Usporedba razvoja aplikacija za Android pomoću Jave i Xamarina

Fiorencis, Marta

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:710447

Rights / Prava: Attribution-NonCommercial-NoDerivs 3.0 Unported/Imenovanje-Nekomercijalno-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-05-05**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marta Fiorencis

USPOREDBA RAZVOJA APLIKACIJA ZA ANDROID POMOĆU JAVE I XAMARINA

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marta Fiorencis

Matični broj: 42010/13–R

Studij: Informacijsko i programsko inženjerstvo

USPOREDBA RAZVOJA APLIKACIJA ZA ANDROID POMOĆU JAVE I XAMARINA

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, rujan 2022.

Marta Fiorencis

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U diplomskom radu je predstavljen i uspoređen proces razvoja Android aplikacije pomoću programskih jezika Jave i C# u razvojnim okruženjima Android Studio (Java) i Visual Studio (C#). Kako bi usporedba bila što objektivnija paralelno su kreirani i uspoređivani koraci razvoja aplikacije u oba razvojna okruženja, bitno je napomenuti da obje aplikacije imaju iste funkcionalnosti kako bi krajnji zaključak mogao biti što detaljniji. Kreirane aplikacije pokrenute su na dostupnim emulatorima i fizičkim mobilnim uređajima. Nakon svih koraka razvoja definirani su kriteriji usporedbe za korištena razvojna okruženja odnosno programske jezike.

Ključne riječi: Java, C#, Android aplikacija, Visual Studio, Android Studio, Xamarin.

Sadržaj

1. Uvod

Tema samog rada je usporedba razvoja aplikacije u razvojnom okruženju Android Studio koristeći programski jezik Java i u razvojnom okruženju Visual Studio koristeći Xamarin.Android alate i C# programski jezik. U radu bit će detaljno paralelno predstavljeni svi koraci razvoja u oba slučaja kako bi čitatelj mogao što jasnije razaznati sličnosti i razlike. Motivaciju za ovaj rad dobila sam kroz diplomski studij jer prije diplomskog studija nisam bila toliko upoznata s programskim jezikom Java i izbjegavala sam Android Studio najviše zbog negativnih stvari koje sam čula ironično od ljudi koji ga također nisu koristili ili su ga koristili vrlo malo. Nakon što sam ga bila prisiljena koristiti tijekom studija shvatila sam da su moji strahovi bili neopravdani i uživala sam u edukaciji, pogotovo jer sam bila zainteresirana za razvoj mobilnih aplikacija. Tema mog završnog rada bila je kreiranje mobilnih aplikacija u Xamarin-u i nakon malo istraživanja primijetila sam da je puno ljudi uspoređivalo Xamarin i Android Studio, samo što kod tih usporedba nisu su se uspoređivali Xamarin.Android alati već Xamarin.Forms alati što po mojem mišljenju nije baš poštena usporedba jer Android Studio je namijenjen izradi android aplikacija, a Xamarin.Forms alati su namijenjeni izradi hibridnih aplikacija i nisu se previše spominjali Xamarin.Android alati, pogotovo ako je netko postavio pitanje na forumu o usporedbi Xamarin-a i Android Studija. Još jedna stvar koju sam primijetila jest ako nije riječ o stručnim člancima, knjigama ili osobama koje su podjednako iskusne u oba programska jezika većina presuda što je bolje koristiti temelji se na programskom jeziku s kojim je netko upoznat te je drugi izbor degradiran iako ga osoba nije nikad koristila. Jednostavnije rečeno osoba je donijela presudu u korist opcije s kojom je bolje upoznata umjesto da stekne ekvivalentnu količinu znanja i o drugoj opciji te potom donese presudu. Također se u velikoj većini članaka/rasprava čija je tema usporedba Xamarin-a i Android Studija uzimaju u obzir tehničke stvari razvojnih okruženja, a ne toliko osobno iskustvo u kreiranju aplikacija. Upravo zbog toga ovaj rad bit će fokusiran na paralelno kreiranje dvije android aplikacije te će na kraju biti izvršena usporedba s kriterijima kao što su dostupna dokumentacija, emulatori, brzina pokretanja aplikacije, veličina aplikacije, pojava i rješavanje grešaka kroz proces razvoja. Svrha samog rada je prezentirati proces razvoja android aplikacije na dva različita načina iz perspektive osobe kojoj programski jezik i razvojno okruženje nisu ključni kriteriji odabira. Nakon što završi s uvodom čitatelj će započeti drugo poglavlje koje sam namijenila kratkom opisu metoda i tehnika. U trećem poglavlju navela sam razvojna okruženja koja su korištena u radu kao i njihovu kratku povijest, zahtjeve sustava te sam pojasnila samu vizualnu strukturu početnog sučelja. U

četvrtom poglavlju sam detaljno objasnila tehnologije razvoja od kojih su neke korištene u radu ali i neke koje se smatraju osnovama u razvoju mobilnih aplikacija. Peto poglavlje sam rezervirala za objašnjavanje usporednog proces razvoja mobilnih aplikacija i pojašnjavanje tehničkih dijelova kao što su isječci programskih kodova kod implementacija funkcionalnosti. Nakon što sam završila proces razvoja sprovela sam usporedbu prema određenim kriterijima koju čitatelj može naći u šestom poglavlju te potom pročitati do kojeg sam zaključka došla kreirajući ovaj diplomski rad.

2. Metode i tehnike rada

U prvom dijelu rada koji će biti više orijentiran teoriji dostupnoj iz navedene literature predstavit ću razvojna okruženja koja sam koristila u usporednom razvoju aplikacija, u ovom slučaju to su Android Studio i Visual Studio. Pojasnit ću tehnologije razvoja i navesti primjere osnovnih elemenata koji se koriste u izradi mobilnih aplikacija bez obzira na programski jezik kao što su aktivnosti, usluge, prijemnici za emitiranje, pružatelji sadržaja, fragmenti, pogledi, namjere, resursi i manifest datoteke jer će se većina elemenata koristiti u usporednoj implementaciji. Također ću pojasniti najčešće korištene i preporučene arhitekture android aplikacija te uobičajeni proces razvoja koji sam i sama slijedila. Cilj ovog pristupa je objasniti samom čitatelju ovog rada koliko razvoj android aplikacije u javi ima sličnosti s razvojem aplikacije u C# programskom jeziku, te koliko utječe samo planiranje i priprema prije početka same implementacije rješenja.

Drugi dio rada je usporedni razvoj mobilnih aplikacija, aplikacije će koristiti podatke iz iste SQL baze i php skripte koje će podatke dohvaćati ili izmjenjivati. Prije samog pisanja koda u InVision Studio alatu kreirala sam skice korisničkih sučelja prema kojima će se kreirati dizajn samih sučelja u obje aplikacije. Svaki sam element kreiranja programskih rješenja u razvojnim okruženjima usporedno prikazala kako bi čitatelju bilo lakše usporediti sami programski kod. Zbog lakšeg snalaženja čitatelja i preglednosti rada podijelila sam implementaciju programskih rješenja na kreiranje dizajna korisničkih sučelja i implementaciju funkcionalnosti te je kod svake funkcionalnosti prikazan zaseban dijagram klasa aplikacije kreirane u Android Studiju i Visual Studiju kako bi se jasnije vidjele sličnosti i razlike programskih rješenja.

Na temelju usporednog razvoja sprovela sam usporedbu pomoću određenih kriterija te pojasnila mane odnosno prednosti korištenih programskih jezika i razvojnih okruženja.

3. Razvojno okruženje

Razvojno okruženje ili IDE (eng. Integrated Development Environment) je softver čija je svrha olakšati razvoj aplikacija integrirajući više potrebnih alata za razvoj u jedan pomoću kojeg korisnik može jednostavnije izvršiti što više procesa razvoja. Cilj razvojnog okruženja jest da uštedi korisniku što više vremena i truda tijekom razvoja (kao na primjer automatsko prepoznavanje grešaka u kodu ili dostupnost dodatnih resursa). Pošto je razvojno okruženje skup alata spojenih u jedan korisnik se ne mora konstantno prebacivati iz programa u program te brinuti o tome da li su alati međusobno kompatibilni. Također razvojna okruženja imaju mogućnost uključivanja raznih dodatnih biblioteka ili paketa u projekt što može znatno ubrzati razvoj. Razvojna okruženja koja sam koristila za ovaj projekt su Android Studio za razvoj mobilne aplikacije pomoću jave i Visual Studio za razvoj pomoću C#. U nastavku će biti navedene točne verzije okruženja, potrebni zahtjevi sustava kako bi se okruženje moglo pokrenuti, postupak instalacije i kreiranje projekta te opis početnog sučelja.

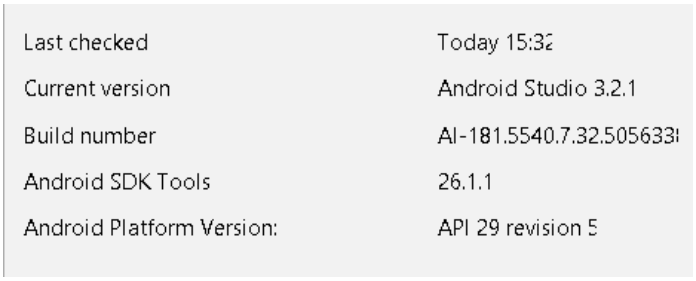
3.1. Android Studio

Android Studio je razvojno okruženje koje se koristi za razvoj Android aplikacija. Kada upišite u google pojam „Android Studio“ i kliknete na prvu poveznicu otvara se službena stranica te prva rečenica koju korisnik vidi jest „Android Studio pruža najbrže alate za izradu aplikacija za sve vrste Android uređaja.“ [1]. Što više dodati takvoj definiciji. Povijest Android Studio započinje u prosincu 2014. godine kada je službeno izdana prva verzija. Jedna od stvari što je pridonijela njegovoj popularnosti jest to što je besplatan (najnovija verzija se može skinuti sa službene stranice https://developer.android.com/studio) ali i učinkovit. Također je dostupan za više operacijskih sustava kao što su Windows, Linux i Mac OS. Kako bi mogli razvijati aplikacije potrebno je instalirati i JDK (Java Development Kit) iliti alate pomoću kojih možemo razvijati, testirati i pokretati programe napisane u Javi. Zbog toga Java i Kotlin su programski jezici pomoću kojih razvijamo aplikacije u Android Studiju. Također kako bi mogli kreirati mobilne aplikacije potreban je i SDK (Android Software Development Kit) skup alata (biblioteke, emulatori, itd.) koji nam omogućuju samo razvijanje aplikacija. Kod odabira SDK verzije treba pripaziti na kompatibilnost s odabranom API (Application Programming interface) verzijom (veći broj API- a znači veći broj mogućnosti i funkcija) [2]. Jednostavno rečeno SDK sadrži alate za izradu aplikacije a API sadrži standarde pomoću kojih aplikacija može komunicirati s drugim alatima ili bazama podataka.

Slika 1: Android Studio – Logo (Izvor: Android Studio, 2022)

3.1.1.Korištena verzija razvojnog okruženja

Za izradu aplikacije čija je zadnja verzija dostupna na GitHub-u [28] koristit ću Android Studio 3.2.1 verziju, SDK 26.1.1 i API 29.



Slika 2: Korištene verzije alata u razvojnom okruženju

3.1.2. Preporučeni zahtjevi sustava

Kako bi se Android Studio mogao uspješno instalirati na Windows operacijskom sustavu i koristiti iznad navedene verzije preporučuju se sljedeći zahtjevi sustava [3]:

- Windows 7/8/10
- Minimalno 4GB RAM-a (preporučuje se 8GB)
- Minimalno 4GB memorije
- Minimalna rezolucija ekrana 1280 x 800

3.1.3. Postupak instalacije

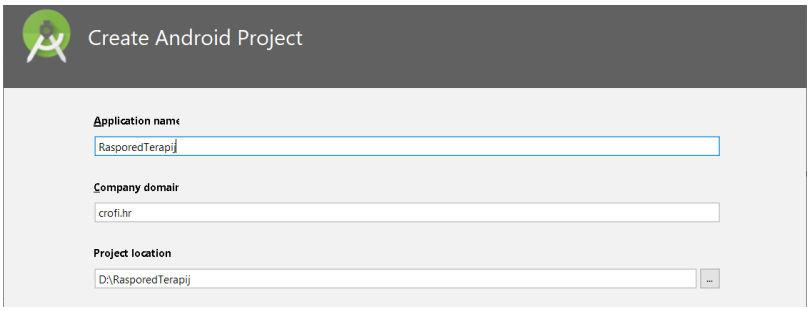
- Skinite datoteku(.zip) koja sadrži željenu verziju sa službene stranice
- Otpakirajte datoteku
- Otvorite .bin datoteku i pokrenite instalaciju (kliknite na studio64.exe ili studio.exe)

- Slijedite upute čarobnjaka te instalirajte sve potrebne SDK pakete (još jednom naglašavam da u ovom koraku mora se pripaziti na izbor verzije SDK alata da ne bi kasnije došlo do nekompatibilnosti)

3.1.4. Kreiranje projekta

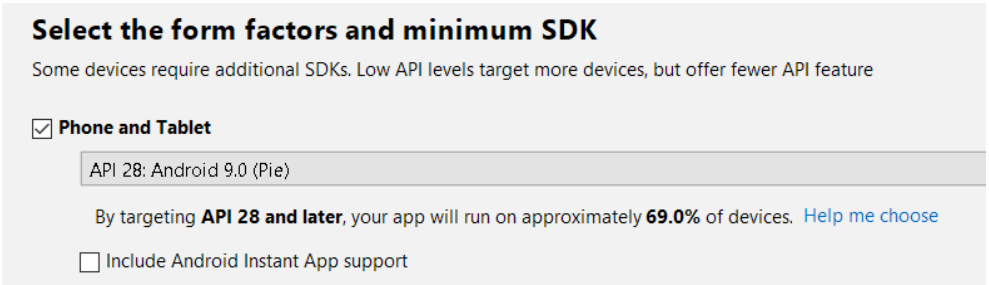
Nakon instalacije kreirat ću projekt sljedeći navedene korake [3]:

- Odaberite File -> New -> New Project te će vam se prozor u kojem upišite Ime aplikacije, ime paketa te lokaciju gdje će projekt biti spremljen, nakon što su podaci uneseni kliknite gumb Next.



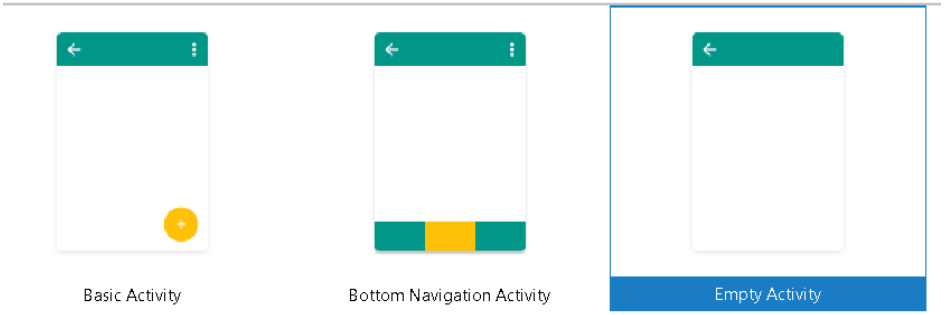
Slika 3: Kreiranje projekta - Android Studio

- Otvorit će vam se prozor u kojem odabirete uređaje za koje je aplikacija namijenjena te minimalni SDK.



Slika 4: Odabir minimalne SDK verzije – Android Studio

- Potom će vam se otvoriti izbor početnih aktivnosti koje možete dodati mobilnoj aplikaciji, u ovom slučaju odabrat ću osnovni oblik aktivnosti (*Empty Activity*).

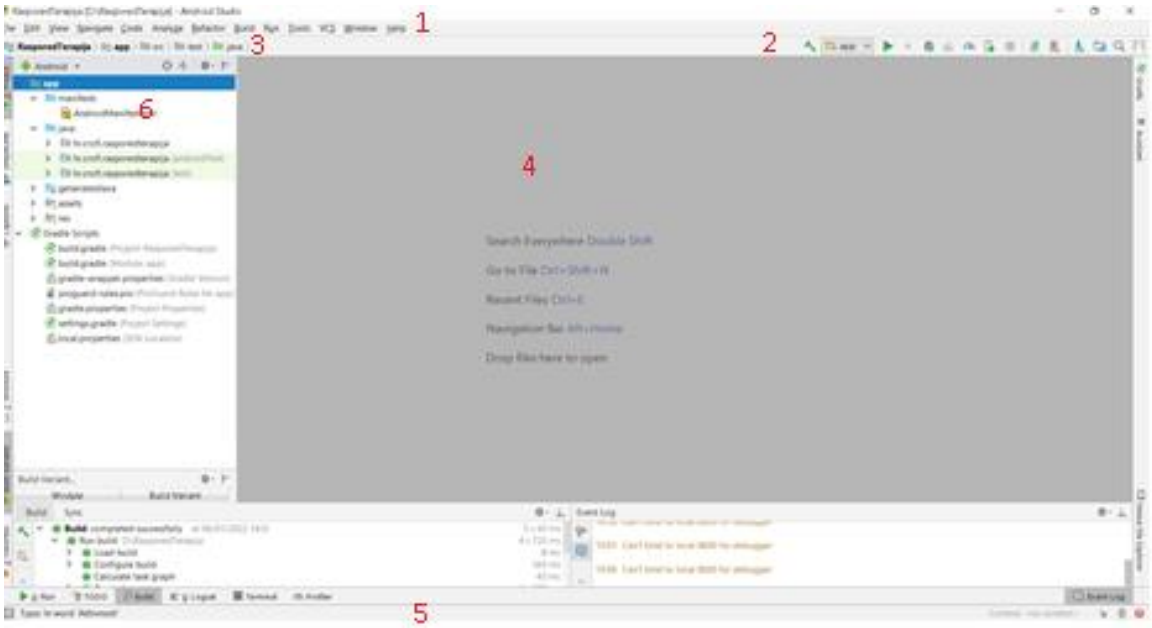


Slika 5: Odabir početne aktivnosti – Android Studio

Ako ste zadovoljni s postavkama kliknite na gumb Finish i projekt će se generirati, u slučaju da imate slabije računalo ovaj korak bi mogao potrajati nekoliko trenutak.

3.1.5. Početno sučelje

Nakon što se projekt uspješno kreirao prikazat će vam se početno sučelje koje ćemo detaljnije pojasniti u nastavku.



Slika 6: Početno sučelje – Android Studio

Kao što se vidi na slici iznad početno sučelje sastoji se od nekoliko osnovnih komponenti [3]:

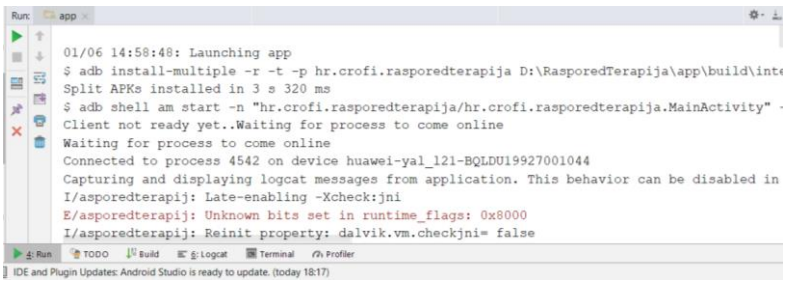
1. Traka s glavnim izbornikom – Na njoj su prikazane dodatni izbornici i funkcije koje korisnik može koristiti u razvojnom okruženju

Slika 7: Glavni izbornik – Android Studio

2. Alatna traka – Ovdje su prikazani prečaci za najčešće korištene funkcionalnosti, korisnik može modificirati alatnu traku po svojim zahtjevima (dodavati ili micati prečace), kao na primjer gumb za pokretanje ili kompajliranje aplikacije.
3. Navigacijska traka – Prikazuje hijerarhiju projekta i put do trenutno otvorenih datoteka ili dokumenata, olakšava korisniku snalaženje u projektu.

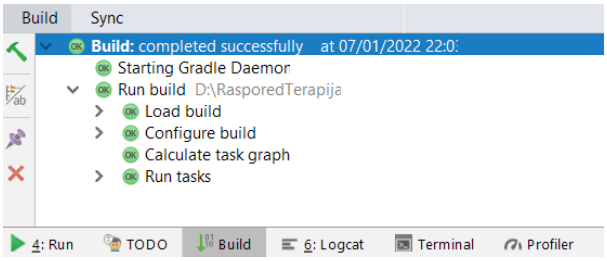
Slika 8: Navigacijska traka – Android Sudio

4. Prozor za uređivanje – Prikazuje kod dokumenata na kojem korisnik trenutno radi, ako korisnik klikne na dokument koji sadrži dizajn aktivnosti u ovom prozoru otvorit će se alat za uređivanje sučelja koji će nam omogućiti modifikaciju dizajna
5. Statusna traka – Obavještava korisnika o stanju projekta ili statusu operacije koja je pokrenuta te se na njoj ispisuju sve poruke ili informacije o projektu (greške, upozorenja, itd.) Naprimjer ako kliknite na Run ili Debug otvara se prozor u kojem će te vidjeti sve zapise sustava tijekom pokretanja i rada aplikacije.



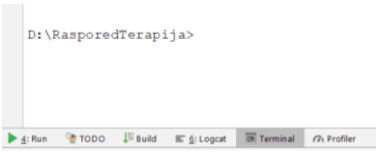
Slika 9: Statusna traka – Android Sudio

Također klikom na Build možete vidjeti kako je prošao proces kompajliranja, ako postoji greška kod Gradle dijela projekta (npr. nedostaje dopuštenje ili referenca vanjsku biblioteku) ovdje će se pojaviti obavijest.



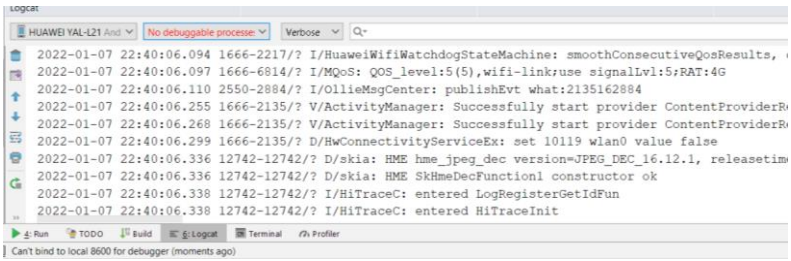
Slika 10: Proces kompajliranja – Android Sudio

Klikom na Terminal otvara se konzolni pristup projektu, odnosno ako želite dodati neki vanjsku biblioteku ili napraviti update postojećih preporučujem iz osobnog iskustva da napravite to ovdje.



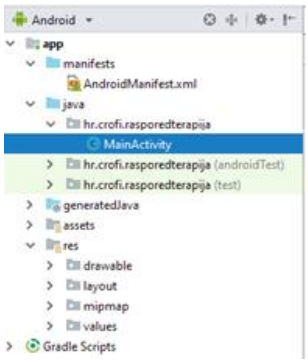
Slika 11: Terminal – Android Sudio

Logcat pruža uvid korisniku u tok i potrebno vrijeme procesa koji se obavljaju tijekom pokretanja aplikacije.



