

6. API szerver fejlesztési dokumentáció

6.1. UML – Osztály diagram



23. ábra Szerveren futó program osztály diagramja

6.2. Bevezetés

Az API alkalmazásnak, már tervezés fázisban meghatározott, két fontos szerepet kell betöltenie. Először is a bűnesetek adatbázisának, másrészt a telefonos alkalmazástól érkező kérések kezelésére kell képesnek lennie. A forrásként funkcionáló, rendőrségi adatbázissal való kommunikációhoz, négy komponens szükséges. Maga az adatbázis, az adatbázis adatait kezelő repository avagy „adattár", a rendőrség által szolgáltatott végpontottal kapcsolatba lépő és az üzleti logikát megvalósító komponensek. A végpontot kezelő projekt egyetlen feladata, az osztályai hívásakor, egy egyszerű Http Get kérés segítségével, a rendőrség bűnügyi adatainak lekérése, majd ezen adatok továbbítása az üzleti logikát megvalósító projekt részére. Az üzleti logika projektje kezeli, a telefon számára létrehozott API projekten kívül, a teljes a programot. Feladatai közé tartozik az adatbázis frissítésének ütemezése, felhasználók és menü kezelése, ezekmellett a rendőrségtól kapott adatok, használhatóvá formázása.A telefon API feladata, feldolgozni a hívás paraméterét, majd ezen váltózótól függően, visszaadni a felhasználó által kiválasztott térképponthoz közeli bűneseteket.

6.3. Adatbázis

A hallgató a bűnügyi és a felhasználói adatbázisok megalkotásával kezdte a munkát. Ehhez a Visual Studio Service-based Database használta fel a már tárgyalt indokok miatt, majd az ADO.NET Entity model segítségével tette a kezelhetővé az adatbázist. Az adatbázis táblái és mezői a következő táblázatokban láthatók.

Users		
ID	Elsődleges kulcs	egész szám - int
USERNAME	Felhasználónév	szöveg - string
PASSWORD	Felhasználónévhez tartozó jelszó	szöveg - string

Crimes		
ID	Elsődleges kulcs	egész szám - int
CRIME_NAME	Bűncselekmény megnevezése	szöveg - string
ADDRESS	Elkövetés helye	szöveg - string
CRIME_DATE	Elkövetés ideje	szöveg - string
LONGITUDE	Térképen megjelenő ikon hosszúsági koordinátája	szöveg - string
LATITUDE	Térképen megjelenő ikon szélességi koordinátája	szöveg - string

6.4. Repository

Ezen a lépések után következett a Repository-k elkészítése. Csupán az alap CRUD műveleteket az-az Create, Read, Update, Delete volt szükséges implementálni, ugyanakkor mind a Read mind a Delete metódusokból több típusúra volt szükség. Fontos, hogy a bűnügyi adatbázisban lévő adatok, több oszlop alapján is listázhatóak legyenek a karbantartó, illetve a telefonos alkalmazástól érkező API hívások számára. A kész programban az adatok lekérhetőek, ID, utca, dátum, koordináták és a bűncselekmények nevei alapján, illetve az adatbázis összes rekordját is képes visszaadni a Repository. Törlés esetén, két típusú törlést kell implementáni. Az egyik az ID alapján való törlés amely esetleges rossz adat bekerülésénél hasznos, a másik az összes rekord törlése, amelyet minden adatbázis frissítésnél hív meg az üzleti logika.

6.5. Kommunikáció a rendőrség végpontjával

Az adatbázis és a Repository létrehozása után, az adatok megszerzésére került a sor. Ehhez egy új projekten belül három osztály kell. Az első a PoliceConnector_Model osztály, amelynek az adattagjai megegyeznek a rendőrség adatbázisának adattagjaival. A mezők tartalma már tárgyalásra került a forrás vizsgálatánál. Tekintve, hogy az API-tól kapott válasz két adatot tartalmaz, magát a bűncselekmény ID-ját (pl. „ROBBERY_60”) és az id-hoz tartozó rekordok listáját, szükséges létrehozni egy segédosztályt is, ahhoz, hogy a Newtonsoft csomagban található Newtonsoft.Json.JsonConvert.DeserializeObject segítségével, deszerializálni lehessen azt.
Információ osztály:

```
public class Info{
        public string shortInfo { get; set; }
        public string longInfo { get; set; }
        public string lat { get; set; }
        public string lon { get; set; }}
```

Segéd osztály:

```
public class PoliceDataRoot{
        public string name { get; set; }
        public List<Info> data { get; set; }}
```

A harmadik osztálynak az API kérések elküldése és a válaszok fogadása a feladata. Ez két lépésben történik. Először az összes Id-t gyűjti ki az algoritmus egy CrimeName típusú változóba. A CrimeName egyetlen adattagja egy string típusú lista.

```
string crimeName_url = @"https://terkep.police.hu/api/odata/v1/Crimes";
HttpWebRequest request =
(HttpWebRequest)WebRequest.Create(crimeName_url);
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
var jsonResult = JsonConvert.DeserializeObject(content).ToString();
CrimeName nev = JsonConvert.DeserializeObject<CrimeName>(jsonResult);
```

Kikötés volt, hogy csak azok az adatok jelenhetnek meg, amik a rendőrség oldalán is. A szakdolgozat írásának pillanatában, ezek, azon ID-ú bűnesetek, amelyekben szerepel a 60-as szám, emiatt egy plusz szűrés is kell. Ennek eredménye képpen már csak a megjelenítendő bűnesetek maradtak a listában.

```
foreach (var item in nev.value)
{
        if (item.name.Contains("60"))
        {
                ids.Add(item.name);
        }
}
```

Megjegyzés: Az „ids” egy string típusú lista.
Ezen kollekció felhasználásával indít kéréseket, amelyekre érkező válaszban, már a konkrét bűnesetek adatai találhatók. Az algoritmus ezeket az adatokat a kimeneti, PoliceDataRoot típusú listába tölti, majd ez lesz a visszatérési értéke.

```
foreach (var item in ids)
{
HttpWebRequest request2 = (HttpWebRequest)WebRequest.Create(
"https://terkep.police.hu/api/odata/v1/Crime('" + item + "')");
…
}
```

6.6. Formázás

Az itt kapott adatok még nem adhatók át az adatbázisnak, formázásra van szükség. Ezt a feladat a Convert metódus látja el. Az egyetlen nehézséget a loginfo adattag jelenti, tekintve, hogy a következő formában érkezik a szerverhez az adat.

rongálás\r\n2022.03.19\r\nLitér, Dózsa György út\r\n

Ezt kell felbontani a bűneset nevére, dátumára és az utcára. A logika, hogy először a „\r” tagokat kicseréli a program üres karakterre, majd a „\n” tagokat pontosvesszőre. A műveletek után már kezelhetőbb formában van sor.

rongálás;2022.03.19;Litér, Dózsa György út;

Az utolsó lépés, felbontani a karaktersorozatot pontosvesszőnként, így tömbként lesz kezelhető és ahogy látható a sorból, a tömb nulladik eleme lesz a bűncselekmény megnevezése, a második lesz a dátum és a harmadik lesz a cím.

6.7. Üzleti logika

A program alapját képző osztályok ezzel készen vannak, az utolsó feladat az üzleti logika implementálása. Az üzleti logika osztály feladatai a már kész komponensek kezelése, a felhasználók számára menü biztosítása, az adatbázis frissítésének ütemezése és a kapott adatok konvertálása.