Slika 12: Logcat – Android Sudio

6. Prozor za prikaz strukture – Prikazuje strukturu projekta te omogućuje korisniku snalaženje u projektu.



Slika 13: Prikaz strukture projekta – Android Sudio

Samo početno sučelje je dosta jasno i jednostavno za korištenje, ali ako koristite Android Studio prvi puta preporučujem detaljno proučavanje dokumentacije.

3.2. Visual Studio (Xamarin)

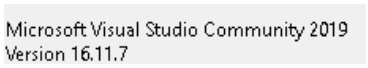
Xamarin-ova povijest započinje u svibnju 2011. i prvotno je bila samostalna tvrtka dok je 2016. nije otkupio Microsoft [4]. Prvotni smisao bio je pružiti razvojno okruženje i alate za razvoj više platformskih (iOS, Android i Windows) aplikacija pomoću .NET, u tom slučaju korisnici koriste Xamarin.forms alate, ali pošto hibridne aplikacije nisu tema ovog rada, dio Xamarin-a koji je bitan su Xamarin.Android alati koji koriste Android SDK (sadrži biblioteke i alate potrebne za razvoj Android aplikacija). Posljednja verzija je Xamarin.Android 12.1, a uskoro će biti i spremna 12.2 verzija [4]. U ovom slučaju Xamarin instalacija izvršava se pomoću razvojnog okruženja Visual Studio i ta verzija Xamarin-a naziva se Xamarin for Visual Studio koja je dostupna za operacijske sustave Windows i Mac OS u ovom slučaju Android Studio ima malu prednost jer je dostupan i za Linux, a ako korisnici Linux operacijskog sustava žele koristiti Xamarin moraju prvo instalirati Java pakete, Mono i Android Studio i tek nakon toga konfigurirati ručno konfigurirati Rider (koji se koristi kao razvojno okruženje za Xamarin) i tek potom instalirati Xamarin tako da je u tom slučaju procedura malo kompliciranija[5]. Ono što je bitno napomenuti kod razvoja Android aplikacija pomoću Xamarin.Android alata jest da programski se piše u C# programskom jeziku te tako omogućuje programerima koji nisu upoznati s Java-om razvijanje mobilnih aplikacija.

Slika 14: Xamarin za Visual Studio – Logo (Izvor: Xamarin, 2022)

3.2.1. Korištena verzija razvojnog okruženja

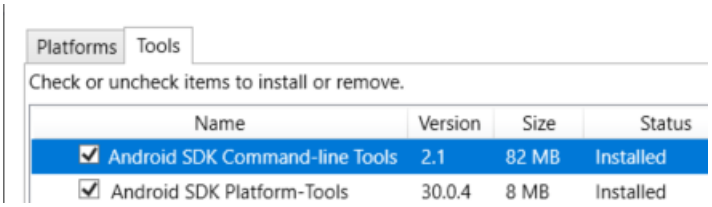
Za izradu aplikacije čiju zadnju verziju možete pronaći na GitHub-u [29] koristila sam:

Microsoft Visual Studio Community 2019, verziju 16.11.7



Slika 15: Visual Studio - verzija (Izvor: Xamarin, 2022)

Xamarin verziju 16.11. i Xamarin.Android alate (SDK) verziju 30.0.4 i API 29



Slika 16: Korištena verzija Xamarin.Android alata

3.2.2. Preporučeni zahtjevi sustava

Ako želite instalirati Xamarin for Visual studio i koristiti *Xamarin.Android* alate potrebno je ispuniti sljedeće zahtjeve sustava[6]:

- Windows 7/810, ali preporučuje se 10
- Minimalno 4GB RAM-a (preporučuje se 8GB)
- Visual Studio 2017 ili 2019 (preporučuje se 2019)
- 20 – 50 GB memorije (Visual Studio uključen)
- Minimalna rezolucija ekrana 1280 x 720

3.2.3. Postupak instalacije

U nastavku će biti pojašnjen postupak instalacije. Ako želite uspješno instalirati Visual Studio pratite sljedeće korake:

- Skinite datoteku koja sadrži željenu verziju Visual Studija sa službene stranice (Visual Studio 2017/2019 Community, Enterprise ili Professional)
- Kliknite na skinuti datoteku i pokrenite instalaciju

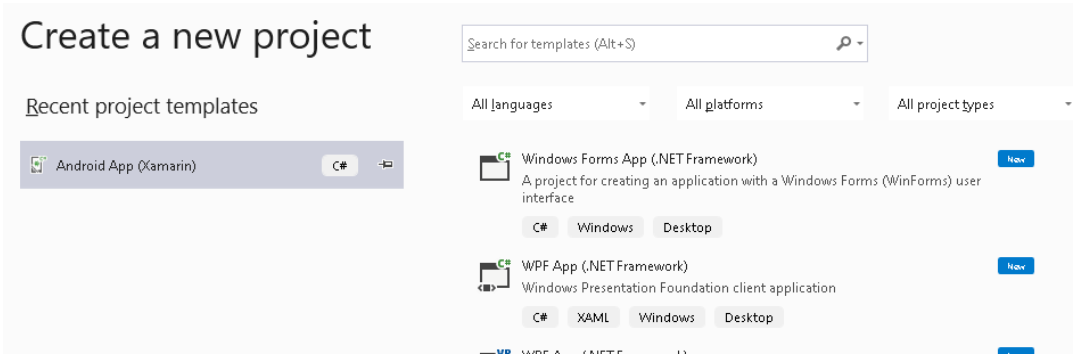
- Nakon što se instalacija pokrenula obvezno odaberite Mobile development with .NET na početnom ekranu i kliknite na install gumb, ovaj korak će automatski instalirati potrebnu verziju JDK-a, ako želite možete promijeniti lokaciju. Korak će također osigurati instalaciju alata Android SDK manager pomoću koji omogućuje podešavanje API verzije aplikacije

Ako je instalacija uspješno izvršena trebala bi se pružiti opcija kreiranja novog projekta.

3.2.4. Kreiranje projekta

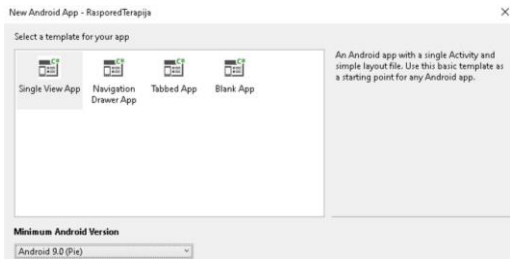
U sljedećem primjeru bit će prikazan postupak kreiranja Android projekta. Slijedite sljedeće korake za uspješno kreiranje projekta:

1. Odaberite File -> New Project te će vam se otvoriti sljedeći prozor



Slika 17: Kreiranje projekta - Xamarin

2. U sljedećem prozoru definira se ime projekta, lokacija gdje će projekt biti spremljen te kontejner(Solution) projekta, ako imate više povezanih projekata možete ih staviti u isti kontejner
3. Prije nego što se projekt generira otvara se prozor u kojem se izabire početna aktivnost aplikacije i minimalna Android verzija. Kada ste zadovoljni s izborom kliknite na gumb ok i projekt će se generirati

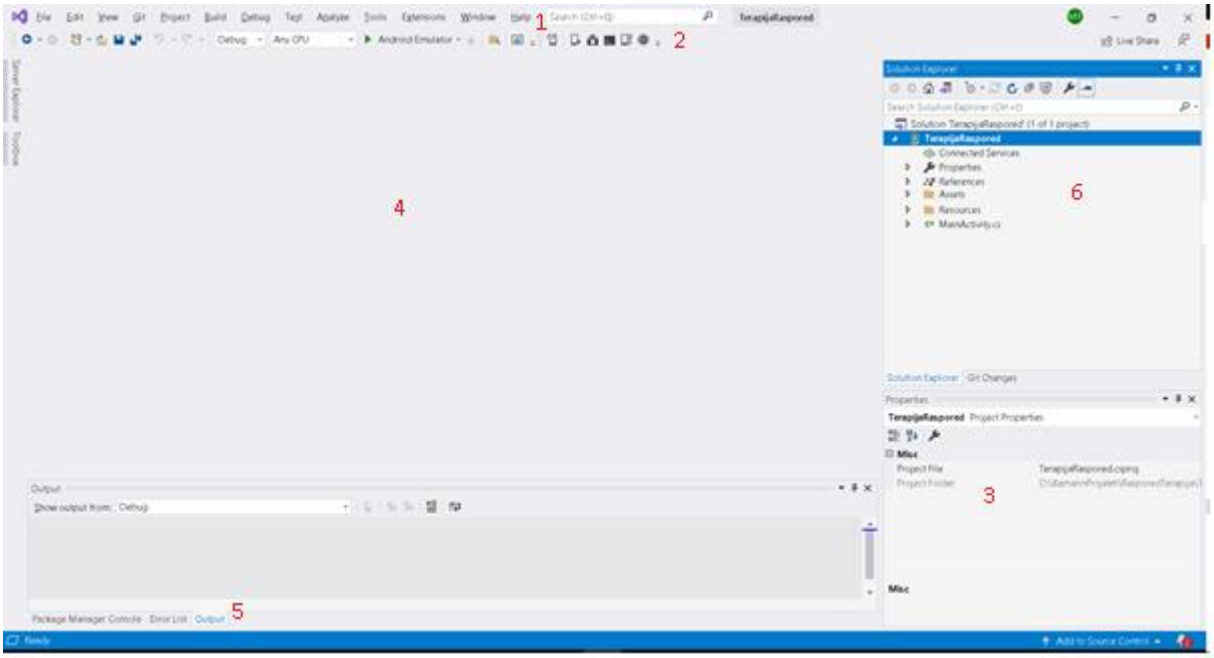


Slika 18:Odabir minimalne Android verzije – Xamarin

Ako je projekt kreiran prikazat će se početno sučelje.

3.2.5. Početno sučelje

Kada je aplikacija uspješno kreirana prikazat će vam se početno sučelje koje se sastoji od nekoliko osnovnih komponenata [7].

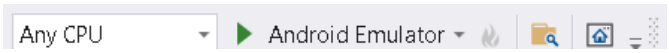


Slika 19: Početno sučelje – Xamarin

1. Traka s glavnim izbornikom – Prikazuje glavni izbornik razvojnog okruženja koji sadrži tematski strukturirane opcije koje se prikazuju ako kliknete na neki od njih. Jedan od važnijih skupa jest „Tools“ pomoću kojih se može koristiti dodatne alate u projektu kao što su na primjer pristup opcije za pristup serveru ili bazi podataka

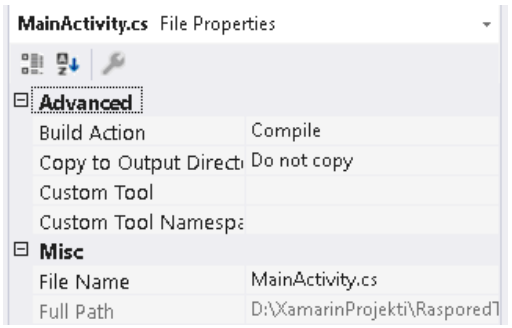
Slika 20: Glavni izbornik - Xamarin

Alatna traka – Prikazuje alate koji omogućuju između ostalog pokretanje i kompajliranje aplikacije, također prikazuje vrstu uređaja ili emulatora na kojem se aplikacija pokreće



Slika 21: Alatna traka - Xamarin

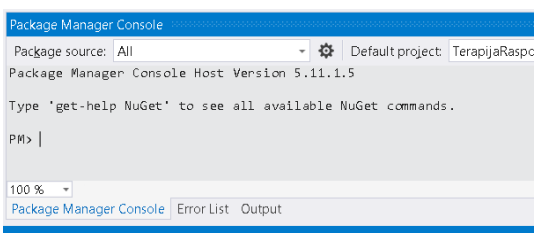
2. Svojstva – U prozor su prikazana svojstva otvorene datoteke u projektu, kao što su ime ili putanja datoteke. Također ima opciju prikaza svojstva abecednim redom ili tematskom kategorizacijom



Slika 22; Prozor za prikaz svojstva projekta – Xamarin

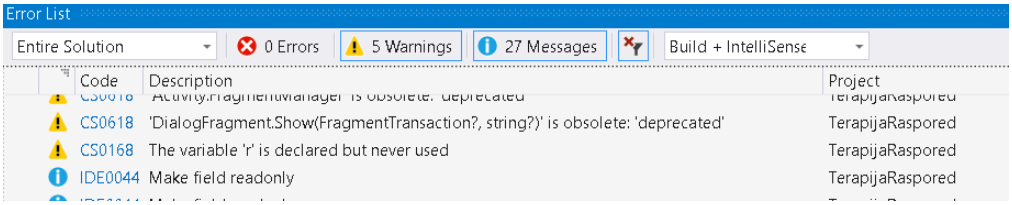
3. Prozor za uređivanje – Prikazuje se kod trenutno otvorene datoteke i omogućuje korisnik daljnji rad. Ako je korisnik otvorio datoteku koja sadrži dizajn korisničkog sučelja, u ovom prozoru će se usporedno prikazati kod i izgled dizajna.
4. Statusni prozor – Omogućuje korištenje alata za praćenje razvoja aplikacije, koji također olakšavaju otkrivanje i popravljanje greška kao što su Packager Manager Console, Error List ili Output

   Packager Manager Console otvara prozor u kojem možemo ručno upravljati s paketima i bibliotekama u projektu, ovo je dobar pristup ako imate problem sa uvozom nekog paketa ili neki paket stalno javlja grešku(najčešće su paketi vezani uz dizajn). Također moram napomenuti da većinu grešaka rješava naredba „dotnet restore“ pomoću koje možemo reinstalirati sve pakete dodane u projekt.



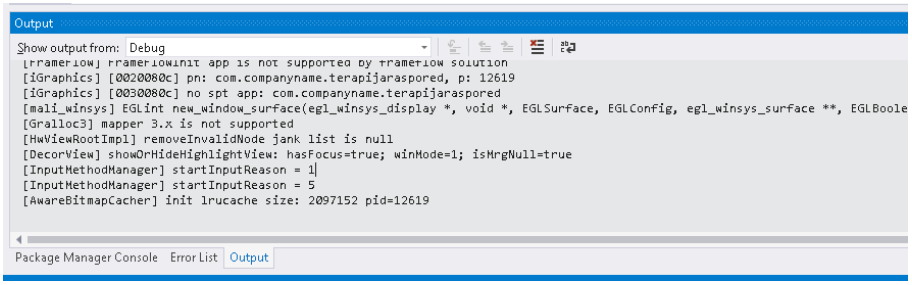
Slika 23: Packager Manager Console – Xamarin

Error List prikazuje sve pogreške, upozorenja ili poruke projekta. Korisnik može regulirati obavijesti klikom na vrstu obavijesti koje želi prikazivati, na primjer ako želi prikazivati samo pogreške može isključiti poruke i upozorenja.



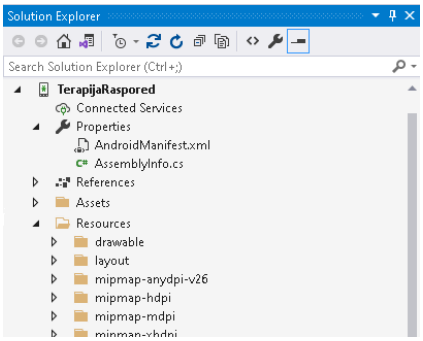
Slika 24: Prozor za prikaz obavijesti – Xamarin

Output prozor prikazuje sve zapise sustava tijekom pokretanja i rada aplikacije (na emulatoru ili priključenom uređaju).



Slika 25: Output prozor – Xamarin

5. Prozor za prikaz strukture (Solution Explorer) – Prikazuje strukturu trenutno otvorenog projekta kako bi korisniku olakšao snalaženje.



Slika 26: Prozor za prikaz strukture projekta – Xamarin

Ako ste početnik, nakon otvaranja početnog sučelja preporučujem kreiranje jednostavnog projekta (Hello World!) kako bi jednostavnije shvatili kako se koristiti sa sučeljem.

4. Tehnologije razvoja

U ovom poglavlju proći ću kroz osnovne elemente razvoja kao što su rad s elementima korisničkog sučelja, poruke korisnicima, rad s lokalnim podacima, korištenje web servisa i vanjskih biblioteka, arhitektura aplikacije te proces razvoja. Ali prvo ću se dotaknuti osnovnih komponenta android aplikacije (koji se ne mijenjaju bez obzira na programski jezik i razvojno okruženje u kojem je aplikacija kreirana) te ih objasniti, a u kasnijim poglavljima bit će uz primjere pojašnjeno kako se te komponente uklapaju u aplikaciju, odnosno kako ih možemo primijeniti kod kreiranja aplikacije.

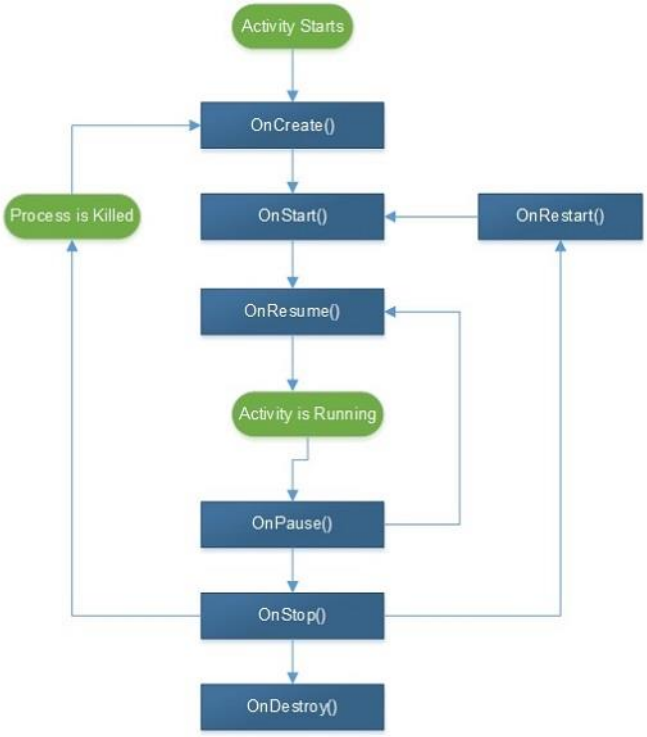
4.1. Osnovne komponente

Kako bi najjasnije objasnila način rada android aplikacije pojasnit ću njezine komponente te način na koji su međusobno povezane. Komponente aplikacije možemo smatrati kao komade slagalice koje kada se pravilno slože prikazuju finalnu sliku odnosno gotovu aplikaciju. Ovdje neće biti podjele između android aplikacija pisanih s Java ili C# programskim jezikom jer u oba slučaja osnovne komponente aplikacija su iste.

4.1.1. Aktivnosti

Komponente koje upravljaju s grafičkim sučeljem aplikacije i detektiraju interakciju korisnika s aplikacijom, nakon što je interakcija detektirana mogu pokrenuti neku protuakciju koja će biti posljedica određene interakcije korisnika (npr. ako korisnik klikne gumb na sučelju, pojavit će se prozor s određenom porukom). Aktivnost (eng. Activities) se može nalaziti u raznim stadijima[9]:

- onCreate() – prvi stadij kada se aktivnost tek kreira
- onStart() – drugi stadij kada je aktivnost kreirana i korisnik može vidjeti sučelje i ostvariti interakciju
- onResume() – stadij kada korisnik ostvaruje interakciju
- onPause() – stadij u kojem aktivnost ne registrira interakciju korisnika sa sučeljem i ne može pokrenuti neku protuakciju
- onStop() – stadij kada aktivnost više nije vidljiva korisniku
- onDestroy() – stadij prije nego je aktivnost uništena
- onRestart() – stadij prije nego se aktivnost ponovno pokreće nakon zaustavljanja



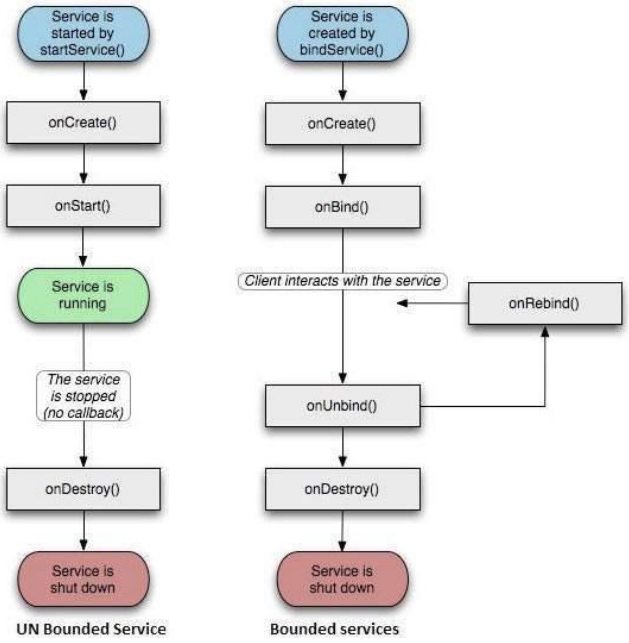
Slika 27: Prikaz veza između stadija aktivnosti [8]

4.1.2. Usluge

Komponenta čiji je cilj obavljanje operacija kojima nije važna interakcija korisnika sa sučeljem i obavljaju se u pozadini aplikacije. Pravilno korištenje usluga (eng. services) utječe na kvalitetu same aplikacije jer ako se glavna dretva aplikacije preoptereti dolazi do sporog rada i zastajkivanja aplikacije [2].

Usluga može poprimiti dva osnovna stadija:

1. Started – stadij u kojem je usluga tek pokrenuta od strane druge komponente naredbom startService()
2. Bound – stadij koji usluga poprima naredbom boundService() kako bi omogućila drugim komponentama slanje zahtjeva i primanje rezultata



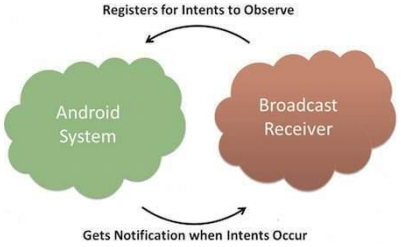
Slika 28: Ciklus izmjenjivanje stadija usluge [9]

4.1.3. Prijemnici za emitiranje

Pružaju mogućnost komuniciranja android aplikacije s drugim aplikacijama, ako aplikacija prima podatke od neke druge aplikacije (npr. trenutnu temperaturu zraka) koristi prijemnik za emitiranje (eng. (Broadcast Receivers)) kako bi dala do znanja drugoj aplikaciji kako su podaci uspješno primljeni.