6.8. A program kezelése

A felhasználói menü egy egyszerű konzolos menü, amiben számokkal lehet navigálni. Két réteggel rendelkezik, a felső réteg kezeli az általános feladatokat, az alsó réteg a kereséseket.
Felső réteg menüpontjai:
1 - List All: Adatbázis összes elemének kilistázása.
2 - Check Remaing Time: Következő frissítésig hátralévő időt.
3 - Search by...: Alsó réteg megnyitása.
4 - Instant Refresh: Az adatbázis azonnali frissítése.
5 - Delete Record: Rekord törlése ID alapján.
Alsó/kereső réteg menüpontjai:
1 – ID: Id alapján keresés.
2 - Crime Name: Bűneset neve alapján keresés.
3 - Crime Address: Cím alapján keresés.
4 - Crime Date: Dátum alapján keresés.
Megjegyzés: A bűnesetek neveit nem csak hosszadalmas lehet begépelni, de egy felhasználótól sem várható el, hogy megjegyezze azokat, ezért segítségként a Crime Name kiválasztása esetén, kilistázásra kerülnek a tárolt bűnesetek nevei, az 24. ábrán látható módon.

```
1 - List All
2 - Check Remaing Time
3 - Search by...
4 - Instant Refresh
5 - Delete Record
Your choice(1-5):3
1 - ID
2 - Crime Name
3 - Crime Address
4 - Crime Date
Your choice(1-4):2
Your Choices:
lakásbetörés
rongálás
tulajdon elleni szabálysértés
rablás, kifosztás
garázdaság, rendbontás
betöréses lopás
személy elleni eroszakos cselekmény
egyéb közterületen elkövetett buncselekmény
gépkocsifeltörés
gépkocsi lopás, jármu önkényes elvétele
lopás
Crime Name:
```

24. ábra Szerveren futó alkalmazás menüje

6.9. Adatbázis frissítése

Az adatbázis frissítésének ütemezése System.Timers.Timer használatával lett implementálva. A program indításakor elindul egy időzítő, amely óránként meghívja a Timer metódust.

```
var timer = new System.Timers.Timer();
timer.Interval = 1000 * 60 * 60;
timer.Elapsed += new System.Timers.ElapsedEventHandler(Timer);
timer.Start();
```

A Timer tartalmazza, a frissítés felső rétegét, amely megvizsgálja az utolsó frissítés óta eltelt időt, majd meghívja az alsó réteget, amely magáért a frissítésért felel. Először a HasElapsed metódus beolvassa lastUpdate.txt log fájlban eltárolt szöveget, azt átkonvertálja dátummá, majd kivonja az akkori pontos időből. Ha a kettő különbsége nagyobb, mint 24, igaz értéket ad vissza és elindulhat a frissítés. Miután a WriteToLog rögzítette a frissítés időpontját és a DeleteAll kitörölte a bűnügyi adatbázis összes elemét meghívható a Refresh metódus.

```
if (HasElapsed())
{
if (WriteToLog())
Console.WriteLine("Log written.");
if (DeleteAll())
Console.WriteLine("Delete successful.");
Refresh();
}
```

A Refresh metódus, a frissítés alsó rétege. A Police_Connector osztály GetData tagjának hívásával, a temp_crime_to_db segédváltozó, megkapja PoliceDataRoot típusú listát, miután ezt a fentebb tárgyalt módon, az adatbázis számára megfelelő formátumúvá konvertálja. Ezek után, foreach segítségével, egyesével beviszi az adatokat az adatbázisba. Amennyiben ez sikerült, az az a Repository Create metódusának visszatérési értéke egy, kiírja a konzolra a bevitt információt.

```
Console.Clear();
var temp_crime_to_db = Convert(pol_con.GetData());
int counter = 0;
foreach (var item in temp_crime_to_db)
{
if (crime_repo.Create(item) == 1)
{
Console.WriteLine("Row Inserted: " + counter.ToString() + " " + item[0]  + " " +
item[1]);
counter++;}}
```

6.10. API végpont konfigurálása

A két alapkomponens készen van, már képes a program lekérdezni és eltárolni a rendőrség adatait. Következő lépés ezen adatok megosztása a telefon számára. Ehhez a hallgató az ODATA használata mellett döntött. Ahogy az már tárgyalásra került az ODATA csomag egy „best-practice” gyűjtemény, API megvalósításhoz, ezzel segítve és gyorsítva a fejlesztők munkáját. Az ODATA működéséhez három komponens szükséges. A WebApiConfig.cs amelyben definiálásra kerül a végpont, a Controller ami kezeli a beérkező hívásokat és a Model ami a válasz adatstruktúráját adja.

Az alap elképzelés az volt, hogy a hívás paramétereként több utcanév érkezik, majd a válasz, az összes, ezen utcákban elkövetett bűnesetet fogja tartalmazni. Ha e logika mentén történt volna a fejlesztés egy plusz lépésre lett volna szükséges a telefonos applikációban. A válaszként kapott bűnesetek koordinátáinak szűrésére. Meg kellett volna vizsgálni, hogy a válasz bűneseteinek koordinátái, a térképen lévő körön belül vannak-e. Ezt egyszerűbben is meg lehet oldani, olyan módon, amivel tehermentesíthető az alkalmazás. A térképen lévő kör középpontjának koordinátái lesznek a hívás paraméterei, így a szerveren futó program tudja elvégezni az említett számítást. Az API hívás elérési útját a következő porperty-vel lehet definiálni:

[ODataRoute("Crimes({latlong})")]

Példa hívás URL:

http:// 192.168.100.2:7293/Crimes('47,5160696425799;19,0512116253376')

A paraméter tartalmazza a hosszúsági és a magassági koordinátákat, pontosvesszővel elválasztva. Első lépés a paraméter felbontása, majd a két szám, double típusúvá konvertálása. Miután ez sikerült következhet a koordináták alapján keresés. A cél, hogy az adatbázisból visszaadja, a paraméterként kapott pont, ~250 méteres sugarában lévő bűneseteket. Tekintve, hogy a rendőrség, tizedes vessző helyett pontot használ, az eltárolt koordináták vizsgálatához „CultureInfo.InvariantCulture” formátumtípust is meg kell adni.

```
var convertTemp = crimes_db.CRIMES_60.AsEnumerable().Where(t =>
        double.Parse(t.LATITUDE, CultureInfo.InvariantCulture) >= lng - 0.002
        &&
        double.Parse(t.LATITUDE, CultureInfo.InvariantCulture) <= lng + 0.002
        &&
        double.Parse(t.LONGITUDE, CultureInfo.InvariantCulture) >= lat - 0.002
        &&
        double.Parse(t.LONGITUDE, CultureInfo.InvariantCulture) <= lat + 0.002
                );
```

Minél gyorsabban kell képesnek lennie a programnak, megtalálni a feltételnek megfelelő adatbázis sorokat. Emiatt, koordináta számítás helyett, egy konkrét, előre meghatározott értéket használ a program. 0.001 körülbelül 111m-nek felel meg így a 0.002-vel közel pontos eredményt lehet kapni.[46]

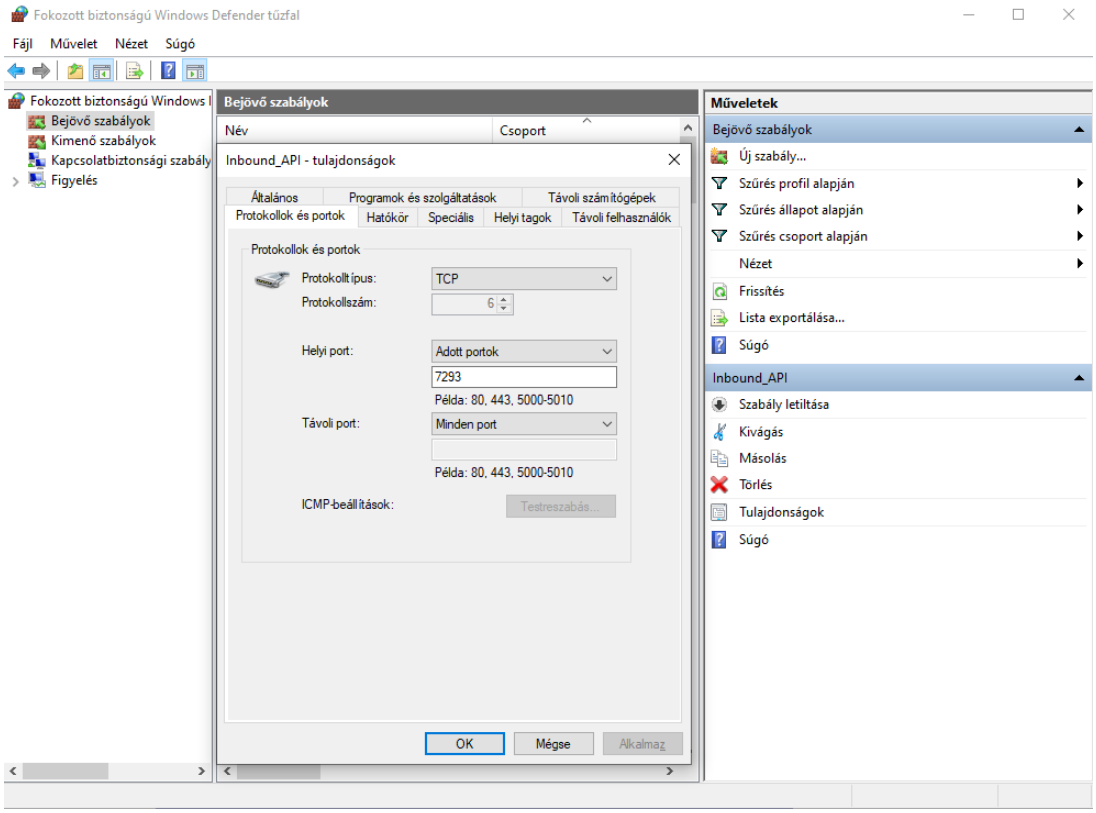
A példa URL-re érkező válaszra:


```
{"@odata.context":"http://192.168.100.2:7293/$metadata#Crimes","value":
[{
"ID":"1774",
"CRIME_NAME":"rongálás",
"ADDRESS":"Budapest 13. ker, Csanády út",
"CRIME_DATE":"2022.01.12",
"LONGITUDE":"19.053089",
"LATITUDE":"47.517664"}},
…
]}
```