Životni ciklus prijemnika možemo podijeliti u dva bitna stadija:

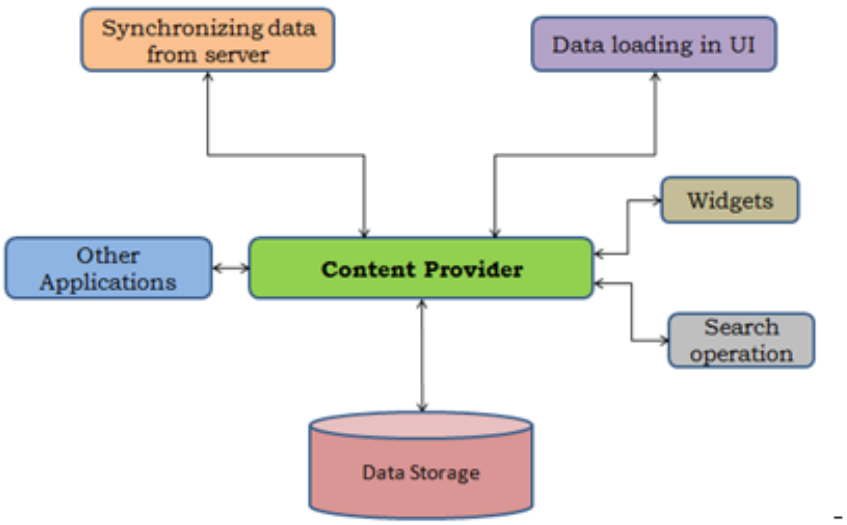
1. Kreiranje prijemnika – stadij u kojem se definira akcija koja se događa nakon što su podaci od druge aplikacije uspješno primljeni (akcija se definira u onReceive() metodi)
2. Registracija prijemnika – definiranje događaja koji će okinuti primanje podataka u prijemniku



Slika 29: Životni ciklus prijemnika [9]

4.1.4. Pružatelji sadržaja

Komuniciraju sa slojem podataka u aplikaciji, odnosno dobavljaju ili dijele podatke na zahtjev aplikacije iz baze podataka ili s drugim aplikacija.



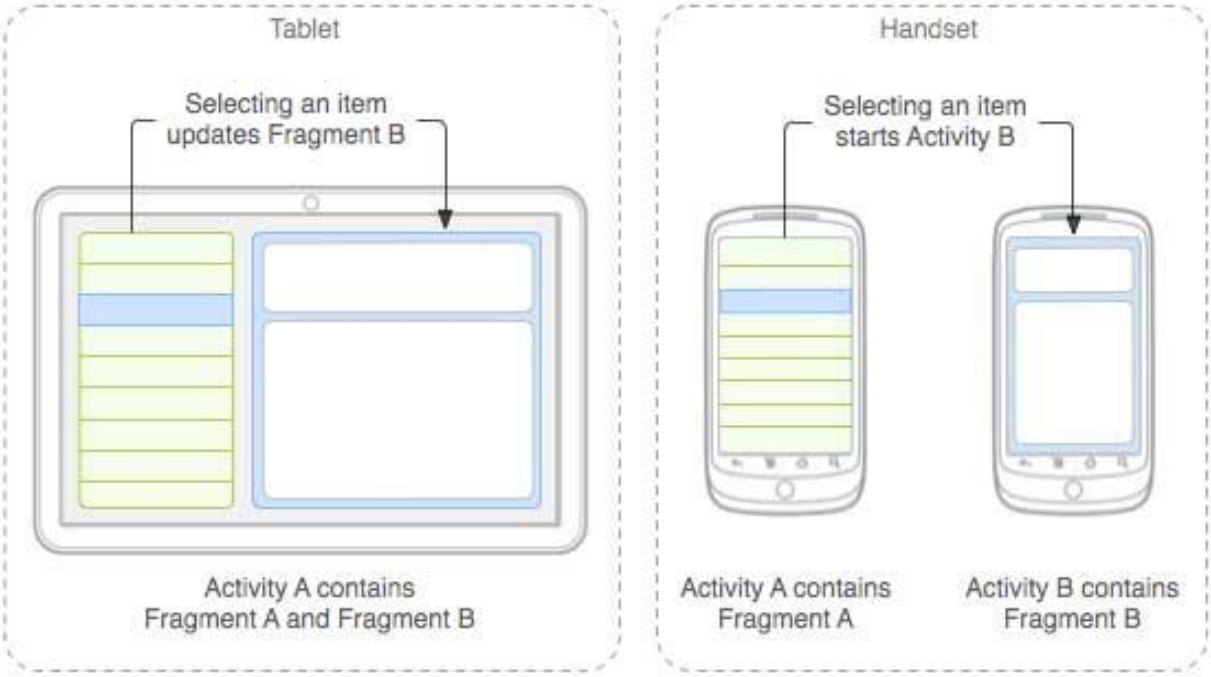
Slika 30: Pružatelj sadržaja [10]

Kako bi pružatelji sadržaja (eng. Content Providers) mogli komunicirati sa slojem podataka, potrebno je koristiti nekoliko metoda:

- onCreate() – u njoj se definira akcija koja se događa kada je pružatelj kreiran
- query() – metoda koja se okida kada su primljeni podaci od klijenta, te se u njoj definira što će se dogoditi s primljenim podacima
- insert() – pomoću ove metode možemo umetnuti novi zapis
- delete() – metoda koja briše postojeći zapis
- update() – metoda pomoću koje možemo osvježavati zapise
- getType() – vraća tip podataka zapisa (npr. ako je vraćenog URI zapisa podataka *content://grad/ulica/adresa,* tip podataka će biti adresa)

4.1.5. Fragmenti

Dio aktivnosti, odnosno pod-aktivnost koja se pokreće nakon nekog događaja na glavnoj aktivnosti (npr. u glavnoj aktivnosti postoji padajući izbornik koji sadrži imena zaposlenika tvrtke i gumb s tekstom „Više informacija“, kada korisnik odabere ime zaposlenika i klikne na gumb aplikacija pokreće fragment (eng. Fragment) u kojem će se ispisati dodatne informacije o zaposleniku), samom korisniku fragment će izgledati kao zasebna aktivnost. Stadiji životnog ciklusa fragmenta je isti kao i kod same aktivnosti.

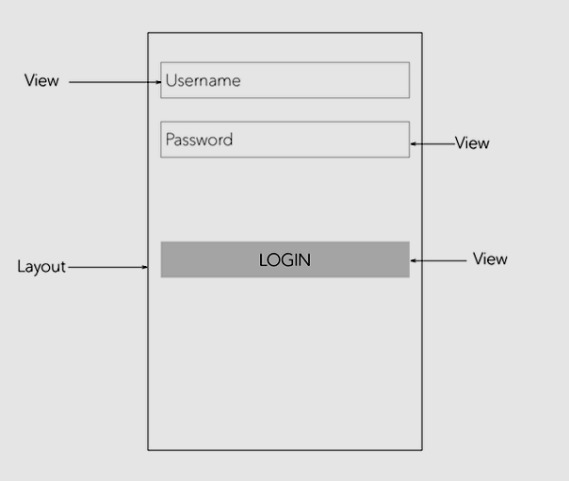


Slika 31: Pokretanje fragmenta [9]

4.1.6.Pogledi

Komponenta koja je zadužena za prikaz dizajna grafičkog sučelja korisniku. Dizajn se definira u zasebnoj datoteci (dolazi u mnogo različitih vrsta kao što su Linear Layout, Relative Layout, Table Layout, itd. i sadrži mnogo raznih elementa kao što su gumbi, izbornici, odabir datuma, itd.) te se prikazuje pomoću pogleda.

Pomoću pogleda (eng. Views) aplikacija može prepoznati je li korisnik imao interakciju s nekim elementom na grafičkom sučelju, odnosno treba li se pokrenuti neki događaj. Kao na primjer slika ispod prikazuje hijerarhiju pogleda u sučelju za logiranje korisnika.



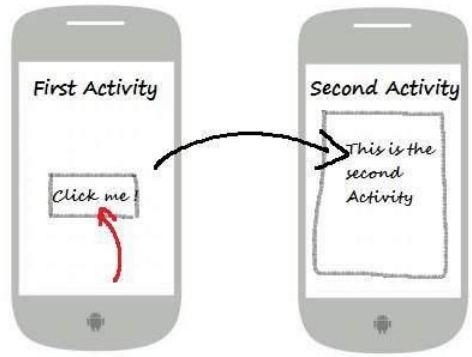
Slika 32: Hijerarhija pogleda [11]

4.1.7.Namjere

Komponenta koja sadrži apstraktni opis operacije koja treba biti izvršena, na primjer ako se na klik gumba korisniku mora otvoriti specifični URL u web pregledniku, u tom slučaju kreira se namjera te se u njoj definira akcija pretraživanja u web pregledniku i akciji se prosljeđuje URL, nakon toga se započinje aktivnost kojoj se prosljeđuje kreirana namjera (eng. Intents) [6].

Namjere dijelimo na dvije vrste:

- Eksplicitne – služe za međusobno povezivanje aktivnosti unutar aplikacije



Slika 33: Eksplicitna namjera [9]

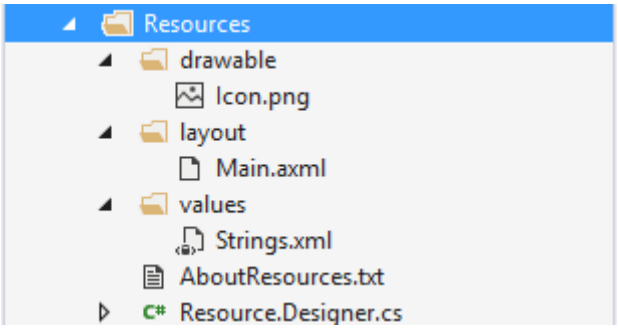
- Implicitne – povezuju aktivnosti unutar aplikacije s elementima izvan aplikacije, kao što je gore navedeni primjer s web preglednikom

4.1.8. Resursi

Vanjski elementi koji se koriste u aplikaciji. U resurse (eng. Resources) aplikacije ubrajamo slike i ikone, bitmape (npr. fontovi), dizajn sučelja te bilo kakve animacije vezane uz dizajn, definiciju boja i string-ova ili bilo koje dodatne datoteke [2].

Spremljeni su u /res datoteku te najčešće sortirani u sljedeće datoteke:

- drawable sadrži slike, ikone ili dijelovi dizajna koji se koriste u sučeljima(gumbi, tekstualna polja, izbornici, itd.)
- layout sadrži XML datoteke sučelja aplikacije
- mipmap sadrži ikonu koja će predstavljati aplikaciju na mobilnom uređaju korisnika te ostale ikone koje se koriste kod pokretanja aplikacije, jedina je razlika što Xamarin automatski generira više mipmap datoteka od kojih svaka predstavlja drugačiju gustoću piksela ikona
- values sadrži definirane boje i string-ove koje se koriste u sučeljima aplikacije



Slika 34: Struktura datoteke resursa [2]

4.1.9. Manifest datoteka

Komponenta koju bi mogli definirati kao ljepilo za sve ostale komponente [3]. Android Manifest je konfiguracijska XML datoteka za aplikaciju u kojoj se definiraju sva dopuštenja (dopuštenje za korištenje Interneta, čitanje memorije, itd..). ili potrebne specifikacije koje su potrebne kako bi ostale komponente pravilno radile. Također sadrži osnovne informacije o samoj aplikaciji kao što je na primjer ime projekta i verzija, na slikama prikazane su manifest datoteke od aplikacije kreirane u Android Studiju i Visual Studiju.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hr.crofi.rasporedterapija">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:requestLegacyExternalStorage="true"
        android:theme="@style/AppTheme">
    </application>
</manifest>
```

Slika 35: Manifest datoteka – Android Studio

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          android:versionCode="1"
          android:versionName="1.0"
          package="com.companyname.terapijaraspored">
  <uses-sdk android:minSdkVersion="28" android:targetSdkVersion="30" />
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
  </application>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

Slika 36: Manifest datoteka – Xamarin

4.2. Rad s elementima korisničkog sučelja

Putem korisničkog sučelja korisnik može ostvariti interakciju s aplikacijom tako da je dizajn korisničkog sučelja iznimno bitan, bez obzira na smisao i funkcionalnosti aplikacije neka pravila dizajna bi se trebala poštovati [14]. Pravila su sljedeća:

- Jasan cilj dizajna – prije početka dizajna sučelja morate znati što je cilj tog sučelja i tko će ga koristiti (kao na primjer sučelje za registraciju mora imati jasno definirano polje za unos potrebnih podataka i naznačena pravila unosa kako se korisnik ne bi zbunio)
- Jednostavnost korištenja – dizajn sučelja ne bi trebao zbunjivati korisnika, odnosno sve interakcije koje korisnik može ostvariti na sučelju moraju imati jasno naznačene

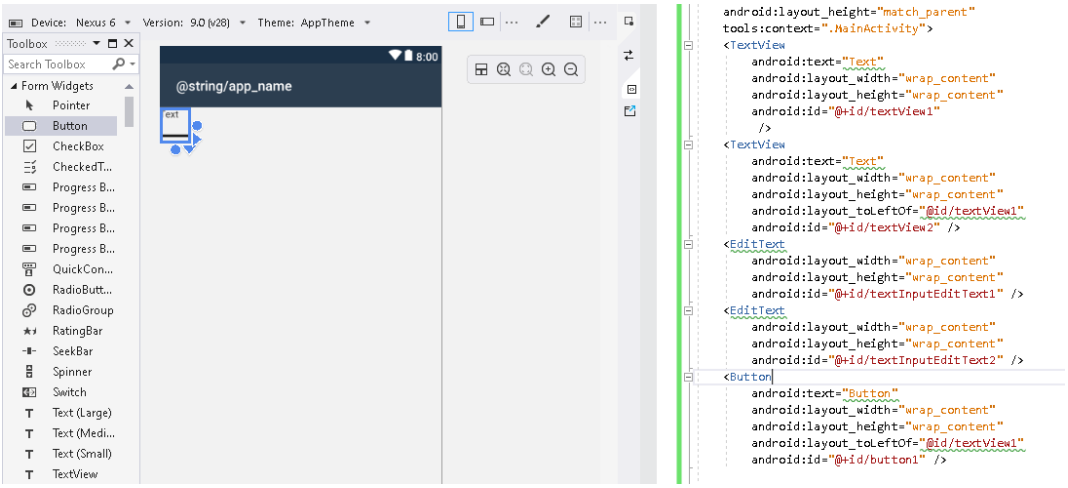
posljedice (na primjer svi gumbi na sučelju trebali bi na sebi imati ispisan tekst ili znak koji korisniku daje informaciju o akciji koja će se ostvariti ako klikne na taj gumb) također je bitno da postupak ostvarivanja interakcija ne bude kompliciran kako bi se i novi korisnici mogli snalaziti. Također je bitno da slijed interakcija ili promjena sučelja bude logično osmišljen

- Korištenje nepisanih pravila dizajna - ako se korisnik koristi s mobilnom aplikacijom vrlo je vjerojatno da je već prije koristio druge mobilne aplikacije ili se barem služio računalom. Kako bi korisniku olakšali snalaženje na sučelju poželjno je primjenjivati neka općenita pravila koja se koriste i u drugim mobilnim aplikacijama ili programima (podcrtavanje linkova, oznaka na gumbu za osvježavanje sučelja, oznaka za izbornik, itd.)
- Poznavanje potreba korisnika – kako bi sučelje bilo ugodno korisniku trebali bi biti upoznati s potrebama korisnika za koja je aplikacija namijenjena, na primjer ako znate da će vašu aplikaciju koristiti većinom stariji ljudi koji dosta često mogu imati probleme s vidom poželjno je da na sučelju budu jasni kontrasti elemenata s kojim se mogu ostvariti interakcije i elemenata sa kojim ne mogu te da općenito elementi i tekst na njima budu veći

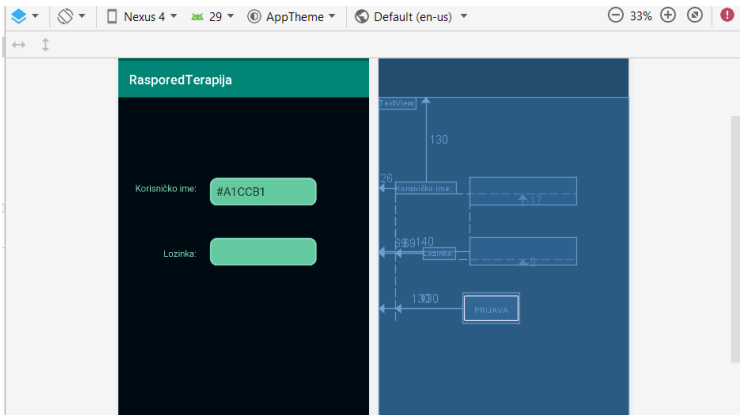
Cilj korisničkog sučelja jest da korisnik može „reći“ aplikaciji što želi od nje. Kako bi to ostvarili neke od osnovnih komponenta koje ćemo koristiti su aktivnosti, pogledi i resursi. Kao što je objašnjeno u prethodnom poglavlju pogled je ustvari komponenta koja je zadužena za prikaz dizajna grafičkog sučelja, tako da u većini stručnih izvora korisničko sučelje (user interface) je klasificirano kao pogled, ali u nastavku će biti objašnjeno kako kreirati elemente dizajna korisničkog sučelja i povezati ih s ostatkom aplikacije.

4.2.1. Kreiranje dizajna korisničkog sučelja

Dizajn sučelja definira se u XML datoteci koja se nalazi u mapi s vanjskim resursima [12]. Bez obzira na prije spomenuta razvojna okruženja sučelje možete dizajnirati ručno pomoću koda ili pomoću alata za dizajn, ako ste upoznati s xml –om preporučila bi prvi način. Ono što je bitno kod dizajna bilo kakvih sučelja za mobilne aplikacije jest to da korisnik može mobilni uređaj držati vertikalno i horizontalno. Zbog toga ako sučelje ne sadrži samo jedan element (npr. pozdravnu poruku) preporučuje se kreirati dizajn sučelja za obje verzije.



Slika 37: Korisničko sučelje za prijavu– Xamarin



Slika 38: Korisničko sučelje za prijavu– Android Studio

Kod kreiranja dizajna prvo se definira tip rasporeda (Layout), kao i Android Studio Xamarin će automatski kao bazni element postaviti LinearLayout, također preporučujem da ga promijenite u RelativeLayout. Kada je definiran tip rasporeda, na sučelje se postavljaju elementi kao što su gumbi, tekstualna polja, padajući izbornici, itd. Kod dizajniranja sučelja mogu se koristiti vanjski resursi, kao na primjer ako želite promijeniti pozadinu nekog elementa na sučelju potrebno je željenu pozadinu staviti u mapu s resursima i definirati njezinu putanju u XML datoteci u kojoj se nalazi element.

Slika 39: XML datoteka sa dizajnom sučelja resursa [6]

Nakon što je sučelje dizajnirano i elementi postavljeni potrebno je definirati akcije koje će se okinuti ako korisnik ostvari interakciju s određenim elementom na sučelju.

4.2.2. Prikaz dizajna sučelja i interakcija sa elementima sučelja

Akcije se definiraju u aktivnostima aplikacije, ali prva akcija koja se mora definirati je prikaz samog sučelja. Kako bi prikazali sučelje kreiramo aktivnost te u njezinom prvom stadiju (onCreate()) pozivamo pogled u kojem je definiran dizajn sučelja (Isječak koda 1).

```
public class MainActivity extends AppCompatActivity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
}
```

Isječak koda 1: Prikaz dizajna sučelja u aktivnosti – Java [12]

Kada je sučelje prikazano korisnik može ostvarivati interakcije s elementima sučelja. Kako bi aplikacija mogla odgovoriti na interakciju korisnika potrebno je u aktivnosti definirati element i događaj koji će aktivirati odgovor.

Slika 40: Element gumba - Xamarin

Na slici iznad označen je element gumba, kada korisnik klikne na njega trebao bi se otvoriti kalendar u kojem korisnik može odabrati datum koji će se potom prikazati u elementu za prikaz teksta (TextView). Prvo je potrebno u aktivnosti kreirati objekte i pridružiti im elemente sa sučelja (Isječak koda 2).

```
Button odaberiDat;
odaberiDat = v1.FindViewById<Button>(Resource.Id.button3);
```

Isječak koda 2: Pridruživanje elementa gumba u aktivnosti - C# [29]

Nakon što su elementi pridruženi određuje se događaj koji će pokretati akciju, u ovom slučaju događaj je pritisak na gumb „Promjeni“ koji je u aktivnosti pridružen objektu odaberiDat. Kada se gumb pritisne poziva se funkcija koja će prikazati kalendar i omogućiti korisniku odabir datuma (Isječak koda 3).

```
odaberiDat.Click += odaberiDatum;
public void odaberiDatum(object sender, EventArgs e) {
     spremanjeDana();
     var odabirDat = DatumOdabir.NewInstance(delegate (DateTime vrijeme) {
     string datumDana = vrijeme.Day + "/" + vrijeme.Month + "/" + vrijeme.Year;
     t1.Text = datumDana;
     provjeriKorisnikeDatuma(datumDana);
     });
     odabirDat.Show(FragmentManager, DatumOdabir.TAG);
    }
```

Isječak koda 3: Prikaz kalendara za odabir datuma - C# [29]

Iako je ovaj primjer jednostavan i na kompliciranijim funkcionalnostima mogu se slijediti isti koraci. Bitna stvar kod rada s elementima sučelja jest sa sigurnošću znati s kojim elementima će se korisnik služiti, kakva će biti interakcija (koji događaj će okidati akciju) i kakva će biti protuakcija.

4.3. Poruke ili obavijesti korisnicima

Ponekad je potrebno obavijestiti korisnika o trenutnom stanju aplikacije kako se ne bi zbunio. Ako se radi o jednostavnoj poruci gdje se korisnik putem tekstualne poruke obavještava o promjeni ili greški u aplikaciji najčešće se u mobilnim aplikacijama to radi pomoću notifikacija koje se mogu prikazivati korisnicima i izvan aplikacije [1] ili obavijesti (Toast poruke) koje se prikazuju kad korisnik koristi aplikaciju. Notifikacije su zasebni element u aplikaciji koje se pokreću kada dolazi do određenog događaja (dolazak poruke, promjena vremena, podsjetnik o obavezi, itd.), pomoću njih se također mogu pokretati i nove aktivnosti koje će korisniku omogućiti da trenutno može djelovati na promjenu stanja.

4.3.1. Kreiranje notifikacije

Kako bi se notifikacija mogla pokrenuti potrebno je kreirati kanal za notifikacije (NotificationChannel) i definirati prioritet kanala koji klasificiramo kao hitne, visoke, srednje i niske [1] (prioritet određuje na koji način će uređaj dati do znanja korisniku da je pokrenuta notifikacija). Osim prioriteta u postupku kreiranja kanala bitno je još definirati ID i ime, u

primjeru ispod kreiran je kanal pod imenom „Kanal1“ s visokim prioritetom [3] (Isječak koda 4).