6.11. Egyéb beállítások a működéshez

Ahhoz, hogy a szerver tudjon kommunikálni a telefonos alkalmazással, engedélyezni kellett a 7293-as porton való kommunikációt és módosítani kellett a szerveren futó program „applicationhost.config” fájlját, hogy bárhonnan érkező kérésekre válaszoljon. Az előbbit a Windows Defender tűzfalon belül új bejövő szabály létrehozásával (25. ábra), az utóbbit az alábbi sor, konfigurációs fájlhoz adásával lehetett elérni. applicationhost.config: `<binding protocol="http" bindingInformation="*:7293:*" />`

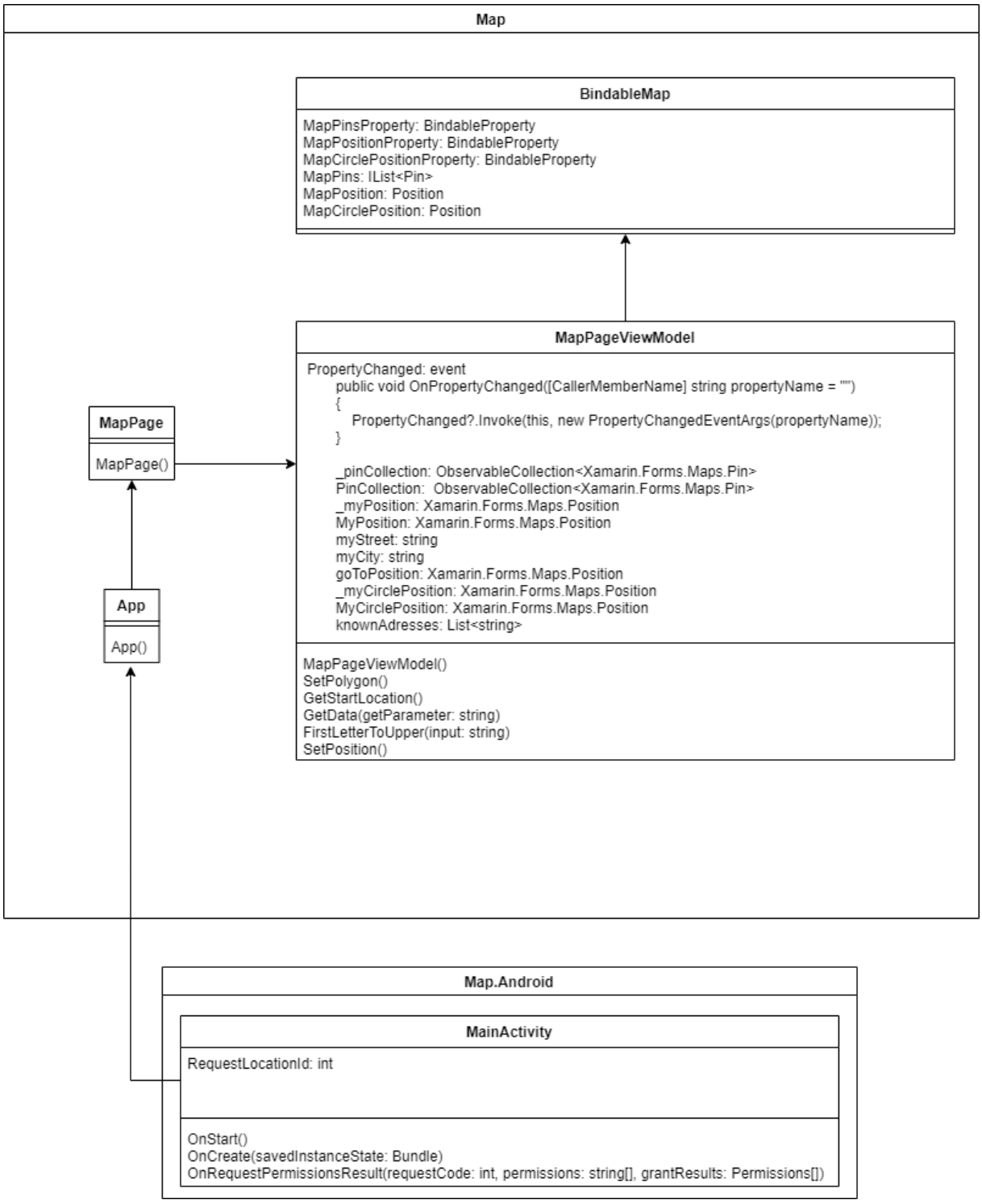
Inbound rule:



25. ábra Inbound szabály beállítása

7. Térkép applikáció fejlesztési dokumentáció

7.1. UML – Osztály diagram

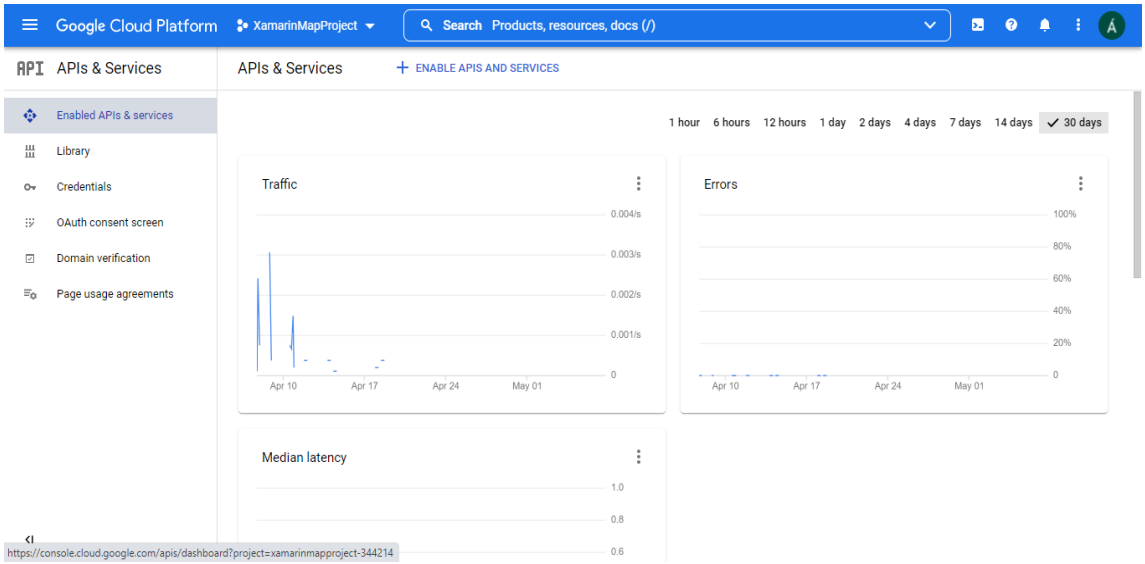


26. ábra Telefonos alkalmazás osztály diagram

7.2. Előkészítés

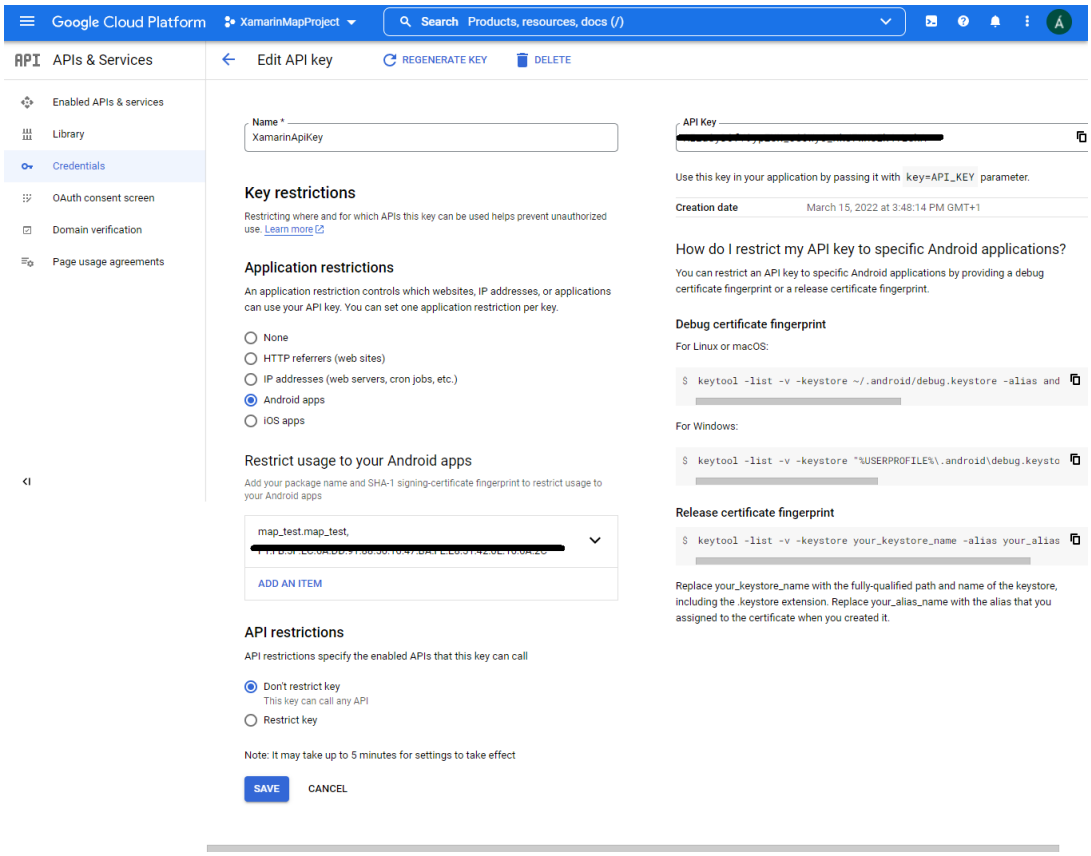
7.2.1 Google Maps API key

Ahhoz, hogy a telefonos alkalmazás rendelkezhessen egy funkcionáló térképpel, szükséges egy API kulcs, amelyet a cloud.google.com címen lehet igényelni. A szolgáltatás fizetős, így egy aktív Gmail fiókon kívül kötelező hozzárendelni számlázási fiókot is. Az ár nem túl magas, 1000 hívásonként 7$-t kell fizetni[50], emellett regisztráláskor, a hallgató kapott 200$ ingyen kreditet, ennek köszönhetően, ez az alacsony költség sem lépett fel. Miután sikerült elkészíteni a fejlesztői fiókot, létrehozható az API kulcs. Generálás után az 27. ábrán latható dashboard fogadja a felhasználót, ahol a kulcs használatának adatait lehet nyomon követni.



27. ábra Google Cloud API kulcs Dashboard

Mielőtt az API kulcs használatba kerül, még érdemes pár beállítást elvégezni a „Credentials” fülnél, mint lekorlátozni az API kulcsot. A hallgató által alkalmazott beállítások (28. ábra) sokkalta inkább, a tesztelés könnyítését szolgálják, alapszintű védelem mellett. Az Application restricitons résznél az Android apps pont lett kiválasztva, aminek hatására csakis Android alkalmazással lehet használni a kulcsot. Az alatta lévő részen, egy SHA-1 certification adható meg, amellyel csak az adott cert. birtokában használható a szolgáltatás. Legalul további pontok adhatók meg, például mely alkalmazásokhoz, API-hoz és SKD-hoz férhetnek hozzá az adott kulccsal.



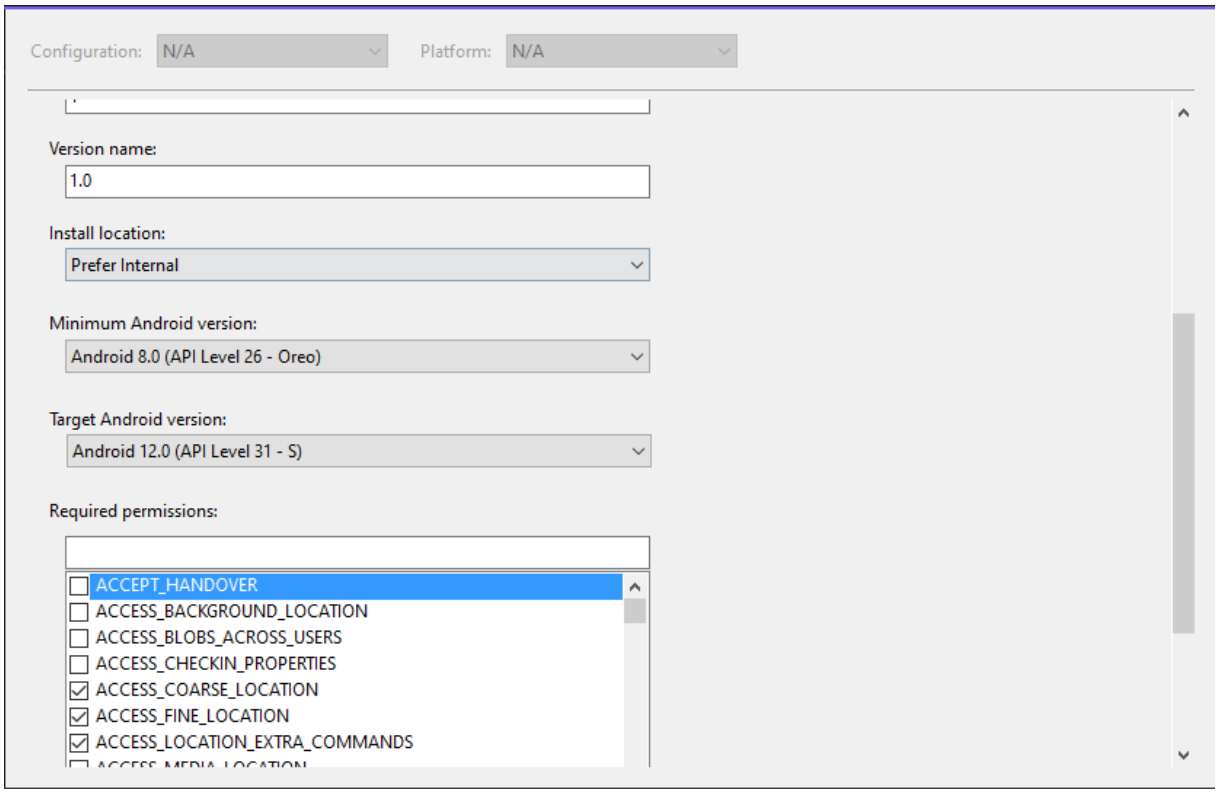
28. ábra Google API kulcs korlátozás ablak

Végül a következő sort kell hozzáadni az AndroidManifest.xml fájlhoz, amely tartalmazza az alkalmazás alap információit az Android eszközök és szolgáltatások számára, és ezzel kész is van a Google Maps térkép használatának az előkészítése:

```
<meta-data android:name="com.google.android.geo.API_KEY"
android:value="*API key*" />
```

7.2.2 Emulátor

Az applikáció tesztelésére két lehetőség van. Az első a saját telefonon, a második egy emulátoron való tesztelés. A hallgató az emulátor használata mellett döntött, a kényelem és az erőforrások könnyebb allokálhatósága végett. Az emulált telefon elkészítéséhez, a Visual Studio Android Device Manager eszközre van szükség. A telefon létrehozásakor kötelező megadni a telefon típusát és az operációs rendszert. Az utóbbira azért érdemes figyelni, mert minden Android verzió más API verzióval rendelkezik és egy alacsonyabb verzióra írt program nem fog működni. Ugyanakkor, ha a legmagasabb verziót választja a fejlesztő, felmerülhet a kérdés, hogy hogyan lehet majd futtatni alacsonyabb verziókon. Megadható a célverzión, vagy Target Android Version-ön kívül egy minimális verziószám, vagy Minimum Andoird Version. Ezek segítségével a két operációs rendszer között kiadott összes verzión működni fog. Az x-edik ábrán láthatók a választott verziók.



29. ábra Telefonos alkalmazás Android beállításai

7.2.3 Jogok

A térkép alkalmazás helyes működéséhez szükséges megkérni pár jogosultságot. A felhasználó dönt, hogy ezeket engedélyezi-e a program futtatásakor.[48]

ACCESS_NETWORK_STATE: Hálózat információihoz való hozzáférés.
WRITE_EXTERNAL_STORAGE: Írás jog külső meghajtóra.
ACCESS_COARSE_LOCATION: Telefon durva (körülbelüli) lokációjához való hozzáférés.
ACCESS_FINE_LOCATION: Telefon pontos lokációjához való hozzáférés.
INTERNET: Végpont nyitáshoz való jog.
ACCESS_LOCATION_EXTRA_COMMANDS: Lokáció extra parancsaihoz való hozzáférés.
ACCESS_MOCK_LOCATION: „Mockolt” lokációhoz való hozzáférés.
ACCESS_WIFI_STATE: Wi-Fi információihoz való hozzáférés.

7.3.Map.Android

Az applikációnak ez a Start-Up projektje. Fontosabb feladatai közé tartozik:

A bővítmények kezelése, ezesetben:

Xamarin.Essentials: Alap Xamarin csomag.
Xamarin.Forms: Alkalmazás felhasználói felületének programozásához.
Xamarin.Forms.Maps: Térkép integrációhoz.
Newtossoft.Json: API válaszokhoz.
Xam.Plugin.Geolocator: Pozíciók koordinátáinak lekérdezéséhez.

A fent felsorolt jogosultságok kérése a felhasználótól:

```
if (CheckSelfPermission(Manifest.Permission.AccessFineLocation) !=
Permission.Granted)
{
RequestPermissions(LocationPermissions, RequestLocationId);
}
```

Inicializálni a komponenseket és betölteni az applikáció következő projektjét:

```
Xamarin.Essentials.Platform.Init(this, savedInstanceState);
Xamarin.Forms.Forms.Init(this, savedInstanceState);
Xamarin.FormsMaps.Init(this, savedInstanceState);
LoadApplication(new App());
```

7.4. Map

A Map projektben találhatók az applikáció modelljei, üzleti rétege és az oldalainak xml fájljai. Az App osztálya konstruktorában, beállításra kerül az alkalmazás főoldala, amelyre példányosításakor átirányít. Ezesetben ez az oldal a MapPage. Minden oldal, alapvetően két komponensből áll. A kinézetéért felelő, .xml és az ezt kezelő programkódot tartalmazó .xml.cs fájlokból. A MapPage esetén az xml fájl legfontosabb része, a következő kódsor, amely létrehozza a térképet:

```
<local:BindableMap
MapType="Hybrid" MapPosition="{Binding MyPosition}"
MapPins="{Binding PinCollection}" MapCirclePosition="{Binding MyCirclePosition}"
MapClicked="Map_MapClicked" IsShowingUser="True"/>
```

7.5. Egyszerűbb beállítások

A térkép komponens natívan kezeli a térképen ujjakkal való navigációt, így ez további beállítást nem igényel. Az IsShowingUser segítségével állítható be a telefon pozíciójának mutatása a térképen. A MapType-al választható ki a térkép típusa. Ezek, Street (31. ábra), Hybrid(32. ábra), Satellite(30. ábra). A hallgató a Hybrid használata mellett döntött a szép kinézet és az utcaneveknek köszönhető könnyű tájékozódás végett.

30-32. ábra Térkép különböző típusai

7.6. Térképen megjelenő elemek

Ahogy a kódsorban látható, a térkép (MapPosition) és a kör (MapCirclePosition) pozíciójának, emellett a rajta található tűknek (MapPins), a kezelését bindolás segítségével más osztályokban végzi a program. Működési elvének a lényege, hogy a hozzájuk kötött változó értékének módosulásakor, lefut a programkód, amely törli a már rajta lévő és létrehozza az új alakzatokat a térképen, a megfelelő helyen. Ehhez három osztályra van szükség. Ezek, MapPage.xml.cs, BindableMap.cs és MapPageViewModel.cs. A MapPage.xml.cs tartalmazza a Map_MapClicked esemény kezelőjét, amely a térképre való kattintáskor hívódik. BindingContext-ként a MapPageViewModel lett megadva, hiszen itt található az összes szükséges adattag. A MapPageViewModel a térképnek és elemeinek a pozícióját tárolja és kezeli. A BindableMap pedig a bindeolt elemek módosítása esetén lefutó programkódot tartalmazza, az-az ez az osztály felel a kör és a tűk megjelenéséért és a térkép mozgatásáért.
Az első feladat, indításkor a telefon pozíciójára navigálni a térképet. A térkép oldalának, a MapPage.xml.cs futtatásával, a MapPageViewModel osztályt is példányosítja. Ekkor,

a konstruktorában lefut egy aszinkron kód, amely lekérdezi a telefon pozícióját, majd odahelyezi a térkép fókuszát. Ehhez szükséges telepíteni Xam.Plugin.Geolocator bővítményt.

```
Task.Run(async () =>{ var position = await
Plugin.Geolocator.CrossGeolocator.Current.GetPositionAsync();
MyPosition = new Xamarin.Forms.Maps.Position(
position.Latitude, position.Longitude);
});
```

A GetPositionAsync visszatérési értéke egy pozíció, amely rendelkezik mind hosszúsági, mind szélességi koordinátákkal, így miután le lett mentve a position segédváltozóba, egyszerű azokat átadni a térkép fókuszának pozícióját tároló, MyPosition-nek. A MyPosition módosulásakor, értesíti a BindableMap osztályban lévő MapPositionProperty-t, amely MoveToRegion(MyPosition) parancs segítségével, a megfelelő helyre irányítja a térképet.

A második a bűncselekmények megjelenítése. Ennek folyamata a következő. A térképre való nyomáskor lefut a MapClicked(System.Object sender, Xamarin.Forms.Maps.MapClickedEventArgs e) metódus, amelynek egyetlen feladata meghívni a MapPageViewModel, void SetPolygon(double latitude, double longitude) metódusát. A SetPolygon paraméterei, az érintés térképen lévő helyének hosszúsága és szélességi koordinátái. Ez a pozíció kinyerhető az esemény argumentumai közül, így a teljes hívás, SetPolygon(e.Position.Latitude, e.Position.Longitude). A metódus először törli a megjelenített bűnesetek adatait tároló, knownAddresses listát, majd elkészíti az API hívás paraméterét. Az API hívás után, válaszban kapott bűnesetek, a getCrime változóban kerülnek eltárolásra.