```
protected void createNotificationChannel(String id, String name, String
description) {
    int importance = NotificationManager.IMPORTANCE_HIGH;
    NotificationChannel channel =
    new NotificationChannel(id, name, importance);
    channel.setDescription(description);
    NotificationManager notificationManager =
    notificationManager.createNotificationChannel(channel);
    }
createNotificationChannel("Kanal.1","Kanal1","Primjer kanala");
```

Isječak koda 4: Kreiranje kanala za notifikacije - Java [3]

Nakon što je kreiran kanal potrebno je definirati sadržaj i id same notifikacije i koji kanal će se koristiti kod pokretanja (Isječak koda 5).

```
protected void sendNotification(View view){
        int notificationID = 101;
        String channelID = " Kanal.1";
        Notification notification =
        new Notification.Builder(NotifyDemoActivity.this,
        channelID)
        .setContentTitle("New Message")
        .setContentText("You've received new messages.")
        .setChannelId(channelID)
        .build();
        notificationManager.notify(notificationID, notification);
    }
```

Isječak koda 5: Definiranje sadržaja notifikacije - Java [3]

Ako se pokretanje notifikacije aktivira nakon nekog događaja, funkcija u kojoj pokrećemo notifikaciju poziva se nakon što se registrira događaj. Na primjer ako se notifikacija iz prethodnog primjera pokreće nakon što korisnik klikne na gumb.

```
Button gumb;
gumb.setOnClickListener( new View.OnClickListener(){
 @Override
 public void onClick(View v){
   sendNotification();
    }
});
```

Isječak koda 6: Pokretanje notifikacije - Java [3]

U ovom primjeru bili su prikazani osnovni koraci kreiranja i prikaza notifikacije, ali notifikacije se mogu koristiti i u kompliciranijim slučajevima, kao na primjer korisniku se može prikazati notifikacija te se dodati gumb kod dizajna notifikacije na koji kad korisnik klikne otvara novu aktivnost aplikacije.

4.3.2. Kreiranje obavijesti koristeći Toast elemente

Razlika između Toast obavijesti i notifikacija je da se Toast elementi ne mogu prikazivati na uređaju ako korisnik ne koristi trenutno aplikaciju [12], ali puno su jednostavniji za korištenje. Zbog tog razloga Toast elementi često se koriste kako bi korisnika obavijestili ako je došlo do pogreške tijekom rada aplikacije (npr. korisnik je unio krive podatke kod logiranja). Samo je potrebno definirati akciju koja će aktivirati prikaz Toast elementa i navesti tekst koji će se ispisati (Isječak koda 7).

```
Toast.MakeText(Application.Context, "Hello toast!",
ToastLength.Short).Show();
```

Isječak koda 7: Toast element - C# [15]

Korisniku će se potom pojaviti jednostavan prozor s porukom koja je definirana kod kreiranja elementa.

Slika 41: Prozor sa porukom [16]

4.4. Rad s lokalnim podacima

U ovom poglavlju će biti objašnjeno kako koristiti, spremati I brisati podatke na mobilnom uređaju. U aplikaciji je ponekad potrebno osim vanjskih podataka koristiti I lokalne podatke, Kako bi uopće mogla pristupiti lokalnim podacima aplikacija mora tražiti dopuštenje, a da bi znala koja dopuštenja mora tražiti od korisnika ona se definiraju u manifest datoteci. Kao primjer uzeti ću spremanje pdf datoteke, ali ista pravila vrijede za bilo koje lokalne podatke (slike, prikaz videa, grafove, itd.). Kako bi mogli imati pristup mapi u koju želimo spremiti pdf datoteku u manifest datoteci potrebno je definirati dopuštenja vezana za vanjsku memoriju, odnosno external storage (Isječak koda 8).

```
<uses-permission
android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

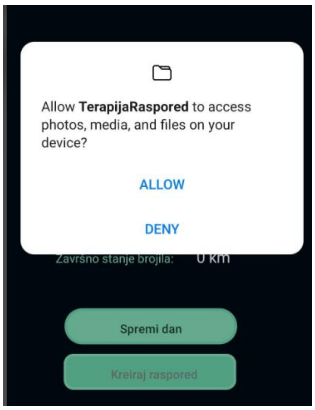
Isječak koda 8: Potrebna dopuštenja za spremanje podataka na uređaj [28]

Kada su definirana dopuštenja na korisniku je da ih odobri, dopuštenje se najčešće traži kod pokretanja aplikacije odnosno u glavnoj aktivnosti ili nakon logiranja kako bi korisnik znao što je potrebno za pravilno funkcioniranje aplikacije, ali može se tražiti I direktno prije same akcije za koju je potrebno dopuštenje (Isječak koda 9).

```
checkPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE,
STORAGE_PERMISSION_CODE);
public void checkPermission(String permission, int requestCode){
    if (ContextCompat.checkSelfPermission(KreiranjeDana.this, permission)
        == PackageManager.PERMISSION_DENIED) {
        ActivityCompat.requestPermissions(KreiranjeDana.this, new String[]
        { permission }, requestCode);
        }
    else {
        Toast.makeText(KreiranjeDana.this, "Permission already granted",
        Toast.LENGTH_SHORT).show();
        }
 }
```

Isječak koda 9: Provjera dopuštenja za korištenje memorije - Java [28]

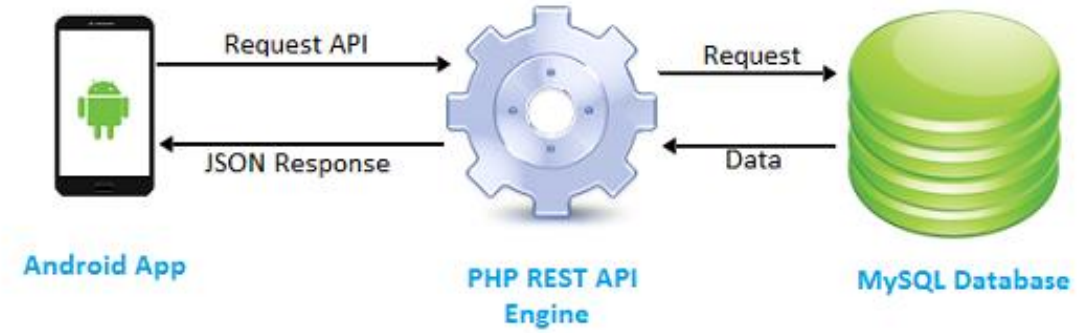
U ovom slučaju provjerava se dopuštenje WRITE_EXTERNAL_STORAGE, odnosno dopuštenje za upis podataka u memoriju mobilnog uređaja. Provjera se izvršava kod pokretanja aktivnosti “KreiranjeDana” I korisniku se prikazuje prozor u kojem može potvrditi ili odbiti zahtjev aplikacije.



Slika 42: Prozor za potvrđivanje zahtjeva aplikacije

4.5. Korištenje web servisa

U većini slučajeva aplikacija će koristiti podatke koji se nalaze u bazi podataka na serveru, pomoću web servisa aplikacija može komunicirati s bazom podataka. Aplikacija uspostavlja konekciju sa servisom i šalje zahtjev, servis uspostavlja konekciju s bazom podataka i obavlja primljeni zahtjev, kada i ako je zahtjev uspješno obavljen servis prenosi rezultat obavljenog zahtjeva aplikaciji [17].



Slika 43: Rest API [17]

PHP REST servisi su jedan od najjednostavnijih načina kako ostvariti ovakav način komunikacije ako se radi o MySQL bazi podataka. Ovakva vrsta servisa kreira se pomoću PHP programskog jezika i najčešće se pohranjuje na isti server gdje se nalazi i baza podataka.

4.5.1. Komunikacija servisa s bazom podataka

Ispod se nalazi primjer servisa (Isječak koda 10) koji prvo uspostavlja vezu nad bazom podataka (u varijabli $con definirani su podaci za pristup bazi), ako je veza uspješno uspostavljena od aplikacije prima korisničko ime i lozinku te pomoću upita nad bazom podataka provjerava postoji li zaposlenik u tablici ZaposleniciAplikacija čije korisničko ime i lozinka odgovara primljenim. Ako korisnik postoji aplikaciji vraća zaposlenikov Id.

```
$con->set_charset("utf8");
 if (mysqli_connect_errno($con)) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
 }
 $username = $_GET['username'];
 $password = $_GET['password'];
 $result = mysqli_query($con,"SELECT * FROM ZaposleniciAplikacija WHERE
 Korime='$username' and Lozinka='$password'");
```

```php
   if($result->num_rows > 0){
      while($res = $result->fetch_assoc()) {
            echo $res["id"];
        }
   else{
        echo "No";
    }
   mysqli_close($con);
```

Isječak koda 10: Primjer PHP servisa

Kada je servis ispunio cilj veza s bazom podataka se zatvara.

4.5.2.Komunikacija aplikacije sa servisom

Kako bi aplikacija mogla uspostaviti vezu sa servisom kreira se zahtjev i prosljeđuje se URL link koji predstavlja destinaciju servisa i ako je potrebno servisu prenosi potrebne podatke (bitno je napomenuti da se podaci mogu prenijeti i nakon što je veza uspostavljena). Kada je veza uspostavljena i podaci poslani aplikacija čeka odgovor servisa, odgovor servisa odnosno rezultat zahtjeva može biti u raznim oblicima kao na primjer JSON objekt [17] ili samo jednostavno ispisan niz znakova, u primjeru ispod primijenjen je drugi jednostavniji slučaj (Isječak koda 11).

```
Try {
      HttpWebRequest request = (HttpWebRequest)WebRequest.Create(link);
      HttpWebResponse response = (HttpWebResponse)request.GetResponse();
      Stream dataStream = response.GetResponseStream();
      StreamReader reader = new StreamReader(dataStream);
      string responseFromServer = reader.ReadToEnd();
      odgovor = responseFromServer;
    }
catch (Exception e) {
         odgovor= "error";
       }
```

Isječak koda 11: Uspostavljanje veze aplikacije sa servisom - C# [17]

Kada je servis odgovorio, aplikacija čita odgovor i nastavlja s radom. Sama komunikacija sa servisom u aplikaciji odvija se u pozadinskoj dretvi. Rad tih dretvi moguće je sinkronizirati kako bi više dretvi odjednom moglo komunicirati sa servisom ili jedna za drugom. Također još jedna od bitnih stvari kod komunikacije s web servisima je pristup internetskoj vezi.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Isječak koda 12: Potrebna dopuštenja za pristup internetskoj vezi [29]

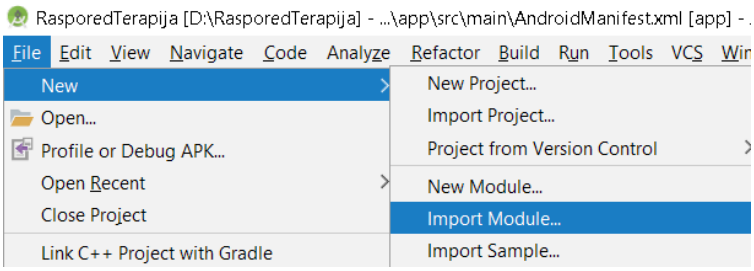
Kao i kod pristupa memoriji uređaja potrebno je u manifest datoteci definirati dopuštenje za pristup internetskoj vezi kako bi aplikacija uopće mogla poslati zahtjev servisu.

4.6. Korištenje 3rd party biblioteka

Vrlo je vjerojatno da će se tijekom razvoja aplikacije pojaviti potreba za korištenjem vanjskih biblioteka. Vanjsku biblioteku možemo interpretirati kao skupinu definiranih funkcionalnosti čija implementacija optimizira razvoj aplikacije [18]. Takve biblioteke često dolaze s vlastitom dokumentacijom koja pojašnjava kako najefikasnije primjenjivati funkcionalnosti u samoj aplikaciji. Prednost korištenja takvih datoteka je redukcija potrebnog vremena kod kreiranja aplikacije, ali mana je što morate biti jako pažljivi da korištene biblioteke odgovaraju ciljanoj Android verziji, zvuči jednostavno ali vrlo lako u ovom segmentu može doći do problema. Zbog tog razloga savjetujem da se prije uključivanja vanjskih biblioteka informirate o kvaliteti same biblioteke i ako ne poznajete nikog tko ju je već koristio, potražite na nekom od foruma iskustva drugih ljudi s korištenjem. Također se ne preporučuje uključivanje neprovjerenih biblioteka koje se ne spominju nigdje u člancima ili knjigama, pogotovo ako se radi o bibliotekama koje se tiču zaštiti ili pristupu podacima. U prethodnim poglavljima nije bilo potrebe za podjelom između aplikacija kreiranih u Android Studiju (Java programski jezik) i Xamarinu (C# programski jezik) jer je princip rada isti samo se kod prilagođava ovisno o programskom jeziku, ali pošto se dosta često znaju pojaviti greške kod dodavanja vanjskih biblioteka u ovom poglavlju odlučila sam detaljnije objasniti postupak dodavanja biblioteka.

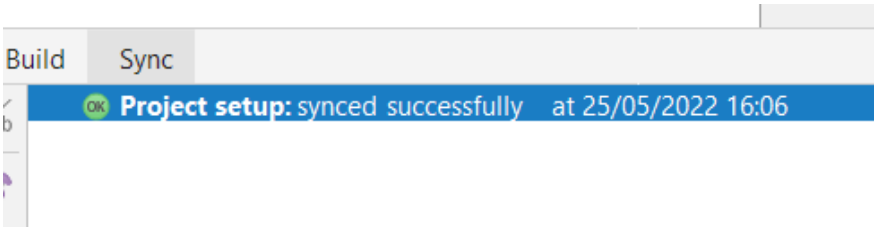
4.6.1. Android Studio

Ako trebate dodati vanjsku biblioteku projektu kreiranom u Android Sudio razvojnom okruženju kliknite na File -> New -> Import Module



Slika 44: Dodavanje vanjske biblioteke – Android studio

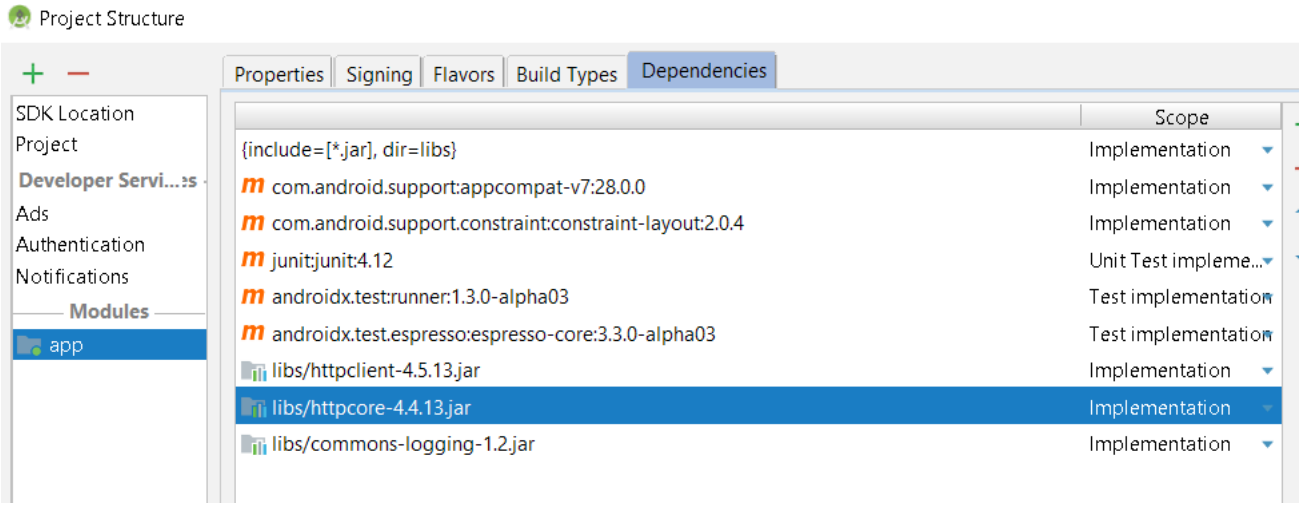
Otvorit će se prozor u kojem je potrebno odabrati željenu biblioteku koja će se dodati u projekt. Nakon toga build-ajte projekt. Otvorite build.gradle (Module:app) datoteku i u dependencies dijelu dodajte ime biblioteke. Sinkronizirajte projekt, ako nije došlo do grešaka trebala bi se pojaviti sljedeća obavijest.



Slika 45: Sinkroniziranje projekta– Android studio

Ako želite vidjeti koje dodatke projekt već koristi kliknite na File -> Project Structure ->Dependecies.

Ovdje će biti prikazane sve biblioteke koje se koriste u projektu.

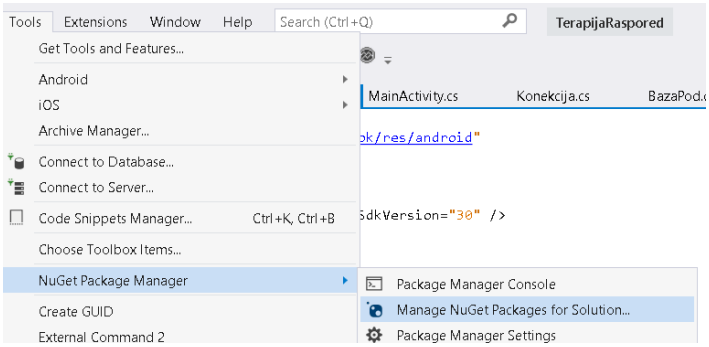


Slika 46: Prikaz svih dodanih biblioteka– Android studio

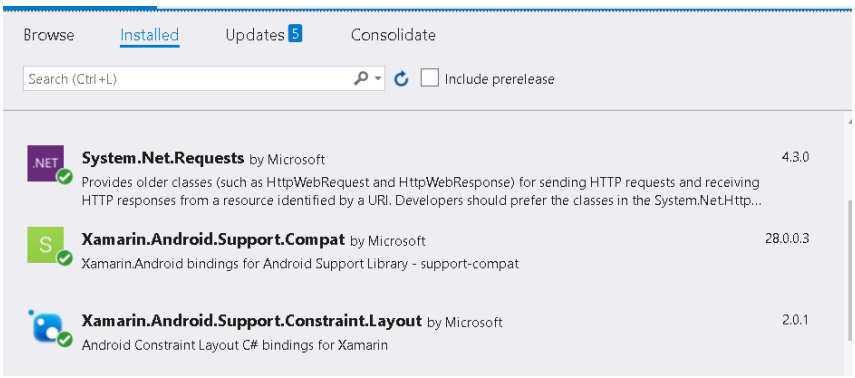
Ako se dogodi greška kod dodavanja, provjerite poklapa li se dodana biblioteka sa prethodno dodanim.

4.6.2. Xamarin (Visual Studio)

Kako bi dodali vanjsku biblioteku kliknite na Tools -> Nuget Package Manager -> Manage NuGet Packages for Solution.



Slika 47: Dodavanje vanjske biblioteke – Xamarin



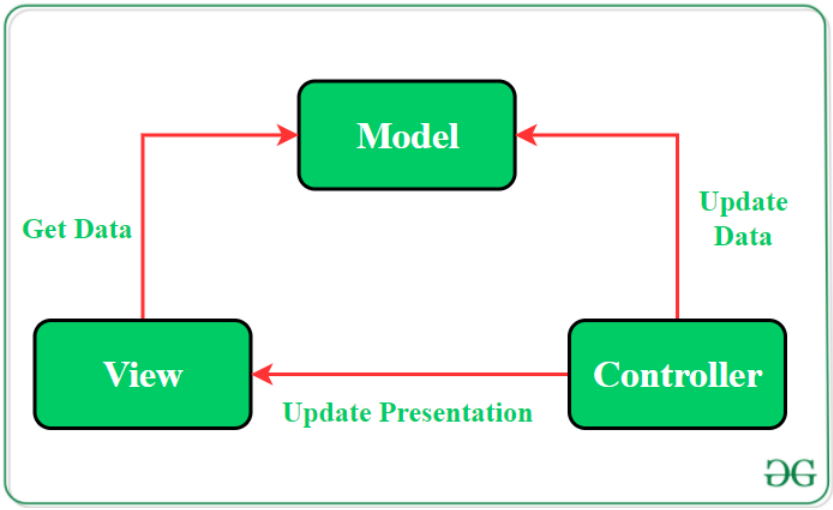
Slika 48: Nuget Package Manager– Xamarin

Otvara se prozor u kojem možete vidjeti već dodane biblioteke, njihova ažuriranja te dodavati nove.

4.7. Uobičajene arhitekture android aplikacija

Kod razvoja aplikacije bitno je kvalitetno isplanirati arhitekturu aplikacije. Kvalitetna arhitektura trebala bi biti jednostavna za korištenje i održavanje kao i dovoljno fleksibilna za nadogradnju kako u budućnosti ako želimo dodati neku novu funkcionalnost neće biti potrebno ispočetka kreirati cijelu aplikaciju. Arhitekture koje se najčešće primjenjuju kod android aplikacija su MVC (Model – View - Controller), MVI (Model – View – Intent) i MVVM (Model – View - ViewModel) u nastavku će biti detaljnije obrazložene.

4.7.1. MVC



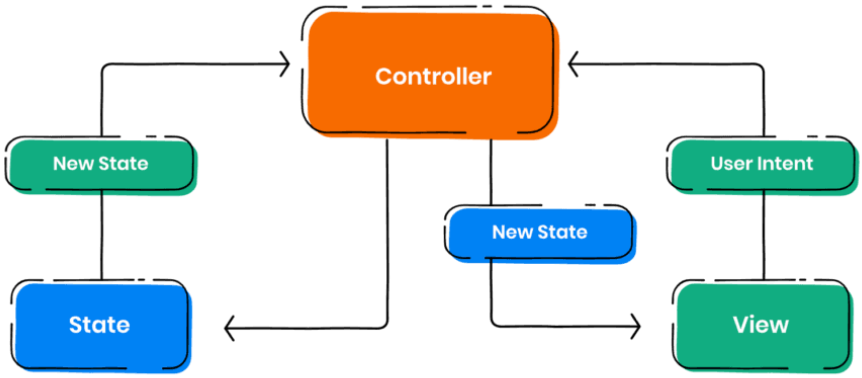
Slika 49: MVC (Model – View - Controller) arhitektura [19]

Ovakva vrsta arhitekture podrazumijeva razdvajanje aplikacije na tri osnovne komponente:

- Model – komponenta koja je zadužena za dobavljanje podataka iz baze i komunikaciju s web servisima (bitno je napomenuti da ne komunicira sa samim sučeljem)
- View – komponenta koja prikazuje sučelje korisniku i omogućuje korisniku komunikaciju s aplikacijom
- Controller – komponenta koja omogućuje komunikaciju između prve dvije komponente, jednostavno rečeno prenosi zahtjeve korisnika Model komponenti te osvježava podatke koji se prikazuju u View komponenti

Na primjer ako želimo kreirati aplikaciju koja ima funkcionalnost prijave korisnika, prvo ću kreirati pogled koji će prikazivati dizajn sučelja, odnosno View komponentu. Potom ću kreirati aktivnost koja će predstavljati Controller komponentu i u njoj definirati prikaz prethodno kreiranog pogleda. Nakon toga ću kreirati zasebnu klasu koja će iz lokalnih podataka ili baze dohvaćati podatke i ona će predstavljati Model komponentu, ali sve funkcionalnosti u ovoj klasi moraju biti definirane tako da ih se može pozvati u aktivnosti (Controller komponenti) i ne smiju koristiti elemente iz pogleda (View komponente) već se dohvaćanje i vraćanje podataka vrši na temelju informacija koje se prosljeđuju tijekom pozivanja funkcionalnosti iz aktivnosti koja će na temelju vraćenih podataka izvršiti određene promjene na pogledu.

4.7.2. MVI



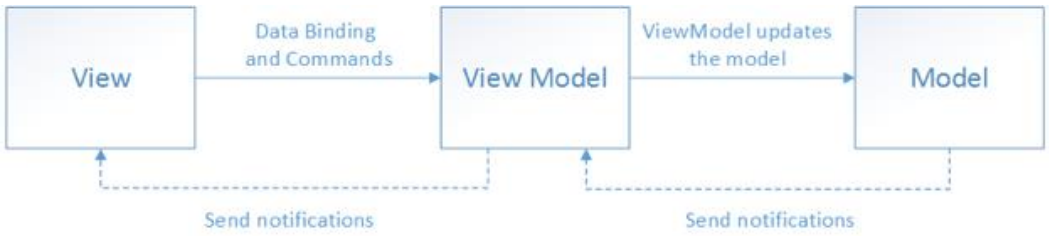
Slika 50: MVI (Model – View - Intent) arhitektura [20]

U MVI arhitekturi aplikacija se također dijeli na tri dijela ali način na koji komponente međusobno funkcioniraju je drugačiji nego kod MVC arhitekture.

- Model – kao i u MVC arhitekturi ova komponenta je zadužena podatke, ali u MVI arhitekturi također je zadužena i za obradu trenutnog stanja aplikacije, odnosno ako se dogodi neka promjena (korisnik klikne na gumb) stanje se mijenja
- View – komponenta zadužena za prikaz i interakciju korisnika sa sučeljem
- Intent – komponenta u kojoj su definirana moguća stanja aplikacije te uz svako stanje pridružena je neka akcija

Temelj ove arhitekture je ustvari trenutno stanje aplikacije, ako korisnik ostvari interakciju s aplikacijom i ako ta interakcija zahtjeva dohvat određenih podataka, aplikacija će promijeniti stanje te će to stanje odrediti daljnji tijek događaja te će se također moći prikazati korisniku. Na primjer ako korisnik koristi aplikaciju koja sadrži funkcionalnost prikaza svih rezultata nogometnih utakmica u zadnjih godinu dana i ta se funkcionalnost aktivira pritiskom na gumb. Kada korisnik klikne na gumb stanje aplikacije promijenit će se u dohvat podataka, u tom stanju korisniku se može npr. prikazati notifikacija s tekstom „Dohvat podataka u procesu“ ili samo traka koja prikazuje postotak dohvaćenih podataka, kad se dohvate svi podaci stanje aplikacije će se ponovo promijeniti, ova promjena stanja može aktivirati promjenu sučelja u kojem će biti ispisani svi dohvaćeni podaci. Cilj ovakve arhitekture je omogućiti tok podataka u jednom smjeru, a u isto vrijeme sve promjene u aplikaciji kontrolirati iz jedne komponente [20].

4.7.3. MVVM



Slika 51: MVVM (Model – View - ViewModel) arhitektura [21]

Cilj ove arhitekture je razdvajanje programske logike i korisničkog sučelja, ostvaruje se razdvajanjem aplikacije na sljedeće komponente:

- Model – komponenta koja je zadužena za upravljanje podacima (iz lokalnih ili vanjskih izvora) koje prosljeđuje u ViewModel komponentu i sadrži poslovnu logiku
- View – komponenta koja sadrži elemente za prikaz korisničkog sučelja (aktivnosti, fragmenti i XML datoteke) i registrira interakciju korisnika s aplikacijom te prosljeđuje zahtjev korisnika ViewModel komponenti
- ViewModel – omogućuje komunikaciju između View i Model komponenti, prima podatke iz Model komponente i prenosi ih View komponenti kako bi se mogli prikazati korisniku

Poanta ovakve arhitekture jest da se View komponenta osvježava svaki puta kada se ViewModel komponenta promjeni te tako korisniku omogući trenutni prikaz stvarnih podataka. Kako bi MVVM arhitektura bila kvalitetno implementirana bitno je uspostaviti jasnu komunikaciju između komponenti. Ako nemate želju striktno primjenjivati neke od arhitektura, ako ništa drugo preporučujem da razdijelite funkcionalnosti na što više detaljnijih i zasebnih klasa (najjednostavnija primjena ovog savjeta je kada u aplikaciji radite na nekoj funkcionalnosti i znate da će se neki dio koda ponavljati kroz aplikaciju, kreirajte posebnu klasu i definirajte funkciju koja prima potrebne parametre i primjenjuje ih na taj dio koda, tako ne morate više puta pisati isti kod već samo kreirate objekt te klase i pozovete funkciju), ovaj savjet će vam omogućiti jednostavniju nadogradnju a i samo snalaženje u aplikaciji. Također kreirajte opisne klase pogotovo ako se radi o podacima koji se koriste u aplikaciji jer tako puno lakše možete s njima upravljati. Pokušajte što više odvojiti sučelje od poslovne logike, što znači da klase u kojima se prikazuje dizajn sučelja i registrira interakcija korisnika sa sučeljem (npr. aktivnosti) sadrže što manje poslovne logike koja bi trebala biti smještena u zasebne klase.

4.8. Uobičajeni proces razvoja

Kada se razvija aplikacija poželjno je slijediti neke osnovne korake kako bi proces prošao što jednostavnije i brže. U nastavku ću navesti korake koje preporučuje m slijediti ili barem ih se donekle pridržavati bez obzira u kojem programskom jeziku kreirate aplikaciju [22].

4.8.1. Definiranje strategije

Prvi korak kod razvoja bilo kakve aplikacije trebao bi biti definiranje strategije samog razvoja. Pod to se podrazumijeva da znate samu svrhu aplikacije koju planirate kreirati, jer ako ne znate svrhu aplikacije ne možete ni jasno definirati funkcionalnosti. Definirajte i istražite ciljane korisnike, na primjer hoće li aplikaciju koristiti ljudi iz drugih zemalja i treba li unajmiti prevoditelja ili će možda aplikaciju koristiti samo starije osobe te je bitno da se koristi jednostavan i jasan dizajn. Također je bitno istražiti postoji li na tržištu već takva aplikacija i ako postoji na što su se korisnici žalili i može li se taj dio nekako unaprijediti. U ovom stadiju razvoja preporučljivo je potražiti drugo mišljenje ili čak više njih, čisto da vidite kako drugi ljudi vide vašu ideju za aplikaciju.

4.8.2. Analiza i planiranje

U ovom koraku potrebno je jasno definirati funkcionalnosti aplikacije i što je potrebno kako bi se ostvarile u određenom vremenskom roku. Na primjer ako je riječ o velikoj tvrtki i velikoj aplikaciji oni će morati ispitati imaju li dovoljno zaposlenika i imaju li zaposlenici potrebno znanje kako bi aplikaciju mogli izraditi u zadanom vremenskom roku i ako nemaju analizirati je li isplativo zaposliti još ljudi, izbaciti neke funkcionalnosti ili pregovarati s klijentom o većoj cijeni proizvoda. Također je potrebno analizirati količinu i brzinu podataka koju će koristiti aplikacija te ima li tvrtka potrebnu opremu za ispuniti takva očekivanja ili će je morati nadograditi i kolika će biti cijena nadogradnje. Iz moje perspektive prije početka kreiranja same aplikacije znala sam da će mi biti potrebna baza podataka, a pošto sam već imala registriranu domenu na netdomeni serveru odlučila sam tamo kreirati bazu i koristiti php servise za komunikaciju. Uglavnom u ovom koraku potrebno je analizirati zahtjeve aplikacije, definirati potrebne i dostupne resurse i isplanirati najisplativiju verziju daljnjeg razvoja.

4.8.3. Dizajn sučelja

Ovaj korak čini se jednostavan ali ustvari jedan je od važnijih jer ljudi su vizualna bića, tako da je bitno kreirati dobar dizajn sučelja, ali što to ustvari znači? Dobar dizajn sučelja mora biti jasan, jednostavan za korištenje i naravno lijep. Ove osnovne principe najjednostavnije će te ispuniti tako da prvo kreirate skice dizajna, možete na papiru ili koristiti neke od alata kao što su Invison.

Slika 52: Invision alat [22]

Kod izrade skice bitno je uzeti obzir tok podataka u aplikaciji i same funkcionalnosti aplikacije. Nakon toga predstavite skicu dizajna budućem korisniku i primijenite savjete ili promijenite stvari koje su bile kritizirane.

4.8.4. Kreiranje aplikacije

Kako bi krenuli sa samim programiranjem aplikacije potrebno je prvo definirati arhitekturu i tehnologiju odnosno metode razvoja. Samu arhitekturu odabirete na temelju funkcionalnostima i zahtjevima aplikacije. Potom je potrebno odabrati tehnologije razvoja koje će najisplativije i najbolje funkcionirati uz odabranu arhitekturu. Kada je sve odabrano i isplanirano, podijelite aplikaciju na određene dijelove i na temelju definirane arhitekture odredite okvirne vremenske rokove kada bi određeni dio aplikacije trebao biti gotov. Zadnji korak u ovo stadiju je samo pisanje koda s ciljem kreiranja prethodno isplanirane aplikacije pridržavajući se vremenskih rokova (kasnije u radu ovaj korak bit će detaljnije objašnjen kako bi se jasnije prikazala razlika između pisanja koda za aplikaciju u javi ili C# programskom jeziku).

4.8.5.Testiranje

Kada je aplikacija kreirana i sve funkcionalnosti naizgled funkcioniraju vrijeme je za testiranje. Kako bi se dobili najdetaljniji rezultati i otkrili skriveni problemi u aplikaciji samo testiranje najbolje je podijeliti u sljedeće kategorije [22]:

- Testiranje korisničkog iskustva – ispituje se hoće li uopće korisnik biti zadovoljan s aplikacijom. Koncentrira se na samu interakciju korisnika sa sučeljem (je li dizajn zadovoljavajući i jesu li sučelja dovoljno jednostavna za korištenje i snalaženje te postoji li koji dio koji bi bilo dobro izmijeniti)
- Testiranje funkcionalnosti – u ovom dijelu svaka funkcionalnost koju bi aplikacija trebala izvršavati mora proći testove koji ju dovode u različite i na kraju ekstremne slučajeve kako bi bili sigurno da se korisnik može služiti aplikacijom u raznim situacijama bez da aplikacija podbaci
- Testiranje performansi aplikacije – ako je aplikacija uspješno prošla kroz test funkcionalnosti potrebno je provjeriti koliko brzo i kvalitetno se te funkcionalnosti izvršavaju. Na primjer koliko je vremena potrebno da se na zahtjev korisnika podaci iz baze ili lokalne memorije učitaju i može li se na koji način to vrijeme skratiti. Koliko memorije i baterije samog mobilnog uređaja aplikacija troši ili hoće li serveri izdržati opterećenje ako aplikaciju koristi više korisnika istovremeno
- Sigurnosno testiranje – testira se sama sigurnost aplikacije, na primjer koliko je teško ili lagano pristupiti bazi podataka nekome tko nema pristup, jesu li privatnost i podaci korisnika zaštićeni i koja razina sigurnosti je uopće potrebna u aplikaciji
- Testiranje na uređajima – kada je aplikacija prošla ostale kategorije, instalira se i testira na samim uređajima kojima je namijenjena

Nakon ovog koraka kreće isporuka aplikacije.

4.8.6. Isporuka aplikacije

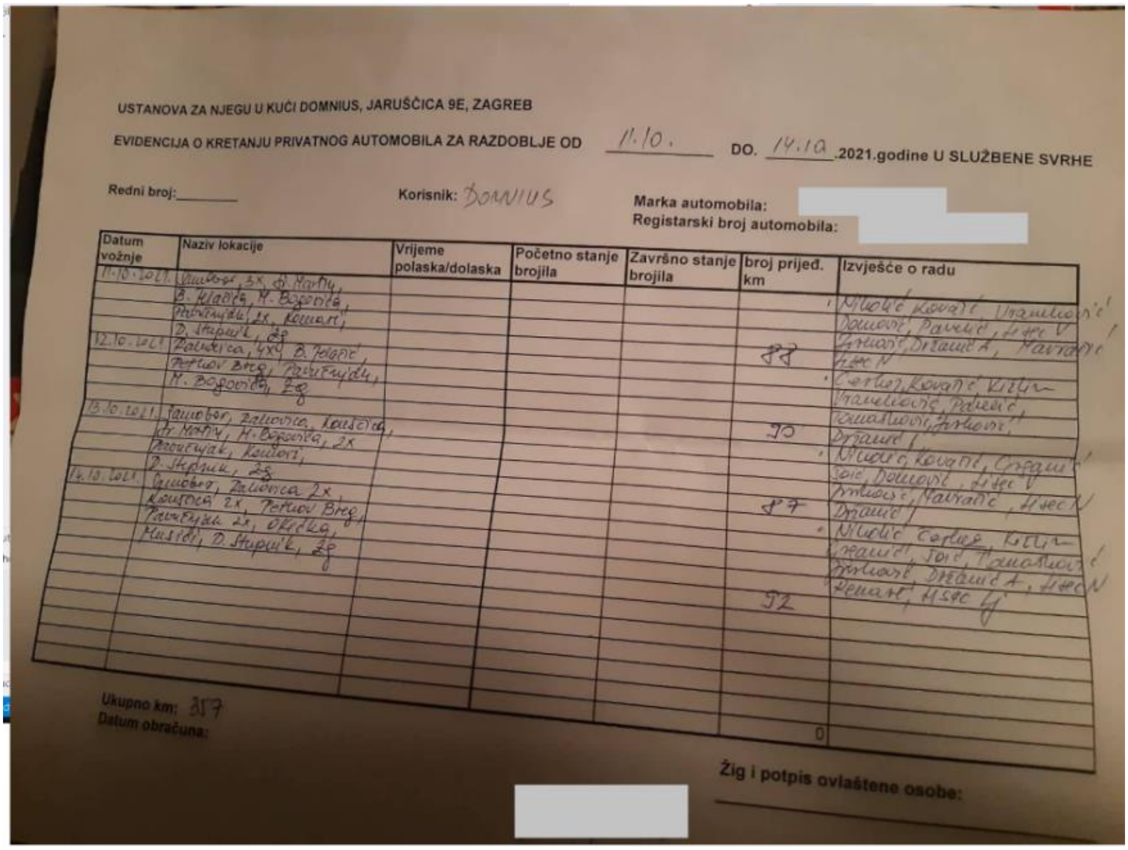
Aplikacija je uspješno kreirana, prošla je sva testiranja i ako su se pojavile, sve greške su popravljene što znači da je vrijeme za isporuku aplikacije. Sama isporuka može se izvršiti na nekoj od mrežnih trgovina aplikacija (npr. Google play) ili u ekstremnijim slučajevima direktno instalirati na mobilne uređaje korisnika. Nakon što je taj korak obavljen i korisnici imaju pristup aplikaciji, bitno je ostati dostupan odnosno pripremiti odgovore na određena pitanja o radu aplikacije, kreirati dokumentaciju za korisnika koji objašnjava rad aplikacije ili čak ako je riječ o aplikaciji koju je naručila tvrtka za svoje zaposlenike održati prezentaciju koja će u detalje provesti korisnike kroz način rada s aplikacijom te će korisnici odmah moći postaviti pitanja ako im neki dio nije jasan.

5. Razvoj aplikacije

U ovom poglavlju bit će prikazan usporedan razvoj aplikacije u oba programska jezika, ali prije nego što dođemo do samog programskog koda potrebno je proći kroz korake definiranja strategije, analize i planiranja i dizajna sučelja. Toplo preporučujem iz iskustva da bez obzira na programski jezik i razvojno okruženje ove korake razvojnog proces obavite prije nego što napišite i jednu liniju koda, jer u suprotnom vrlo lako dolazi do kaotičnog i neurednog projekta. Možda će raditi i izvršavati trenutno zadane funkcionalnosti, ali ako će u budućnosti trebati održavanje ili nadogradnju trebat će vam puno više vremena. Sagledajte proces razvoja projekta kao pospremanje razbacane odjeće u ormar, odjeću možete staviti na hrpu, natrpati u ormar i tehnički će soba biti čista, ali kada će te tražiti specifičan komad odjeće morat će te izvaditi nabacanu hrpu iz ormara, cijelu ju pretražiti i ponovo je natrpati u ormar, k tome sva odjeća će vam biti izgužvana i morat će te ju ispeglati prije nošenja. Umjesto toga možete sortirati komade odjeće, uredno ih posložiti u ormar i kada tražite komad odjeće odmah će te znati gdje tražiti. Zbog toga će u prvom poglavlju biti detaljno opisana svrha aplikacije kako bi se mogla definirati strategija, potom će biti opisane funkcionalnosti aplikacije i zahtjevi baze podataka (Analiza i planiranje). Nakon toga kreirat ću dizajn sučelja na temelju zahtjeva aplikacije, bitno je napomenuti da će obje aplikacije imati isti dizajn i koristit će istu bazu podataka. Kada su ti koraci završeni krenuti ću s korakom kreiranja aplikacije u kojem ću prvo odabrati odgovarajuću arhitekturu i podijeliti na dijelove sami postupak programiranja.

5.1. Opis programskog proizvoda

Smisao aplikacije je olakšati djelatnicima tvrtke Domnius (https://domnius.hr/) bilježenje prijeđenih kilometara tijekom radnog dana te pomoću skupljenih informacija kreirati izvještaj. Jedna od usluga koju tvrtka pruža jest fizikalnu terapiju u kući pacijenta, što znači da fizioterapeut zaposlen u tvrtki dolazi u kuću pacijenta te pruža potrebne usluge fizikalne terapije. Fizioterapeuti koji su zaduženi za taj dio posla svaki radni dan svojim vozilom obilaze klijente, te jednom mjesečno (prije obračuna za plaće) tvrtki moraju podnijeti izvještaj koji sadrži popis klijenata koje su obišli (njihova prezimena), lokacije (ne točna adresa već samo mjesto i ime ulice), vrijeme polaska/dolaska (opcionalno), početno stanje brojila i završno stanje brojila (opcionalno) i broj prijeđenih kilometara. Svi podaci moraju biti sortirani po datumu. Ispod se nalazi primjer formata izvještaja.



USTANOVA ZA NJEGU U KUĆI DOMNIUS, JARUŠČICA 9E, ZAGREB

EVIDENCIJA O KRETANJU PRIVATNOG AUTOMOBILA ZA RAZDOBLJE OD 11.10. DO. 14.10. 2021.godine U SLUŽBENE SVRHE

Redni broj:________ Korisnik: DOMNIUS

Marka automobila:

Registarski broj automobila:

Datum vožnje	Naziv lokacije	Vrijeme polaska/dolaska	Početno stanje brojila	Završno stanje brojila	broj prijeđ. km	Izvješće o radu
11.10.2021.	[illegible]					[illegible]
12.10.2021.	[illegible]				88	[illegible]
13.10.2021.	[illegible]				90	[illegible]
14.10.2021.	[illegible]				87	[illegible]
					92	
					0	

Ukupno km: 357

Datum obračuna:

Žig i potpis ovlaštene osobe:

Slika 53: Ispis izvještaja na papiru

Trenutno zaposlenici ručno ispunjavaju izvještaj te ga potom predaju tvrtki, na temelju predanog izvještaja zaposlenici primaju naknadu za potrošeno gorivo vlastitog vozila. Cilj aplikacije jest automatsko generiranje izvještaja u pdf formatu koji zaposlenik može samo proslijediti tvrtki ili ga ispisati ako želi.

Na temelju svrhe aplikacije možemo zaključiti da nije potreban prijevod teksta u aplikaciji već će biti dovoljan samo hrvatski jezik, također potreban je jasan i jednostavan dizajn (po mogućnosti s jakim kontrastom boja između teksta i pozadine) kako bi korisnici mogli što brže i jednostavnije unijeti podatke jer će ih vrlo vjerojatno unositi kada završe s klijentom, znači u autu ili na pauzi. Pošto će korisnici vjerojatno unositi podatke na brzinu i bez previše provjera potrebno je imati mogućnost ispravka unesenih podataka.

5.2. Funkcionalnosti aplikacije

U prethodnom poglavlju pojašnjena je svrha aplikacije, kako bi aplikacija mogla što jasnije služiti odabranoj svrsi definirane su sljedeće funkcionalnosti.

Funkcionalnost	Opis
Prijava	Zaposlenici se prijavljuju pomoću dodijeljenog korisničkog imena i lozinke. Ova funkcionalnost koristit će se na početnom sučelju kada se pokrene aplikacija, ako korisnik unese točne podatke bit će preusmjeren na sučelje za kreiranje dana
Odabir datuma	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Kako bi olakšali unos i smanjili mogućnost pogreške u formatu unesenog datuma korisniku će se otvoriti kalendar u kojem će moći izabrati željeni datum radnog dana za kojeg unosi podatke
Dodavanje korisnika u popis obiđenih klijenata	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Korisniku će biti prikazan padajući izbornik u kojem će biti ispisani klijenti koji su obiđeni u tom radnom danu. Korisniku će biti omogućeno dodavanje klijenata u popis.
Brisanje korisnika iz popisa obiđenih klijenata	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Korisniku će biti omogućeno brisanje klijenta iz gore spomenutog popisa.
Izračun i ispis prijeđenih kilometara	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Na temelju popisa obiđenih klijenata prikazivat će se okvirni broj prijeđenih kilometara koji se izračunava pomoću koordinatnih podataka svakog klijenta. Također broj kilometara mora se osvježiti ako korisnik doda ili briše klijente iz popisa
Unos stanja brojila	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Korisnik može ako želi unijeti početno i završno stanje brojila u radnom danu kako bi se lakše mogle provjeriti oscilacije prijeđene rute tijekom radnog dana.
Spremanje promjena	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Kada je korisnik zadovoljan s unesenim podacima o radnom danu spremit će promjene u bazu klikom na gumb.