Az API hívás ugyan azalapján a logika alapján működik, mint a szerveren futó programnál. Első lépésben összeállítja a kérés URL-jét, elküldi a kérést, majd a választ, a NewtonSoft bővítmény segítségével deszerializálja.

```
string crimeURL = "http://192.168.100.2:7293/Crimes('" + getParameter + "')";
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(crimeURL);
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
…
var jsonResult = JsonConvert.DeserializeObject(content).ToString();
var temp = JsonConvert.DeserializeObject<CrimeConnectorRoot>(jsonResult);
```

Ezzel már az alkalmazásnál vannak a szükséges adatok, már csak meg kell jeleníteni egy pin-en. Egy tűnek négy darab alap adattagja van, a pozíció, a típus, a címke és a cím. Ahhoz, hogy az API-tól kapott válasz információi jelenjenek meg a tűkön, az ezeket tároló listán, a getCrime elemein kell végig lépegetni, és az elemek adatait átadni a tű megfelelő tagjának. A tű pozícióját a LONGITUDE és LATITUDE adattagok határozzák meg, a címke az elkövetés dátumából és bűncselekmény megnevezéséből áll, míg a cím az bármiféle módosítás nélkül átadható.

```
public void SetPolygon(double lat, double lng){
knownAdresses.Clear();
```

```
MyCirclePosition = new Xamarin.Forms.Maps.Position(lat, lng);
var getParameter = MyCirclePosition.Latitude.ToString() + ";" +
MyCirclePosition.Longitude.ToString();
var getCrime = GetData(getParameter.Replace('.',','));
PinCollection.Clear();
foreach (var item in getCrime.value){
        if (item.ID != null){
        var upperStart = FirstLetterToUpper(item.CRIME_NAME);
        PinCollection.Add(new Pin() { Position =
        new Xamarin.Forms.Maps.Position(
        double.Parse(item.LATITUDE), double.Parse(item.LONGITUDE)),
        Type = PinType.Generic, Label = item.CRIME_DATE + " - " +
        upperStart, Address = item.ADDRESS });
        } }}
```

Mostanra minden pozíció beállításra került és minden térképelem létre lett hozva. Ahhoz, hogy ezek meg is jelenjenek a térképen, a már említett BindableMap osztályra lesz szükség. Ugyanúgy, mint a térkép pozíciójának változásakor, a tűket tároló lista és a kör középpontjának a változásakor is értesül a BindableMap osztály hozzá tartozó property-je. Az előbbinél, a MapPinsProperty értesül a módosításról, majd az e.NewItems-ben található új tűket, egyesével hozzáadja a térképhez, egy foreach segítségével. A kódban bindable változó az maga a térkép.

```
foreach (var item in e.NewItems)
bindable.Pins.Add((Pin)item);
```

Az utóbbinál a MapCirclePositionProperty-n belül, módosítás hatására, először törölni kell a már lent lévő kört, majd egy újat létrehozni. A térkép összes eleme törölhető, hiszen ezek a tűk és a kör, amelyeket minden MapClicked esemény után újra el kell helyezni. A kör tulajdonságai közül egy sem igényel különösebb magyarázatot, a Center a kör középpontja, ezesetben a térkép érintésének a helye, Radius a sugár, amely 250 méterre lett beállítva, StrokeColor a szegély színe, FillColor a kitöltés színe, a StrokeWidth pedig a vonalvastagság.

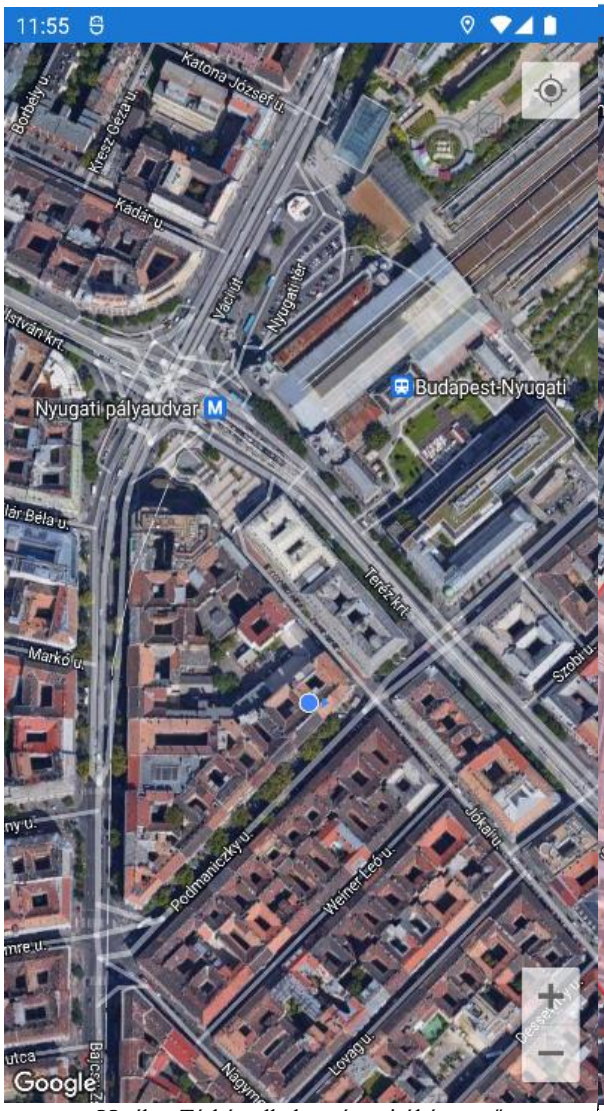
```
((BindableMap)b).MapElements.Clear();
((BindableMap)b).MapElements.Add(new Circle{
Center = (Position)n,
Radius = new Distance(250),
StrokeColor = Color.FromHex("#88FF0000"),
StrokeWidth = 8,
FillColor = Color.FromHex("#88FFC0CB")});
```

7.7. Alkalmazás működés közben, tesztelés

Tesztelés során fontos, hogy az applikáció a helyes paramétereket állítsa össze és erre valóban azokat a válaszokat adja, amelyek valósak. Az alkalmazás felállás után az 33. ábrán látható képernyővel fogad. A telefon pozíciójára fókuszál és kijelzi azt. Az 34. ábrán a térkép érintése utáni állapot látszódik. Az érintés helyén megjelent a kör és egy kis várakozás után a bűncselekmények szimbólumai is megjelentek. Az adatai, a tűre bökésnél jelennek meg. A kijelölt bűneset adatai:

ID: 38137, Crime name: tulajdon elleni szabálysértés, Crime address: Budapest VI. ker. 06. ker, Nyugati út, Crime date: 2022.03.18, Longitude: 19.056739, Latitude: 47.511592

A megnevezés, a dátum és a cím is jó, emellett a koordináták is jó helyre mutatnak így a teszt sikeres.[47]

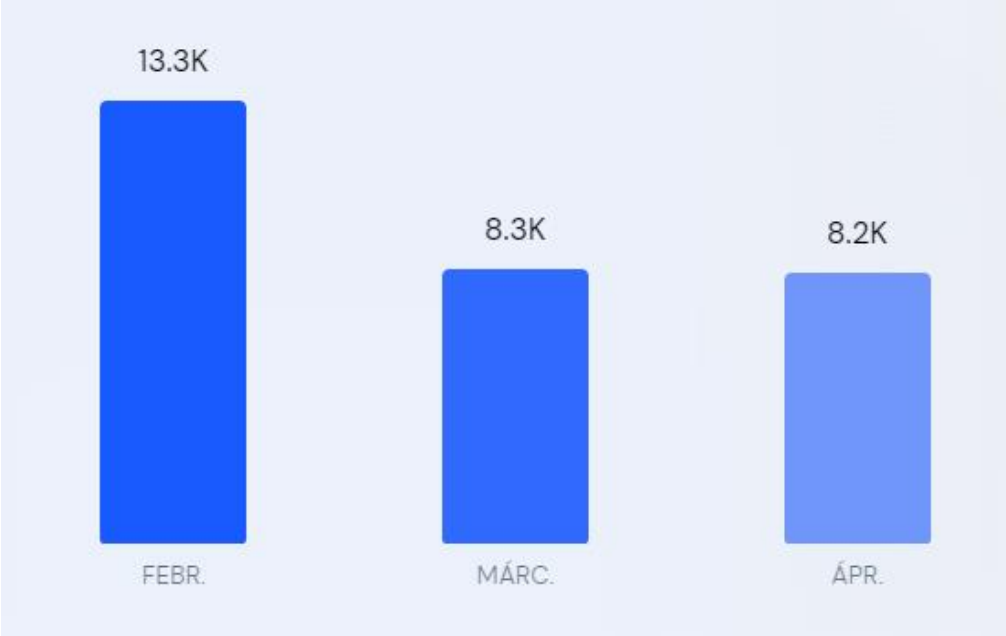


33. ábra Térkép alkalmazás nyitóképernyő

34. ábra Térkép alkalmazás érintés után

7.8. Várható látogatottság

Ahogy az 35. ábrán látható a rendőrség bűnügyi térképét, átlagosan 9000-szer látogatják havi szinten. Az alkalmazás publikálása esetén, hasonló igénybevételre lehet számítani.[48]



35. ábra Rendőrség Bűnügyi térképének látogatottsága

8. Összefoglalás

A szakdolgozat célja egy olyan telefonos applikáció elkészítése volt, ami hordozhatóbbá és hozzáférhetőbbé teszi a rendőrség bűnügyi térképét. Ehhez először a hallgató felvette a kapcsolatot a rendőrséggel, hogy meggyőződjön az ötlet legalitásáról, majd elkezdte tanulmányozni az API végpontjukat. Miután meggyőződött az ötlet megvalósíthatóságáról, kezdődött a kutatás.

Kutatás során górcső alá került a jelenleg népszerű telefonos applikációk kialakítása, a rendőrség bűnügyi térképének története, és az általuk biztosított végpont tulajdonságai. A saját rendszerhez érve, vizsgálatra kerültek a különböző megvalósítási módok, mint szükséges-e a szerver vagy elég a mobil alkalmazás, érdemes-e felhőbe kiköltöztetni a szervert vagy kielégítő megoldás a saját szerver, illetve a telefonos alkalmazást milyen nyelven érdemes írni. A kutatás sikeres volt, informált döntést sikerült hozni a rendszer minden komponenséről, megvalósítás során a szakdolgozat író nem sokban tért el az architektúratervben felvázolt rendszertől. Javában az anyagi vonzat és a rendszerekkel kapcsolatos, egyetemen már megszerzett tudás mennyisége alapján történt a választás.

A saját végpontot biztosító szerver programkódjával kezdődött a fejlesztés. Itt a hallgató betekintést nyert, egy nagyobb program kialakításának sajátosságaira és nehézségeibe. A tervezési fázis során pár át nem gondolt elem, mint a beérkező API hívás paraméterei vagy az adatbázis adattagjainak típusa, komoly utómunkát és módosításokat igényeltek ezzel rávilágítva a fejlesztés ezen részének kiemelkedő fontosságára. Az OData használata, valóban megkönnyítette a munkát, szerencsés, hogy a rendőrség rendszere felvetette ennek a használatának a lehetőségét. Legfontosabb kikötés az autonóm működés volt, ezt a Timer és logolás használatával könnyű volt teljesíteni.

A telefonos applikáció programozása volt a szakdolgozat legnagyobb kihívása. A Xamarin csomaggal való ismerkedés, mint a térkép integrálása, egyes oldalak hívási lánca, jogosultságok kérése és az alakzatok kezelése, rengeteg időt felemésztett. Azonban meg kell jegyezni, hogy a megoszló vélemények ellenére, tökéletesen alkalmas egyszerűbb programok elkészítésére.

8.1. Tovább fejlesztési lehetőségek

A rendszer ellátja a feladatát, lekéri bűneseteket a rendőrségtől majd ezeket meg is jeleníti a felhasználó számára, de emellett még van tér továbbfejlesztési lehetőségeknek.

Az első ötlet amivel a hallgató elkezdett foglalkozni, de idő hiányában be kellett vele fejeznie a munkát, az utca kereső. Az alkalmazás kezelését jelentősen megkönnyítené, ha egy keresővel lenne ellátva, amelybe beleírva az utcanevet a térkép rögtön odanavigál, így megspórolva az ujjal való mozgatást.

A következő lehetőség egy útvonal tervező implementálása. Egy Google Maps-ben található útvonaltervezőt lehetne implementálni, azzal a lehetőséggel, hogy a biztonságosabb utat választja. A rendszer összegyűjti az összes lehetséges útvonalat, majd leellenőrzi hány bűnesetet érintene, majd szerint a szám szerint rangsorol. A felhasználó választhatna ezek közül.

A kör méretének állítása szintén felmerült. A kör szegélyének huzásával, állítható lenne annak sugara. A szerveren futó programban ehhez kis módosítást kellene eszközölni. Paraméterként a koordináták mellett a kör sugara is szerepelhetne, majd az adatbázisban való lekérdezéskor, ez alapján a szám alapján keresné a büneseteket az előre megadott érték helyett.

Egy új menü hozzáadásával, lehetne váltogatni a térkép típusát, hiszen vannak akiknek a szatelit kép teljesen felesleges, csak a lényegre kíváncsiak, így az utcanézet számukra a legalkalmasabb.

Végeredményben, a hallgató elérte a szakdolgozata célját, megalkotta rendőrség bűnügyi térképének, telefonra írt változatát.

8.2. Summary

The aim of the dissertation was to create a mobile application that makes the criminal map of the police, more portable and accessible. To achive this, the student first contacted the police to make sure the idea was legal and after their positive answer, he began to study their API endpoint. After being convinced of the feasibility of the idea, the research began.

The research was focused on the development of the currently popular mobile applications' architecture, the history of the police criminal map, and the characteristics of the endpoint they provide. Regarding the own system, different implementation methods were examined, such as whether a server is necessary or the mobile application can work alone, without any outside help, or whether it is worth to move the server's program to the cloud or it can say on a bare-metal machine. The research was successful, an informed decision was made about all the components of the system. During the implementation of the dissertation's programs, the author have not deviated much from the outlined architectural plan. In general, the choices were made on the basis of the financial implications and the amount of already gathered knowledge from the university.

The development began with the program code for the server that provided its own endpoint. Here, the student gained insight into the specifics and difficulties of designing a larger program. During the design phase, a few unexpected elements, such as the problems with the parameters of the incoming API call or the type of data in the database, required serious rework and modifications, highlighting the paramount importance of this part of the development. The use of OData has really made the job easier. It is very fortunate that, thanks to the police's system the author found it. The most important stipulation was the capability of autonomous operation, which was easy to accomplish using the Timer and logging.

The programming of the telephone application was the biggest challenge of the dissertation. Getting to know the Xamarin package, like integrating the map, the calling chain of some pages, requesting permissions, and managing shapes, consumed a lot of time. However, it should be noted that despite the divergent opinions, it is perfectly suited for making simpler programs.

Eventually, the student achieved the goal of his dissertation, creating a mobile version of the police crime map.

9. Ábrajegyzék

10.Forrásjegyzék

[1] Uber Google Maps API használata

(https://www.cnbc.com/2019/04/11/uber-paid-google-58-million-over-three-years-for-map-services.html),

utoljára megtekintve: 2021. 10. 14.

[2] Google Maps API árazás

(https://developers.google.com/maps/billing/gmp-billing),

utoljára megtekintve: 2021. 10. 17.

[3] Nginx - Load Balancer leírás

(https://www.nginx.com/resources/glossary/load-balancing/),

utoljára megtekintve: 2021. 10. 06.

[4] Uber architektúra

(https://www.geeksforgeeks.org/system-design-of-uber-app-uber-system-architecture/),

utoljára megtekintve: 2021. 10. 30.

[5] Waze mikroszervíz architektúra

(https://cloud.google.com/blog/products/gcp/how-waze-tames-production-chaos-using-spinnaker-managed-pipeline-templates-and-infrastructure-as-code),

utoljára megtekintve: 2021. 10. 10.

[6] Mobil operációs rendszerek piaci sorrendje

(https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/),

utoljára megtekintve: 2021. 11. 03.

[7] 2002-es Bűnügyi adatgyűjtés

(https://www.origo.hu/itthon/20021025bunugyi.html),

utoljára megtekintve: 2021. 12. 03.

[8] 2002-es internet használat adatok

(https://www.origo.hu/techbazis/200212192002.html),

utoljára megtekintve: 2021. 10. 09.

[9] Bűnügyi térkép megjelenése

(https://www.mfttt.hu/mftttportal/index.php/hireink/20-hir/141-bnugyi-terkep-mindenkinek),

utoljára megtekintve: 2021. 10. 09.

[10] RESTful webszolgáltatás

(https://developer.ibm.com/articles/ws-restful/),

utoljára megtekintve: 2021. 10. 21.

[11] ODATA leírás

https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata),

utoljára megtekintve: 2022. 03. 24.

[12] Fizikai szerver és felhő összehasonlítás

https://www.morefield.com/blog/on-premises-vs-cloud/),

utoljára megtekintve: 2021. 11. 09.