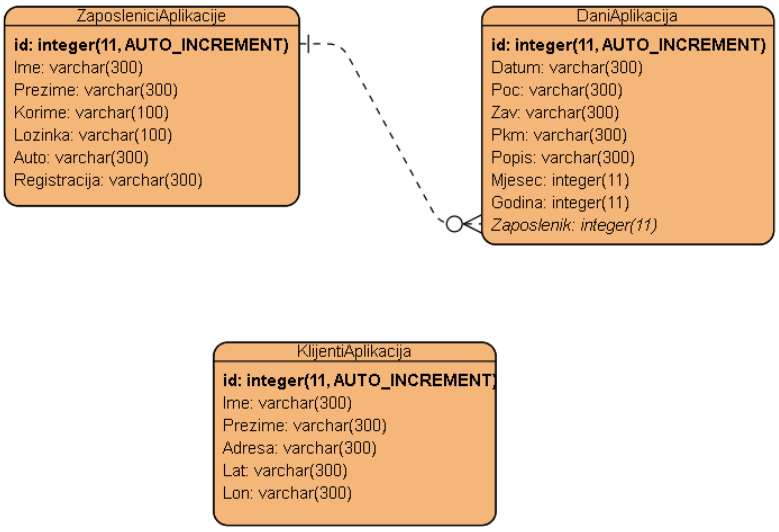
Generiranje izvještaja za odabrane datume u .pdf formatu	Funkcionalnost će biti ostvarena na sučelju za kreiranje dana. Korisnik će imati mogućnost odabrati raspon datuma u čiji će se podaci koristiti u izvještaju. Na temelju unesenih podataka odabranih datuma u mobilnom uređaju bit će generiran izvještaj u pdf formatu.

Tablica 1: Opis funkcionalnosti aplikacije

Ono što moram napomenuti jest da zaposlenici ne žele da ih se prati putem lokacije mobitela već da se izračuna broj prijeđenih kilometara na temelju unesenih klijenata, također razlika između početnog i završnog stanja brojila nije uvijek ista prijeđenim kilometrima, ali tvrtka želi da broj prijeđenih kilometara za dan predstavlja udaljenost između lokacije klijenata koji se nalaze u popisu za taj datum.

5.3. Baza podataka

Prije kreiranja baze podataka potrebno je definirati njene zahtjeve, kao na primjer koliko će korisnika koristiti ovu aplikaciju, u ovom slučaju dobila sam odgovor između 20 i 50, što nije veliki broj. Također unos podataka jednog korisnika neće utjecati na unesene podatke drugih korisnika. Na temelju prethodno utvrđenih funkcionalnosti aplikacije baza mora sadržavati podatke o samim korisnicima odnosno zaposlenicima, klijentima aplikacije i podatke o radnom danu zaposlenika. Na slici 54 je prikazan ERA model (kreiran u alatu Visual Paradigm Online [23]) i detaljniji opis tablica prema kojem je kreirana baza.



Slika 54: ERA model baze podataka

U nastavku će biti detaljnije pojašnjena svrha određenih entiteta u tablicama.

Tablica ZaposleniciAplikacije Sadrži podatke o zaposlenicima koji će koristiti aplikaciju.	
Ime entiteta	**Opis**
id	Id zaposlenika, entitet se koristi kao primarni ključ tablice te vanjski ključ u tablici DaniAplikacije i služi za identifikaciju određenog zaposlenika
Auto	Vrsta automobila koju zaposlenik vozi (npr. Clio), entitet će se koristiti kod krajnjeg ispisa .pdf izvještaja

Tablica 2: Opis tablice ZaposleniciAplikacije

Tablica KlijentiAplikacija Sadrži podatke o klijentima koje zaposlenici obilaze	
Ime entiteta	**Opis**
id	Id klijenta, u bazi se nalazi popis svih klijenata tvrtke koje zaposlenici obilaze, na temelju identifikacijskog broja u aplikaciji će se koristiti podaci određenog klijenta, Id klijenta se koristi u tablici DaniApikacija kod bilježenja popisa obiđenih klijenata za taj dan
Lat	Zemljopisna širina koordinata adrese klijenta, aplikacija koristi ovaj podatak kod izračunavanja udaljenosti između obiđenih klijenata
Lon	Zemljopisna dužina koordinata adrese klijenta, aplikacija koristi ovaj podatak kod izračunavanja udaljenosti između obiđenih klijenata

Tablica 3: Opis tablice KlijentiAplikacije

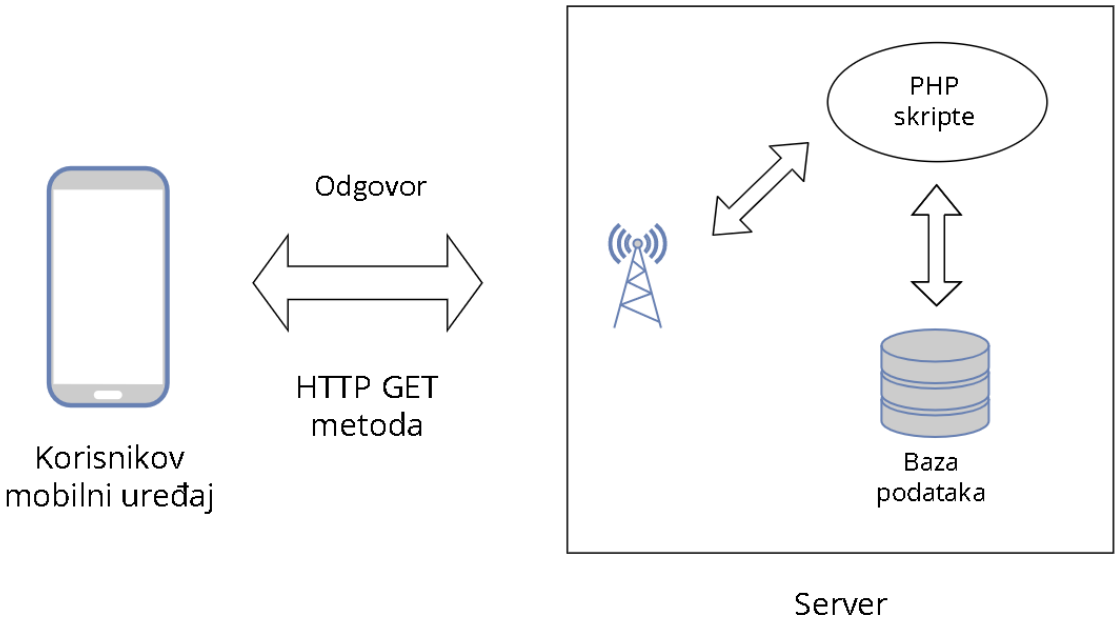
Tablica DaniApikacija Sadrži podatke o radnim danima zaposlenika, zaposlenici unose podatke za svaki radni dan te se na temelju tih podataka kreira finalni izvještaj	
Ime entiteta	**Opis**
Poc	Početno stanje brojila automobila, zaposlenik unosi stanje brojila na početku radnog dana
Zav	Završno stanje brojila automobila, zaposlenik također unosi stanje brojila automobila na kraju radnog dana, smisao unosa podataka o stanju brojila je kontroliranje mogućih oscilacija duljine rute koju će izračunati aplikacija na temelju unesenih podataka o obiđenim klijentima
Pkm	Prijeđeni kilometri u radnom danu, ovaj podatak aplikacija izračunava na temelju popisa obiđenih klijenata
Popis	Popis obiđenih klijenata u radnom danu, sadrži niz identifikacijskih brojeva klijenata aplikacije koje unosi korisnik, aplikacija će na temelju niza dohvatiti podatke od svakog klijenta i pomoću koordinata izračunati prijeđenu rutu u radnom danu zaposlenika

Tablica 4: Opis tablice DaniAplikacije

Kao što se vidi iz ERA modela baza nije pretjerano zahtjevna te ću ju kreirati na serveru netdomena.com na kojem već imam prethodno kreiran račun.

5.4. Pristup bazi podataka

Aplikacije (Android Studio [28] i Xamarin [29] verzija) će pristupati bazi podataka pomoću php skripti koje se također nalaze na serveru šaljući određenoj skripti HTTP GET zahtjev te će potom php skripta izvršiti određenu funkciju nad bazom i poslati nazad odgovor na korisnikov uređaj. Na slici ispod prikazana je planirana komunikacija uređaja s bazom podataka.



Slika 55: Prikaz komunikacije aplikacije s bazom podataka

Skripte kojima će se pristupati su sljedeće:

- Logiranje.php – prima korisničko ime i lozinku te provjerava postoji li zaposlenik u tablici ZaposleniciAplikacija čije korisničko ime i lozinka odgovara primljenim. Ako korisnik postoji vraća zaposlenikov Id.

```
$con->set_charset("utf8");
if (mysqli_connect_errno($con)) {
      echo "Failed to connect to MySQL: " . mysqli_connect_error();
   }
   $username = $_GET['username'];
   $password = $_GET['password'];
   $result = mysqli_query($con,"SELECT * FROM ZaposleniciAplikacija
   WHERE Korime='$username' and Lozinka='$password'");
if($result->num_rows > 0){
      while($res = $result->fetch_assoc()) {
            echo $res["id"]. "#" . $res["Ime"]. "#"
            . $res["Prezime"]. "#" . $res["Auto"]. "#"
            . $res["Registracija"];
         }
    }
    else{
        echo "No";
    }
  mysqli_close($con);
```

Isječak koda 13: Skripta za prijavu - Logiranje.php

- Sviklijenti.php – vraća sve podatke klijenata unesenih u tablicu KlijentiAplikacija te ih sortira abecednim redom na temelju prezimena

```
$con->set_charset("utf8");
   if (mysqli_connect_errno($con)) {
      echo "Failed to connect to MySQL: " . mysqli_connect_error();
   }
   $username = $_GET['username'];
   $result = mysqli_query($con,"SELECT * FROM KlijentiApikacija ORDER
   BY Prezime ASC");
    if($result->num_rows > 0){
      while($res = $result->fetch_assoc()) {
            echo $res["id"]. "#" . $res["Ime"]. "#"
                  . $res["Prezime"]. "#" . $res["Adresa"]. "#" .
                  $res["Lat"]. "#" . $res["Lon"]."///";
        }
    }
   mysqli_close($con);
```

Isječak koda 14: Skripta za dohvat svih klijenata - Sviklijenti.php

- SviDani.php – prima id prijavljenog zaposlenika i vraća sve podatke dana koje je unio zaposlenik iz tablice DaniAplikacije

```
   $con->set_charset("utf8");
   if (mysqli_connect_errno($con)) {
      echo "Failed to connect to MySQL: " . mysqli connect error();
   }
   $username = $_GET['id'];
   $result = mysqli_query($con,"SELECT * FROM DaniApikacija WHERE
   Zaposlenik='$username'");
   if($result->num_rows > 0){
      while($res = $result->fetch_assoc()) {
         echo $res["id"]. "#" . $res["Datum"]. "#" . $res["Poc"]. "#" .
         $res["Zav"]. "#" . $res["Pkm"]. "#" . $res["Popis"]. "#" .
         $res["Mjesec"]. "#" . $res["Godina"]."///";
       }
   }
   mysqli_close($con);
```

Isječak koda 15: Skripta za dohvat podataka o danima – SviDani.php

- UpdateDan.php – prima sve podatke o danu iz tablice i Id zaposlenika koji je promijenio podatke zatim u tablici DaniAplikacije briše zapis čiji podaci o datumu i zaposleniku odgovaraju primljenim podacima, potom kreira novi zapis.

```
$con->set charset("utf8");

if (mysqli_connect_errno($con)) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
$Datum = $_GET['Datum'];
$Poc = $_GET['Poc'];
$Zav = $_GET['Zav'];
$Pkm = $_GET['Pkm'];
$Popis = $_GET['Popis'];
$Mjesec = $_GET['Mjesec'];
$Godina = $_GET['Godina'];
$Zaposlenik = $_GET['Zaposlenik'];

$brisanje = "DELETE FROM DaniApikacija WHERE Datum='$Datum' AND
Zaposlenik=$Zaposlenik";

$umetanje = "INSERT INTO DaniApikacija (Datum, Poc, Zav, Pkm, Popis,
Mjesec, Godina, Zaposlenik) VALUES ('$Datum', '$Poc','$Zav',
'$Pkm','$Popis', $Mjesec,$Godina,$Zaposlenik)";
if (mysqli_query($con,$brisanje)) {
    if (mysqli_query($con,$umetanje)) {
        echo "Uspjeh";
    }
}
mysqli_close($con);
```

Isječak koda 16: Skripta za promjenu podataka u danu - UpdateDan.php

U ovom slučaju gdje je baza podataka jednostavna i nema potrebe za konstantnim osvježavanjem podataka u aplikaciji već samo kad korisnik unese nove podatke (par puta dnevno) ovakav način komunikacije aplikacije s bazom podataka funkcionirat će sasvim u redu.

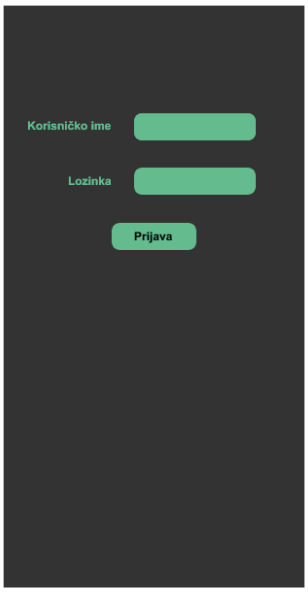
5.5. Dizajn sučelja

Kako bi olakšala programski dio razvoja aplikacije prvo ću kreirati skicu dizajna aplikacije, osobno sam u ovom koraku koristila Invision studio [25] alat, ali tehnički prvotni dizajn možete kreirati i na papiru. Aplikacija će se sastojati od 2 glavna korisnička sučelja i 3 sučelja koja će biti prikazana kao iskočni prozori.

5.5.1. Prijava

Početno sučelje aplikacije izvršavat će funkcionalnost prijave korisnika, kako bi uspješno funkcioniralo sastojat će se od sljedećih elemenata:

- 2 elementa za prikaz teksta
- 2 elementa za unos teksta u koje će korisnik unijeti korisničko ime i lozinku
- gumb za prijavu



Slika 56: Početni dizajn sučelja za prijavu – InVision Studio

Nakon što korisnik klikne na gumb za pokretanje aplikacije prvo će vidjeti sučelje za prijavu u kojem upisuje svoje podatke i klikne na gumb „Prijava“. Ako su podaci točni aplikacija ga prosljeđuje na sučelje za kreiranje dana.

5.5.2. Kreiranje dana

Nakon što se korisnik uspješno prijavio bit će proslijeđen na sučelje koje izvršava funkcionalnost unosa podataka za 1 dan te se sastoji od sljedećih elementa:

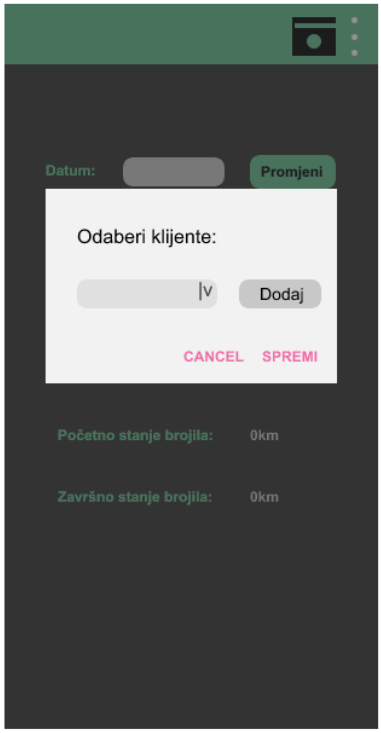
- 7 elementa za prikaz teksta
- 3 elementa za unos teksta u koje će korisnik unositi podatke
- Padajući izbornik u kojem će korisnik moći vidjeti popis obiđenih korisnika

Slika 57: Početni dizajn sučelja za kreiranje dana – InVision Studio

Prvo je potrebno odabrati željeni datum klikom na gumb „Promjeni“, potom se otvara iskočni prozor u kojem korisnik klikne željeni datum te ako je zadovoljan odabirom klikne na gumb „SPREMI“, ako nije klikne na gumb „CANCEL“.

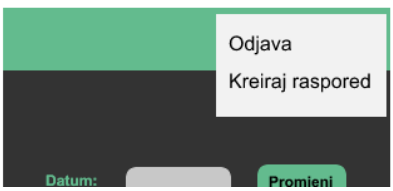
Slika 58: Početni dizajn sučelja za odabir datuma – InVision Studio

Klikom na bilo koji od dva gumba iskočni prozor se zatvara. Ispod datuma na glavnom sučelju prikazan je padajući izbornik koji će sadržavati redoslijed klijenata koji su obiđeni taj dan. Kako bi korisnik dodao novog klijenta u redoslijed kliknuti će na gumb „Dodaj klijenta“ te se otvara iskočni prozor koji će sadržavati sve dostupne klijente.



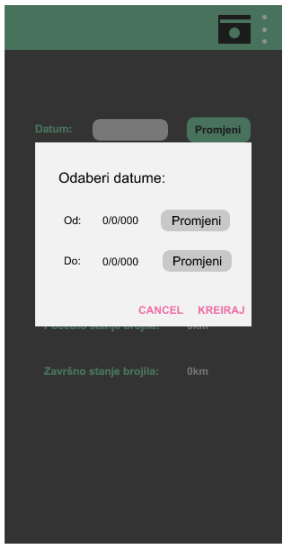
Slika 59: Početni dizajn sučelja za odabir klijenta – InVision Studio

Korisnik će iz padajućeg izbornika kliknuti na željenog klijenta i potom kliknuti na gumb „Dodaj“. Ako želi dodati još klijenata za taj dan, ponovit će postupak i kada ih je sve dodao kliknuti će na gumb „SPREMI“, ako nije zadovoljan odabirom kliknuti će na gumb „CANCEL“. Odabrani klijenti pojavit će se u padajućem izborniku na glavnom sučelju i ispisat će se trenutni broj prijeđenih kilometara na temelju redoslijeda klijenata. Ako nije zadovoljan redoslijedom, korisnik će moći iz padajućeg izbornika odabrati klijenta i izbrisati ga klikom na gumb „Izbriši“, na temelju brisanja preračunati će se broj prijeđenih kilometara. Potom korisnik može unijeti početno i završno stanje brojila. Kada je zadovoljan s unesenim podacima za dan, kliknuti će na ikonu za spremanju u desnom gornjem uglu, te će se sve promjene spremiti. Kako bi kreirao .pdf datoteku korisnik klikće na 3 bijele vertikalne točke te se otvara mali izbornik u kojem će odabrati opciju „Kreiraj raspored“, ako klikne na opciju „Odjava“ aplikacija ga vraća na sučelje za prijavu.



Slika 60: Početni dizajn izbornika– InVision Studio

Kada klikne na opciju za kreiranje rasporeda pojavljuje se novi iskočni prozor u kojem korisnik odabire od kojeg do kojeg datuma će u .pdf datoteci biti prikazani podaci.



Slika 61: Početni dizajn sučelja za kreiranje rasporeda – InVision Studio

Korisnik će odabrati datume tako da će kliknuti gumb „Promjeni“ te će se ponovo pojaviti iskočni prozor za odabir datuma kao i na glavnom sučelju. Kada je korisnik zadovoljan s odabirom datuma, kliknuti će na gumb „Kreiraj“ te će na temelju unesenih podataka aplikacija kreirati .pdf datoteku u memoriji samog uređaja.

5.6. Kreiranje korisničkih sučelja aplikacije

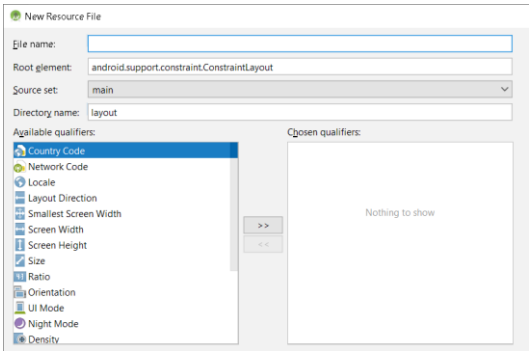
U ovom slučaju odlučila sam prvo kreirati sva sučelja te potom implementirati funkcionalnosti. Zbog lakšeg snalaženja i modifikacije sami kod aplikacije bit će raspoređen u dijelove prema funkcionalnosti koju izvršava (osobno mi je takav način rada najbolji) te ću ga i kreirati po tom redoslijedu.

5.6.1. Sučelje za prijavu

Prvo ću kreirati sučelje za prijavu, nakon toga sučelje za kreiranje dana te potom sučelja za odabir datuma i dodavanje klijenata. Sva sučelja bit će kreirana na temelju skice dizajna s eventualnim prilagodbama. Ono što je bitno zapamtiti kod dizajna bilo kakvih sučelja za mobilne aplikacije jest to da korisnik može mobilni uređaj držati vertikalno i horizontalno. Zbog toga ako sučelje ne sadrži samo jedan element(npr. pozdravnu poruku) preporučuje se kreirati dizajn sučelja za obje verzije.

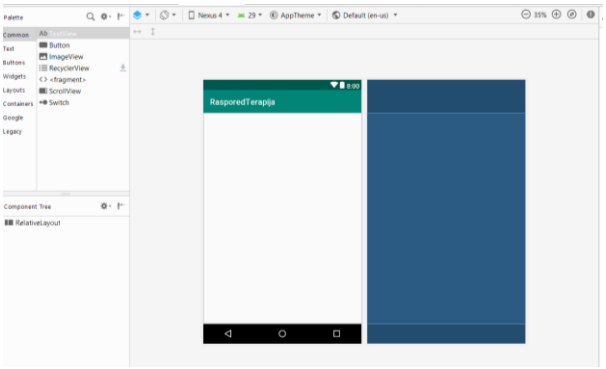
5.6.1.1. Android Studio

Kako bi kreirali novo korisničko sučelje u prozoru gdje je prikazana struktura projekta desnim klikom kliknemo na datoteku /res/layout i odaberemo New-> Layout resource file. Nakon toga otvara nam se sljedeći prozor:



Slika 62: Kreiranje sučelja – Android Studio

Upišite ime sučelja, odaberite kakav tip dizajna želite koristiti, iz osobnog iskustva preporučuje m RelativeLayout, pogotovo ako niste baš vješti s xml-om, ali ako sučelje nema previše elemenata, možete ostaviti i ConstraintLayout. Sva sučelja u ovoj aplikaciji koristit će RelativeLayout kao bazni element dizajna. Zatim odaberite uz koju aktivnost će sučelje biti vezano i ostavite ime datoteke gdje će sučelje biti spremljeno. Uzmite u obzir da razvojno okruženje kod kreiranja sučelja automatski kreira vertikalnu verziju dizajna, tako da ako kreirate horizontalnu iz izbornika opcija odaberite Orientation i iz izbornik Screen orientation odaberite Landscape. Ime datoteke će se promijeniti u layout-land, ostavite ga tako, a ostale podatke ispunite isto kao što ste i za vertikalnu verziju. Kada ste zadovoljni postavkama kliknite gumb Ok i sučelje će se kreirati. Kada je sučelje kreirano pojavit će se pregled početnog dizajna sučelja, odnosno prazno sučelje.