[13] NIST modell

https://csrc.nist.gov/publications/detail/sp/800-145/final),

utoljára megtekintve: 2021. 11. 09.

[14] AWS tulajdonságok

https://aws.amazon.com/ec2/sla/historical/#:~:text=AWS%20will%20use%20commercially%20reasonable%20efforts%20to%20make%20Amazon%20EC2,the%20%E2%80%9CService%20Commitment%E2%80%9D),

utoljára megtekintve: 2021. 11. 22.

[15] Szomszédsági támadás

https://www.researchgate.net/figure/Neighbor-attacks-between-virtual-machines-VMs_fig19_305310582),

utoljára megtekintve: 2021. 10. 27.

[16] AWS árazás

https://aws.amazon.com/ec2/pricing/on-demand/),

utoljára megtekintve: 2021. 11. 21.

[17] Hypervisor leírás

https://www.vmware.com/topics/glossary/content/hypervisor#:~:text=A%20hypervisor%2C%20also%20known%20as,such%20as%20memory%20and%20processing),

utoljára megtekintve: 2021. 10. 01.

[18] Windows Server rendszer követelmények

https://docs.microsoft.com/en-us/windows-server/get-started/hardware-requirements),

utoljára megtekintve: 2021. 10. 23.

[19] Windows termékek listája

https://www.microsoft.com/en-us/evalcenter/evaluate-hyper-v-server-2019),

utoljára megtekintve: 2021. 10. 23.

[20] CentOS rendszer követelmények

https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux/centos-migration?options=production-environments&sc_cid=7013a000002psi2AAA&gclid=Cj0KCQiA15yNBhDTARIsAGnwe0V_lkJ7x-dRr9LkpenPTnVthda7wMinJOCFlrSd_MHj1qKdDBEHCM4aAuZ2EALw_wcB&gclsrc=aw.ds),

utoljára megtekintve: 2021. 10. 23.

[21] CentOS End of Life

(https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux/centos-migration?options=production-environments&sc_cid=7013a000002psi2AAA&gclid=Cj0KCQiA15yNBhDTARIsAGnwe0V_lkJ7x-dRr9LkpenPTnVthda7wMinJOCFlrSd_MHj1qKdDBEHCM4aAuZ2EALw_wcB&gclsrc=aw.ds),

utoljára megtekintve: 2021. 10. 23.

[22] Ubuntu Server rendszer követelmények

(https://help.ubuntu.com/community/Installation/SystemRequirements),

utoljára megtekintve: 2021. 11. 12.

[23] Legnépszerűbb Webszerver alkalmazások

(https://digitalintheround.com/what-is-the-most-popular-web-server/),

utoljára megtekintve: 2021. 11. 30.

[24] Nginx és Apache összehasonlítás

(https://serverguy.com/comparison/apache-vs-nginx/#:~:text=The%20main%20difference%20between%20Apache,multiple%20requests%20within%20one%20thread.),

utoljára megtekintve: 2022. 03. 01.

[25] 2020-as bűnügyi statisztikák

(https://bsr.bm.hu/),

utoljára megtekintve: 2022. 02. 16.

[26] SQL – NonSQL összehasonlítás

(https://www.thorntech.com/sql-vs-nosql/),

utoljára megtekintve: 2021. 10. 09.

[27] SQL – NonSQL összehasonlítás

(https://www.educba.com/mysql-vs-nosql/),

utoljára megtekintve: 2021. 09. 30.

[28] Service – based adatbázis létrehozása

(https://docs.microsoft.com/en-us/visualstudio/data-tools/create-a-sql-database-by-using-a-designer?view=vs-2022),

utoljára megtekintve: 2021. 09. 30.

[29] ADO.NET Entity Framework

(https://www.entityframeworktutorial.net/entityframework6/create-entity-data-model.aspx),

utoljára megtekintve: 2021. 10. 09.

[30] ODATA implementálás

(https://www.odata.org/blog/how-to-use-web-api-odata-to-build-an-odata-v4-service-without-entity-framework/),

utoljára megtekintve: 2022. 03. 20.

[31] Google Maps API kulcs igénylése

(https://developers.google.com/maps/documentation/maps-static/get-api-key),

utoljára megtekintve: 2022. 05. 02.

[32] Android Operációs Rendszer

(https://www.investopedia.com/terms/a/android-operating-system.asp),

utoljára megtekintve: 2021. 10. 09.

[33] Android népszerűségének okai

(https://www.eyerys.com/articles/too-many-android-phones-market-here-are-reasons),

utoljára megtekintve: 2021. 10. 09.

[34] Androidra való fejlesztés irányelvei

(https://play.google.com/about/developer-content-policy/#!),

utoljára megtekintve: 2022. 04. 30.

[35] Publikálás Androidra

(https://www.altexsoft.com/blog/engineering/pros-and-cons-of-android-app-.development/),

utoljára megtekintve: 2022. 05. 06.

[36] Android Operációs Rendszerrel rendelkező telefonok

(https://qz.com/472767/there-are-now-more-than-24000-different-android-devices/),

utoljára megtekintve: 2021. 11. 05.

[37] IOS Operációs Rendszerrel rendelkező telefonok

(https://www.theverge.com/2020/6/22/21299686/apple-ios-14-ipados-macos-big-sur-watchos-7-compatibility-update-new-software-wwdc-2020),

utoljára megtekintve: 2021. 11. 09.

[38] Xamain Google Maps integráció

(https://docs.microsoft.com/en-us/xamarin/android/platform/maps-and-location/maps/maps-api),

utoljára megtekintve: 2022. 04. 22.

[39] Xamarin form elhelyezés

(https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/map/polygons),

utoljára megtekintve: 2022. 04. 22.

[40] Xamarin Marker elhelyezés

(https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/map/pins),

utoljára megtekintve: 2022. 04. 22.

[41] Jim Bennett : Xamarin in Action: Creating Native Cross-platform Mobile Apps. Manning Publications. 2018

(https://developers.google.com/maps/documentation/android-sdk/controls),

utoljára megtekintve: 2022. 04. 01.

[42] Xamarin problémái

(https://medium.com/@kibotu/everything-that-is-wrong-with-xamarin-and-why-it-is-bad-for-you-a5399075e50a),

utoljára megtekintve: 2021. 10. 03.

[43] Xamarin problémái

(https://betterprogramming.pub/xamarin-forms-is-dying-and-maui-may-not-be-future-f0395575ac5d),

utoljára megtekintve: 2021. 10. 05.

[44] Xamarin problémái

(https://foresightmobile.com/blog/2020/09/15/isxamarindead),

utoljára megtekintve: 2021. 10. 04.

[45] Xamarin problémái

(https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/),

utoljára megtekintve: 2021. 11. 19.

[46] Koordináták különbségének méterre váltása

(https://www.usna.edu/Users/oceano/pguth/md_help/html/approx_equivalents.htm), utoljára megtekintve: 2021. 04. 15.

[47] Telefonos applikáció koordinátáinak tesztelése

(https://www.google.com/search?q=47.511592+19.056739+&sxsrf=ALiCzsb8eFzFtvpyV8WAhqAn3iQva4AJeQ%3A1652348400288&ei=8NV8YuyWEY2csAfTlqnIDA&ved=0ahUKEwjszsa71dn3AhUNDuwKHVNLCskQ4dUDCA4&uact=5&oq=47.511592+19.056739+&gs_lcp=Cgdnd3Mtd2l6EAM6BAgjECdKBAhBGAFKBAhGGABQuARYghJg9hVoAXAAeACAAWSIAcYCkgEDMy4xmAEAoAEBoAECwAEB&sclient=gws-wiz),

utoljára megtekintve: 2022. 05. 06.

[48] Rendőrség bűnűgyi térképének látogatottsági adatai

(https://www.similarweb.com/website/terkep.police.hu/#traffic),

utoljára megtekintve: 2022. 05. 01.

[49] Telefonos alkalmazás által kérhető jogosultságok

(https://developer.android.com/reference/android/Manifest.permission),

utoljára megtekintve: 2022. 05. 05.

[50] Google Maps API kulcs árazása

(https://mapsplatform.google.com/pricing),

utoljára megtekintve: 2022. 05. 05.