Slika 63: Pregled početnog dizajna sučelja – Android Studio

Ako želimo vidjeti xml kod dizajna, kliknemo na gumb Text u donjem lijevom kutu. S lijeve strane nalazi se izbornik elemenata koje možemo dodati na sučelje. Mislim da je najlakše kreirati zamišljeni dizajn tako da u pregledu sučelja iz izbornika dodamo željene elemente, postavimo ih na sučelje te se potom prebacimo na xml kod dizajna i tamo dodamo željene atribute elementima. Kako bi dodali elemente na sučelje, kliknite na element iz izbornika i povucite ga na sučelje. Kada kliknete na element, otvorit će vam se prozor a atributima određenog elementa, možete i ovdje promijeniti atribute umjesto u xml kodu. Ono što preporučuje m kod dizajniranja strukture elemenata(ako ste početnik u xml-u) jest da obratite pozornost na layout_align atribute jer oni omogućuju da se elementi poravnavaju na temelju drugih elemenata, a ako elemente poravnavate na temelju baznog elementa, može se dogoditi da se elementi raštrkaju ako se promjeni veličina uređaja.

Sljedeće na redu jest kreiranje dizajna za elemente na aplikaciji. Za svaki dizajn elementa u aplikaciji kreiramo xml datoteku u direktoriju /res /drawable te ga pridružujemo elementu na sučelju. Kreirat ćemo za početak dizajn gumba koji će imati 2 stanja, u stanju mirovanja i kada je kliknut. Kako bi to postigli, kreiramo 3 xml datoteke:

- button_enabled – predstavlja dizajn gumba u stanju mirovanja

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" android:padding="15dp">
    <solid android:color="#7ED5B3" />
    <corners
        android:bottomRightRadius="19dp"
        android:bottomLeftRadius="19dp"
        android:topRightRadius="19dp" />
    <padding android:top="14dp"
        android:bottom="10dp" />
    <stroke android:width="3dip" android:color="#64C9A1" />
</shape>
```

Isječak koda 17: Dizajn gumba u stanju mirovanja [28]

- button_pressed – predstavlja dizajn kliknutog gumba
- button – u ovoj datoteci definiramo 2 moguća stanja elementa (normalno i kliknuto), te za svako stanje pridružujemo zasebni dizajn

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:state_pressed="true"
        android:drawable="@drawable/button_pressed" />
    <item
        android:state_enabled="true"
        android:drawable="@drawable/button_enabled" />
</selector>
```

Isječak koda 18: Definiranje mogućih stanja gumba [28]

Nakon toga elementu na sučelju (u ovom slučaju gumbu za prijavu) pridružujemo kreirani dizajn.

Sljedeće ćemo kreirati dizajn za unos teksta, u ovom slučaju element neće imati dizajn za više stanja te će biti dovoljna samo jedna datoteka. U /res/drawable kreiramo novu xml datoteku koju ćemo nazvati edit_text i kreiramo željeni dizajn.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" android:padding="15dp">
    <solid android:color="#64C9A1" />
    <corners
        android:bottomRightRadius="10dp"
        android:bottomLeftRadius="10dp"
        android:topLeftRadius="10dp"
        android:topRightRadius="10dp" />
    <padding
      android:top="14dp"
      android:bottom="10dp"
      android:left="10dp"  />
    <stroke android:width="3dip" android:color="#7ED5B3" />
</shape>
```

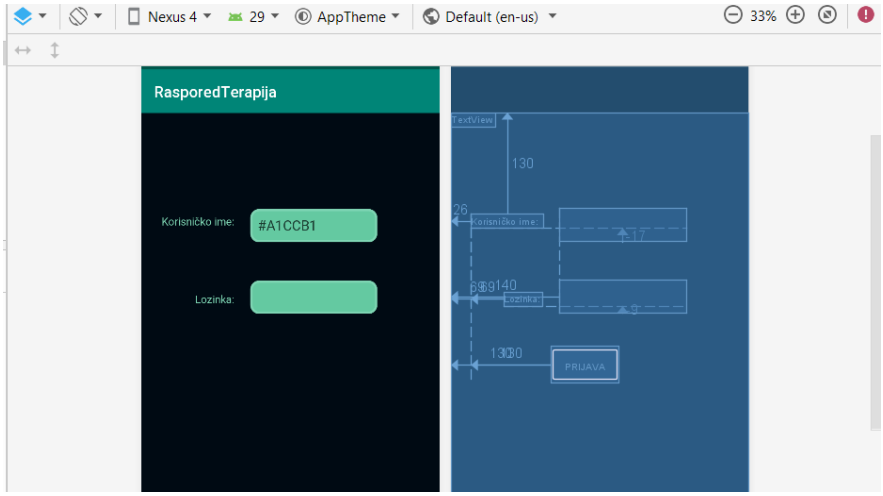
Isječak koda 19: Dizajn elementa za unos teksta [28]

Kada je dizajn završen pridružujemo ga svim elementima za unos teksta na sučelju. Ako želimo promijeniti boju pozadini aplikacije, definiramo ju u baznom elementu sučelja.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000A13"
```

Isječak koda 20: Promjena boje pozadine [28]

Također ćemo još promijeniti boju svim elementima za prikaz teksta na sučelju, tako da se uklapaju u dizajn. Ako se sada prebacite na pregled prikaza sučelja, ono će izgledati kao što je prikazano na slici 64. Kao što vidimo u pregledu nije prikazan odgovarajući dizajn gumba, ali nema veze jer ponekad ako je u pridruženom dizajnu elementa definirano više stanja, element se neće prikazati u pregledniku, ali normalno će se prikazivati na emulatoru ili mobilnom uređaju, tako da ako se dogodi ova situacija, samo pokrenite aplikaciju i pokazat će se je li stvarno u pitanju greška ili samo nemogućnost prikaza u pregledniku.

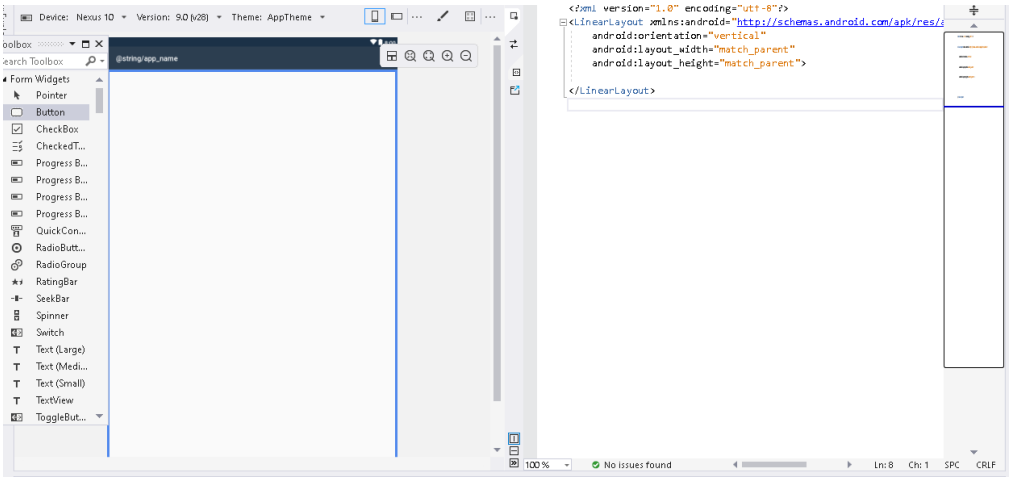


Slika 64: Prikaz dizajna sučelja – Android Studio

Kada smo završili s vertikalnim dijelom dizajna, najjednostavnije je kopirati xml kod sučelja i zalijepiti u datoteku s horizontalnim dizajnom ako je već kreirana(ako nije slijedite korake iz prethodnog poglavlja). Android Studio u prozoru za pregled strukture projekta datoteke s horizontalnim dizajnom označava sa (land). Nakon što ste zalijepili xml kod, najbolje je pokrenuti aplikaciju na emulatoru ili uređaju i vidjeti kako izgleda u horizontalnom prikazu, odnosno koje promjene treba napraviti u dizajnu.

5.6.1.2. Xamarin

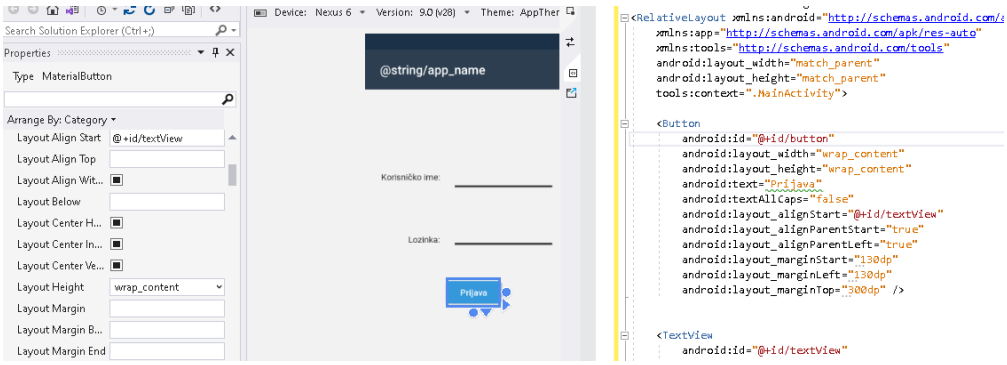
Nakon što je završen proces kreiranja sučelja otvara se preglednik dizajna sučelja i xml kod dizajna. Ako mijenjamo xml kod dizajn sve promjene će se automatski prikazati u pregledniku sučelja.



Slika 65: Preglednik sučelja – Xamarin

S lijeve strane nalazi se gumb Toolbox koji otvara izbornik elemenata za sučelje. Xamarin će automatski kao bazni element sučelja postaviti LinerLayout, koji ćemo ručno u xml kodu zamijeniti za RelativeLayout i povezat ćemo sučelje s početnom aktivnosti.

Kao i u prethodnom postupku elemente na sučelje dodajemo tako da iz izbornika kliknemo na element i povučemo ga na pregled sučelja(ako koristite RelativeLayout ponekad se element ne želi kreirati u sučelju tako u izborniku umjesto da odvučete element samo 2 puta kliknite na njega i element će se kreirati). U pregledniku sučelja ne možete mijenjati poziciju elemenata tako da kliknete element i odvučete ga na željeno mjesto(kao što Android Studio dozvoljava). Kada su dodani svi željeni elementi, u xml kodu definiramo njihove pozicije. U ovom koraku dizajna vrijede ista pravila kao i u Android Studiju (obratiti pozornost na layout_align atribute). Xml kod možete mijenjati ručno ili u prozoru a atributima koji se otvara nakon što kliknete na element(kao što se vidi na slici ispod), u svakom slučaju morate biti upoznati s funkcijama atributa.

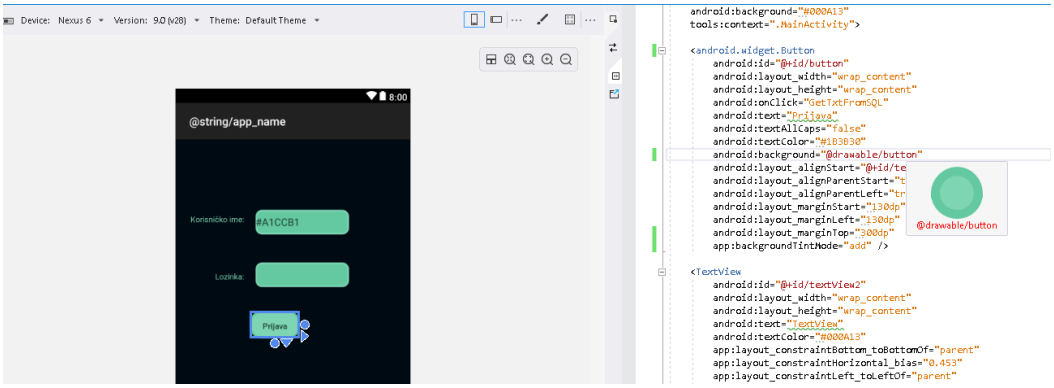


Slika 66: Prikaz dizajna sučelja – Xamarin

Nakon što ste zadovoljni strukturom sučelja kreiramo dizajn za svaki element. Kreiranje dizajna u Xamarin-u ima vrlo sličan princip kao i Android Studio, u direktoriju /Resources/drawable kreiramo xml datoteku, u njoj definiramo željeni dizajn te ga pridružujemo elementu na sučelju. Kao i u prethodnom primjeru kreirat ćemo sljedeće datoteke:

- button_enabled – predstavlja dizajn gumba u stanju mirovanja
- button_pressed – predstavlja dizajn kliknutog gumba
- button – u ovoj datoteci definiramo da će element imati 2 stanja(normalno i kliknuto), te za svako stanje pridružujemo zasebni dizajn
- edit_text - dizajn za element unosa teksta

Xml kod dizajna je isti kao i prije. Kada je dizajn završen, pridružujemo ga elementima kao i prošli puta. U ovom slučaju Xamarin ima jednu prednost, a to je kada pridružimo dizajn elementu u xml kodu automatski vidimo kako dizajn izgleda i bez da se prebacujemo na preglednik dizajna kao što se vidi na slici 67. Također preglednik dizajna nema problema s prikazom gumba.



Slika 67: Promjena boje elementa sučelja – Xamarin

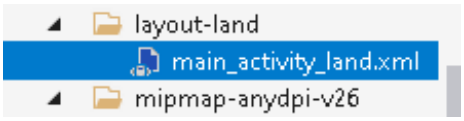
Kod kreiranja dizajna sučelja obratite pozornost na zadanu temu u pregledniku dizajna, Xamarin će automatski temu postaviti na AppCompat temu, prebacite je na Default temu. Ako preskočite taj korak ponekad neki elementi(npr.gumb) neće htjeti poprimiti zadani dizajn iako ste ga pravilno definirali. Kada ste promijenili temu pojavit će vam se greške kod kompajliranja projekta jer je promjena teme izbrisala definirane boje koje se koriste u baznoj temi. Kako bi popravili tu grešku otvorite datoteku /Resources/values/styles.xml. U datoteci izbrišite sve definirane elemente unutar elementa <style> kako bi kod izgledao kao u isječku koda 21.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">
     </style>
</resources>
```

Isječak koda 21: Promjena teme dizajna [29]

Nakon toga očistite i kompajlirate projekt, greške bi trebale nestati i elementi će poprimiti zadani dizajn, u ovoj datoteci možete i kreirati zasebnu temu. Također bitno je napomenuti da kod kreiranja gumba element ne definirate kao samo <Button> već kao <android.widget.Button>, jer ponekad kod kompajliranja projekta može se dogoditi greška i

Xamarin neće prepoznati <Button> kao element gumba. Kada je završen dizajn za vertikalni prikaz, kreirajte dizajn za horizontalni prikaz.



Slika 68: Horizontalni prikaz sučelja – Xamarin

Prvo u direktoriju /Resources kreirajte mapu layout-land, u njoj kreirajte xml datoteku pod nazivom main_activity_land. U njoj ćemo definirati dizajn za horizontalni prikaz. Kao i prije najlakše je kopirati dizajn za vertikalni prikaz i napraviti preinake. Također je bitno da svi elementi imaju isti id kao i u horizontalnom prikazu kao što je prikazano u isječku koda 22.

```
if (widthInDp > heightInDp)
    {
   v1 =  this.LayoutInflater.Inflate(Resource.Layout.activity_kreiranja_dana_land,
   null);
    }
else
    {
    v1 = this.LayoutInflater.Inflate(Resource.Layout.activity_kreiranje_dana, null);
    }
SetContentView(v1);
```

Isječak koda 22: Odabir dizajna za horizontalni ili vertikalni prikaz [29]

Nakon kreiranih preinaka na dizajnu, otvorite aktivnost koja je pridružena sučelju i u onCreate funkciji(funkcija koja se prva pokreće) provjerite širinu i visinu ekrana, ako je širina veća od visine otvorit će se sučelje za horizontalni prikaz, inače se otvara sučelje za vertikalni prikaz.

5.6.2. Sučelje za kreiranje dana

Ovo sučelje će omogućiti korisniku kreiranja rasporeda obiđenih klijenata za određeni dan, u sljedećim poglavljima bit će pojašnjen postupak kreiranja sučelja.

5.6.2.1. Android Studio

Kreirajte novo sučelje pod imenom activity_kreiranje_dana, zatim promijenite bazni element sučelja u RelativeLayout i postavite sve željene elemente na sučelje.

Postupak je isti kao i za sučelje prijave, osim što u ovom sučelju postoji element padajućeg izbornika tako da za njega moramo kreirati posebni dizajn kako bi odgovarao ostatku. U direktoriju /res/drawable kreiramo novu xml datoteku koju ćemo nazvati spinner i definirati dizajn za padajući izbornik kao u isječku koda 23.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#EEF2F5" />
    <corners android:radius="13dp" />
    <stroke
        android:width="1dp"
        android:color="@android:color/darker_gray" />
</shape>
```

Isječak koda 23: Dizajn elementa padajućeg izbornika [28]

Ovaj dizajn nećemo direktno primijeniti na element već ćemo ga ugnijezditi u zasebni RelativeLayout element na koji ćemo primijeniti dizajn. Na ovaj način primjene dizajna možemo dodati sliku strelice na kraj padajućeg izbornika.

Slika 69: Padajući izbornik

Željenu sliku spremite u direktorij /res/drawable pod imenom strelica. Zatim u elementu RelativeLayout u kojem se nalazi padajući izbornik dodajte element <ImageView> koji će služiti za prikaz slike strelice, na element povežite sliku kao što je prikazano u isječku koda 24.

```
<ImageView
    android:layout_width="30dp"
    android:layout_height="30dp"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"
    android:layout_gravity="center"
    android:src="@drawable/strelica" />
```

Isječak koda 24: Prikaz strelice u padajućem izborniku [28]

Kao i prije gumbi kojima je promijenjen dizajn se ne vide, ali normalno se pokazuju nakon pokretanja aplikacije. Sada je na redu vertikalni prikaz sučelja, postupak je isti kao i kod prethodnog sučelja (kreiramo sučelje za horizontalni prikaz s istim nazivom kao i za vertikalni).

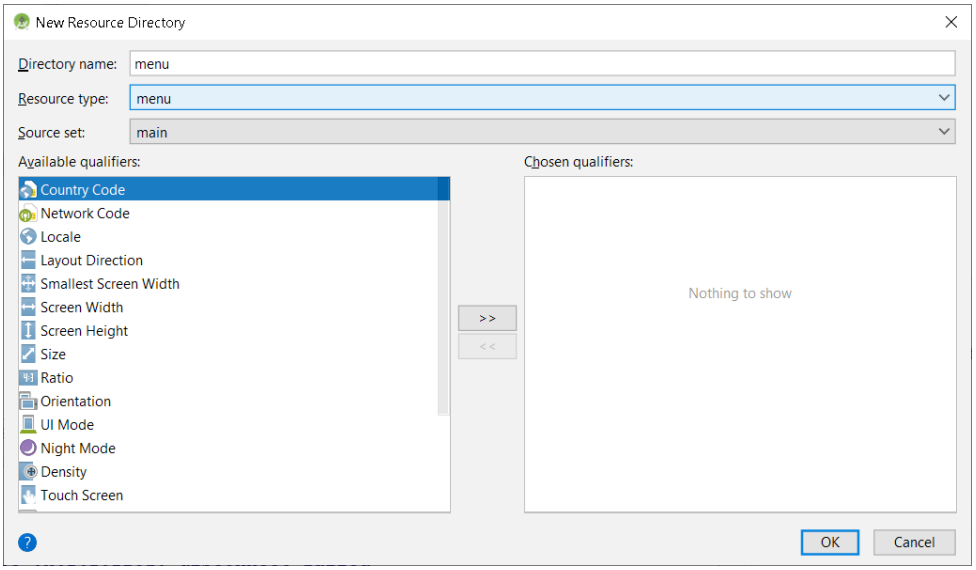
Napravimo neke preinake tako da dizajn odgovara dimenzijama, ali u ovom slučaju bazni element moramo ugnijezditi u <ScrollView> element koji je prikazan u isječku koda 25, kako bi korisnik mogao imati prikaz nad cijelim sučeljem i u vertikalnom prikazu.

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#000A13"
    android:fillViewport="true">
```

Isječak koda 25: Dizajn vertikalnog prikaza sučelja [28]

Ako izostavimo ovaj element a sučelje sadrži previše elemenata korisnik će moći vidjeti samo one koji stanu na ekran te neće imati pristup ostalim. Na sučelju za kreiranje novog dana dodat ćemo još i izbornik kako bi se korisnik mogao lakše snalaziti u aplikaciji. Izbornik će se nalaziti na vrhu sučelja. Te će sadržavati poveznice na spremanje dana, kreiranje rasporeda i odjavu korisnika.

Kako bi kreirali element menu- a kliknemo desnim klikom na direktorij /res te odaberemo New Resource Directory. Otvorit će se prozor kao na slici 70.



Slika 70: Kreiranje novog resursa - Android Studio

Iz padajućeg izbornika odaberite menu i kliknite gumb Ok. Trebao bi se kreirati direktorij pod nazivom menu. Kliknite desnim klikom na direktorij i odaberite New-> Menu resource file te ga nazovite menu. U direktoriju će se kreirati xml datoteka u kojoj ćemo definirati strukturu izbornika kao što je prikazano u isječku koda 26.

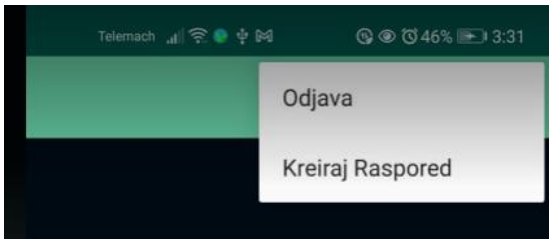
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="#7ED5B3">
    <item
        android:id="@+id/spremi"
        android:icon="@drawable/save"
        android:title="Spremi"
        app:showAsAction="ifRoom"/>
    <item
        android:id="@+id/odjavi"
        android:title="Odjava"
        app:showAsAction="never"/>
    <item
        android:id="@+id/kreirajRas"
        android:title="Kreiraj Raspored"
        app:showAsAction="never"/>
</menu>
```

Isječak koda 26: Dizajn izbornika [28]

Želimo da korisniku bude uvijek dostupna poveznica na spremanje dana, zbog toga je kreirana posebna ikona za spremanje. Ikona je pohranjena u direktoriju /res/drawable te je pridružena poveznici za spremanje. Kako bi se izbornik uspješno prikazivao u aktivnosti koja je pridružena sučelju(KreiranjeDana) definiramo onCreateOptionsMenu() koja je prikazana u isječku koda 27 te pomoću nje prikazujemo izbornik.

```
public boolean onCreateOptionsMenu(Menu menu){
    MenuInflater inf=getMenuInflater();
    inf.inflate(R.menu.menu, menu);
    return true;
}
```

Isječak koda 27: Prikaz izbornika [28]

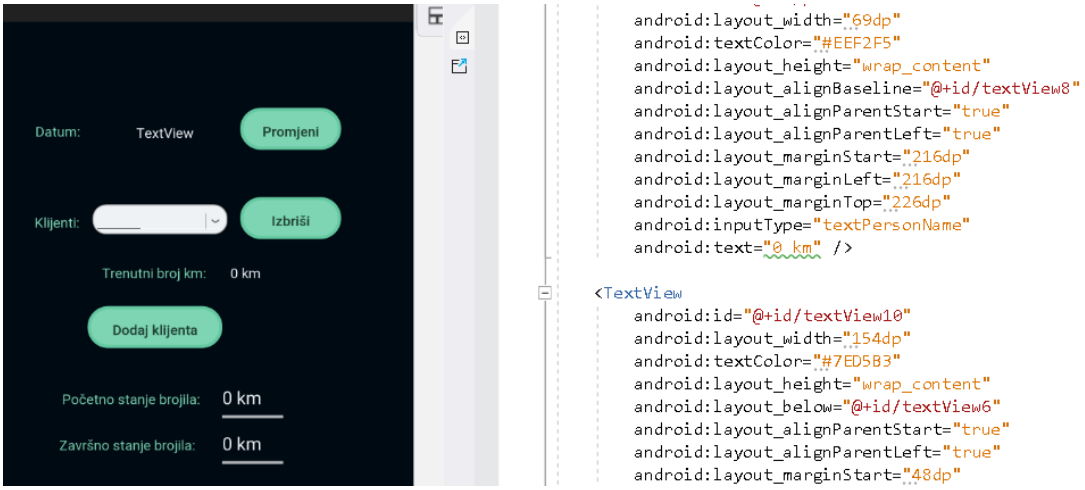


Slika 71: Izbornik aplikacije – Android Studio

Nakon pokretanja aplikacije u sučelju za kreiranje dana trebao bi se pojaviti funkcionalan izbornik. Izbornik se prikazuje svaki puta kad korisnik klikne na ikonu za otvaranje.

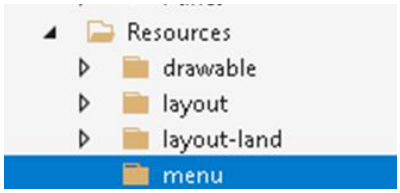
5.6.2.2. Xamarin

U direktoriju /Resources/layout kreiramo novu xml datoteku pod nazivom activity_kreiranja_dana. Dodajte sve željene elemente i u xml kodu definirajte strukturu sučelja(xml kod je isti kao i kod postupka za Android Studio, osim što kod definiranja elementa gumba moramo pripaziti da ga kreiramo kao <android.widget.Button>). Kako bi mogli dodati sliku strelice smještamo ju u direktorij /Resource/drawable. Nakon svih promjena u pregledniku dizajna sučelje bi trebalo izgledati otprilike kao na slici 72.



Slika 72: Sučelje za kreiranje dana - Xamarin

Kako bi kreirali horizontalni dizajn u direktoriju Resources/layout-land kreiramo xml datoteku pod nazivom activity_kreiranja_dana_land. U nju kopiramo vertikalni dizajn i sve ugnijezdimo u <ScrollView> element(isto kao i u Android Studiju). Napravimo preinake u marginama elementa kako bi dizajn odgovarao horizontalnom prikazu. U pregledniku dizajna možemo vidjeti kako će otprilike izgledati sučelje. Kao i u sučelju za logiranje moramo u pridruženoj aktivnosti provjeravamo veličinu ekrana na kojoj se pokreće aplikacija i na temelju toga odabiremo sučelje. Sljedeći korak je dodavanje izbornika, u Visual Studiju, morate kreirati prazni direktorij pod nazivom /resources/menu. U tom direktoriju kreirajte xml datoteku tipa menu.



Slika 73: Direktorij menu - Xamarin

Nakon kreiranja datoteke možete upotrijebiti isti xml kod za menu kao i u Android Studiju. Također u povezanoj aktivnosti morate dodati funkciju OnCreateOptionsMenu() koja je prikazana u isječku koda 28.

```
public override bool OnCreateOptionsMenu(IMenu menu) {
            MenuInflater.Inflate(Resource.Menu.menu, menu);
            return true;
  }
```

Isječak koda 28: Funkcija za prikaz izbornika [29]

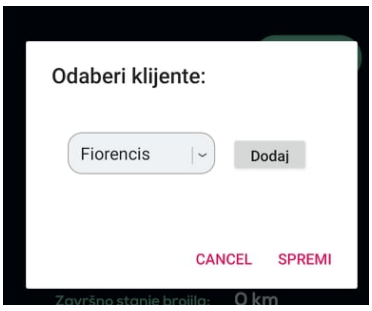
Ako je sve uspješno kompajlirano, aplikacija bi se trebalo uspješno pokrenuti i prikazati izbornik na drugom sučelju.

5.6.3. Sučelje za odabir klijenata

Kao što je bilo spomenuto na početku korisniku se mora omogućiti bilježenje obiđenih klijenata po danu. Trenutno obiđeni klijenti prikazivat će se u padajućem izborniku na sučelju za kreiranje dana. Kako bi korisnik mogao dodavati klijente bit će kreirano sučelje koje će sadržavati popis svih klijenata te će putem njega korisnik moći izabrati i dodati klijenta.

5.6.3.1. Android Studio

U direktoriju /res/layout kreirat ćemo novu xml datoteku pod nazivom dodavanje_klijenata koja će sadržavati padajući izbornik i jedan gumb. Trenutno nije bitno što elementi nisu centrirani jer će sučelje biti prikazano kao iskočni prozor nakon što korisnik klikne na gumb za dodavanje klijenta. Nakon što se pokrene aplikacija sučelje će izgledati kao na slici 74.

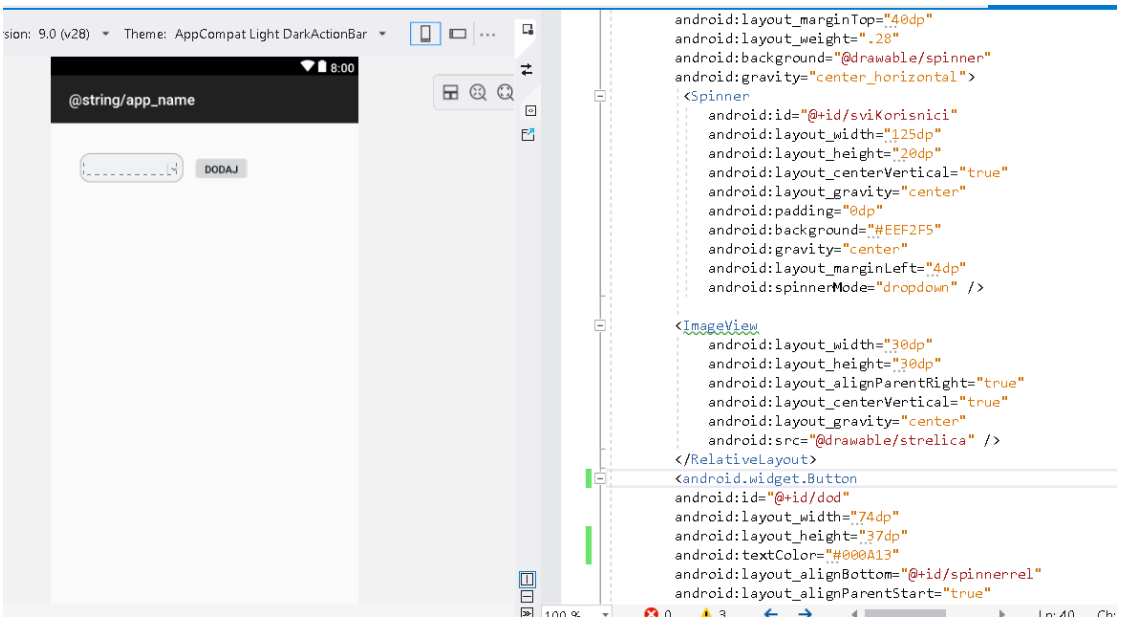


Slika 74: Sučelje za odabir klijenata - Android Studio

U ovom slučaju nema potrebe dvije vrste dizajna (vertikalni i horizontalni) pošto iskočni prozor i u jednom i u drugom obliku ne mijenja oblik.

5.6.3.2. Xamarin

U direktoriju /Resources/layout kreirat ćemo xml datoteku dodavanje_klijenta. Kao i prijašnja verzija sastojat će se od padajućeg izbornika i jednog gumba.



Slika 75: Prikaz sučelja za odabir klijenta - Xamarin

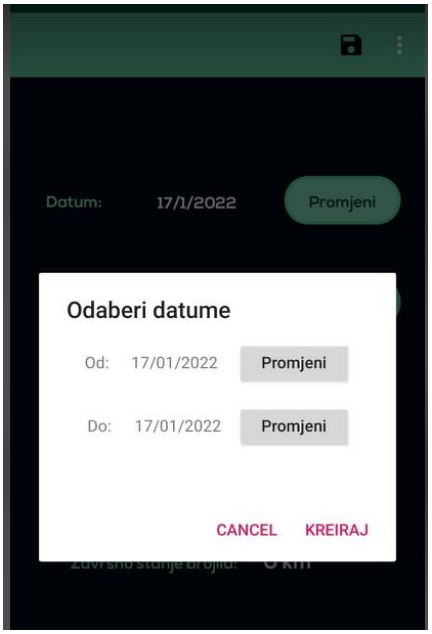
Za sada izgled sučelja nećemo puno mijenjati. Kada se aplikacija pokrene sučelje bi trebalo izgledati isto kao i u Android Studiju.

5.6.4. Sučelje za odabir datuma za kreiranje rasporeda

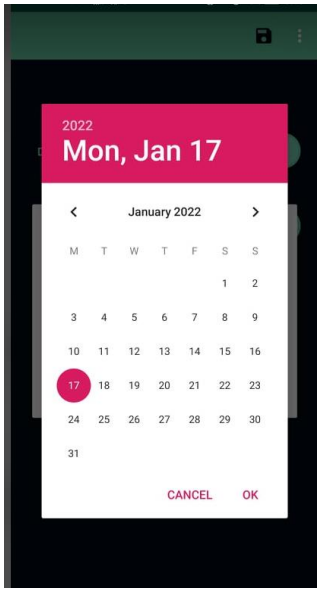
Nakon što je korisnik unio podatke za sve potrebne dane, treba mu biti omogućena funkcionalnost odabira datuma za kreiranje .pdf datoteke rasporeda. Sučelje će se također prikazivati kao iskočni prozor nakon što korisnik u izborniku odabere opciju Kreiraj raspored.

5.6.4.1. Android Studio

U direktoriju /res/layout/ kreirajte datoteku raspored.xml. Glavni elementi sučelja su 2 elementa za prikaz u kojima će se ispisati odabrani datumi i 2 gumba pomoću kojih će korisnik mijenjati datume. Nakon pokretanja aplikacije trebalo bi izgledati kao na slici 76.



Slika 76: Sučelje za odabir datuma za kreiranje rasporeda - Android Studio

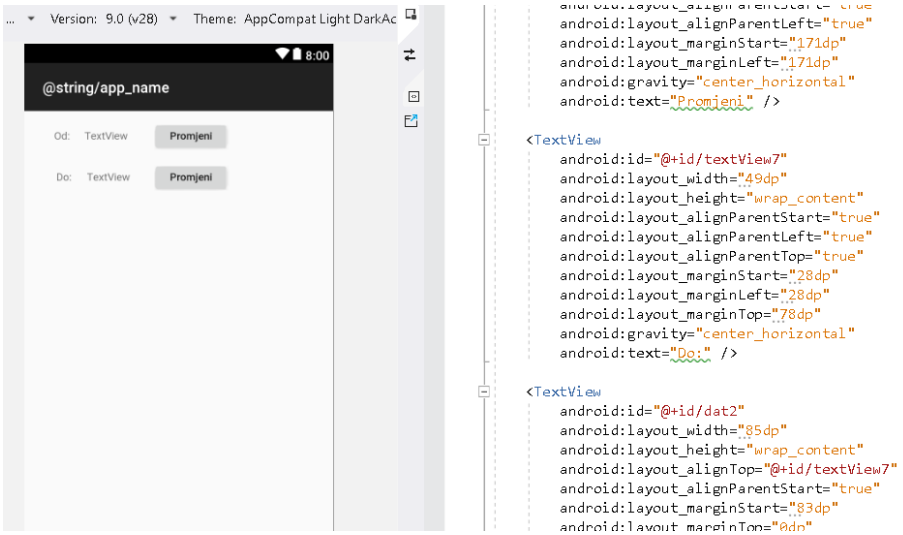


Slika 77: Odabir datuma - Android Studio

Korisnik će jednostavno moći odabrati željene datume i za te datume bit će kreiran raspored. Kasnije će se još dodati funkcionalnost otvaranja kalendara za odabir datuma kao što je prikazano na slici 77.

5.6.4.2. Xamarin

U direktoriju /Resources/layout kreiramo datoteku raspored.xml. Postavimo i pozicioniramo sve željene elemente na sučelje.



Slika 78: Sučelje za odabir datuma za kreiranje rasporeda - Xamarin

Nakon pokretanja aplikacije, sučelje bi trebalo izgledati isto kao i verzija koja je kreirana u Android Studiju. Što se tiče same izrade dizajna sučelja mogu zaključiti da nema nekih velikih razlika u samom procesu osim što je potrebno u Xamarin-u ručno specificirati kada se mijenja dizajn iz vertikalnog u horizontalni.

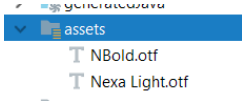
5.6.5. Promjena fonta

Ako želite u aplikaciji postaviti specifičan font(koji nije dostupan), morate ga prvo skinuti. Na Internetu možete naći mnogo besplatnih fontova na stranicama poput freefonts. Osobno sam skinula dvije verzija Nexa fonta(Bold i Light). Promjena fonta neće se vidjeti ni u jednom pregledniku dizajna, morat će te pokrenuti aplikaciju da bi promjene došle do izražaja.

5.6.5.1. Android studio

Kako bi mogli koristiti fontove u aplikaciji, datoteke spremite u direktorij /assets kao što je prikazano na slici 79 te zatim u aktivnosti koja je pridružena sučelju definirajte elemente putem njihovog id-a iz sučelja i definirajte željeni font.

Nakon toga upotrijebite funkciju setTypeface kako bi svim željenim elementima promijenili font kao što je prikazano u isječku koda 29.



Slika 79: Odabir novog fonta - Android Studio

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    Typeface type = Typeface.createFromAsset(getAssets(),"NBold.otf");
}
```

Isječak koda 29: Promjena fonta u aplikaciji [28]

Nakon što pokrenete aplikaciju font bi trebao biti promijenjen.

5.6.5.2. Xamarin

Sličan postupak koristi se i u Xamarin-u. Prvo datoteke fontova smjestimo u direktorij /Assets.

```
protected override void OnCreate(Bundle savedInstanceState) {
            base.OnCreate(savedInstanceState);
           Xamarin.Essentials.Platform.Init(this, savedInstanceState);
            var rezolucija = Resources.DisplayMetrics;
            var widthInDp = ConvertPixelsToDp(rezolucija.WidthPixels);
            var heightInDp = ConvertPixelsToDp(rezolucija.HeightPixels);
            if (widthInDp > heightInDp)
            {
              v1 = this.LayoutInflater.Inflate(Resource.Layout.main_activity_land,
              null);
            }
            else {
                v1 = this.LayoutInflater.Inflate(Resource.Layout.activity_main, null);
            }
            SetContentView(v1);
            Typeface font = Typeface.CreateFromAsset(Assets, "NBold.otf");
}
```

Isječak koda 30: Promjena fonta u aplikaciji [29]

Zatim otvaramo aktivnost pridruženu sučelju i u funkciji onCreate(ako želimo da se font primjeni odmah nakon pokretanja aplikacije) definiramo elemente sučelja i željeni font te potom elementima promijenimo font kao što je prikazano u isječku koda 30.

5.7. Implementacija funkcionalnosti aplikacije

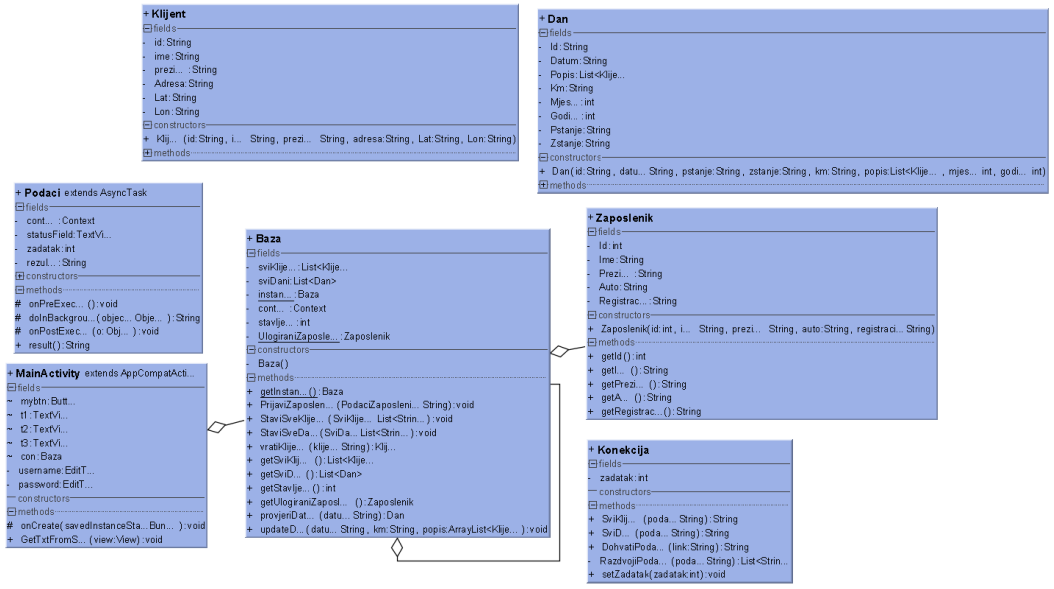
Nakon dizajna sučelja prva funkcionalnost koju ću kreirati bit će prijava korisnika u aplikaciju, potom mogućnost korisnika da odabere željeni datum i dodavanje te brisanje korisnika iz popisa za taj dan. Sljedeća funkcionalnost je izračun i ispis prijeđenih kilometara za taj dan i spremanje unesenih podataka u bazu. Zadnja funkcionalnost će biti generiranje .pdf dokumenta na temelju unesenih podataka. Također će uz svaku funkcionalnost biti generiran UML dijagram klasa pomoću alata Visual Paradigm (Android Studio) i Visual Studio Class Designer (Visual Studio). Obje verzije aplikacije iz kojih su uzeti isječci koda nalaze se na GitHub-u te su linkovi navedeni u literaturi (Android Studio [28] i Xamarin [29]).

5.7.1. Prijava

Kada korisnik na početnom sučelju upiše korisničko ime i lozinku te klikne na gumb za prijavu, aplikacija mora provjeriti ispravnost upisanih podataka. Ako su podaci točni, iz baze se preuzimaju podaci o klijentima i danima koje je korisnik prethodno unio i korisnika se preusmjeruje na sučelje za kreiranje dana. Ako podaci nisu točni korisniku se ispisuje obavijest.

5.7.1.1. Android Studio

Klase će biti kreirane na temelju dijagrama klasa prikazanog na slici 80.



Slika 80: Dijagram klasa implementacije funkcionalnosti prijave - Android Studio

Kako bi se podaci mogli spremati i provjeravati prvo ćemo kreirati opisne klase za dane, klijente i prijavljenog zaposlenika. Cilj ovih klasa jest uredna pohrana podataka koje će aplikacija primiti iz baze.

```
public class Klijent {

    private String id;
    private String ime;
    private String prezime;
    private String Adresa;
    private String Lat;
    private String Lon;

}
```

Isječak koda 31: Klasa Klijent [28]

```
public class Dan {
    private String Id;
    private String Datum;
    private List<Klijent> Popis=new ArrayList<Klijent>();
    private String Km;
    private int Mjesec;
    private int Godina;
    private String Pstanje;
    private String Zstanje;

}
```

Isječak koda 32: Klasa Dan [28]

```
public class Zaposlenik {
    private int Id;
    private String Ime;
    private String Prezime;
    private String Auto;
    private String Registracija;

}
```

Isječak koda 33: Klasa Zaposlenik [28]

Klasama koje su prikazane u gore navedenim isječcima koda će kasnije biti dodani get- eri i set –eri za elemente. Nakon toga kreiramo singleton klasu pod nazivom Baza čija je početna verzija prikazana u isječku koda 34 i sadrži liste klijenata i dana, također će u njoj biti pohranjen id korisnika koji se uspješno prijavio.

```
public class Baza {
    private List<Klijent> sviKlijenti;
    private List<Dan> sviDani;
    private static Baza instanca = null;
    private Context context;
    private int stavljeno=0;
    private static Zaposlenik Ulogiranizaposlenik;
```

```
public static Baza getInstance() {
    if (instanca == null) {
        instanca = new Baza();
    }
    return instanca;
}

private Baza() {
    sviKlijenti=new ArrayList<Klijent>();
    sviDani=new ArrayList<Dan>();

}
```

Isječak koda 34: Početna verzija klase Baza [28]

Klasa Baza je singleton zbog toga što se nakon uspješne prijave korisnika iz baze na serveru dobivaju svi podaci o klijentima i danima od prijavljenog korisnika kako bi se podaci za izradu rasporeda i dodavanje klijenata mogli dobivati iz konstantnog lokalnog izvora, bez potrebe da svaki puta aplikacija komunicira s bazom. U istom direktoriju kreiramo klase pod nazivom Podaci i Konekcija. U klasi Podaci koja je prikazana u isječku koda 35 ovisno o zadanom zadatku pomoću klase Konekcija dohvaćaju se i čitaju podaci iz baze. Također je bitno napomenuti da u ovom slučaju klasa Podaci će djelovati kao asinkrona dretva.

```
public class Podaci extends AsyncTask{

    public Podaci(Context context, TextView statusField, int zadatak){}
    protected void onPreExecute(){
        }
        @Override
        protected String doInBackground(Object[] objects) {
            String link=objects[0].toString();
            Konekcija con= new Konekcija();
            String rez="";
            switch(this.zadatak){
                case 1:
                    rez=con.DohvatiPodatke(link);
                    rezultat=rez;
                    break;
                case 2:
                    String podaci=con.DohvatiPodatke(link);
                    String d=con.SviKlijenti(podaci);
                    rez=d;
                    break;
                case 3:
                    String podaci2=con.DohvatiPodatke(link);
                    String dani=con.SviDani(podaci2);
                    rez=dani;
                    break;
                case 4:
                    String status=con.DohvatiPodatke(link);
                    System.out.println(status);
                    rez=status;
                    break;
            }
```

```
            return rez;
        }
        @Override
        protected void onPostExecute(Object o) {
            rezultat=o.toString();
        }
        public String result(){
            return rezultat;
        }
    }
```

Isječak koda 35: Klasa Podaci [28]

Ovisno o zadatku klasi Konekcija se prosljeđuju url poveznice na php skripte. Ako se dohvaća više podataka(svi klijenti ili svi dani korisnika) u klasi Konekcija kao što je prikazano u isječku koda 36 podaci se razdvajaju i spremaju u odgovarajuće liste koje se nalaze u klasi Baza.

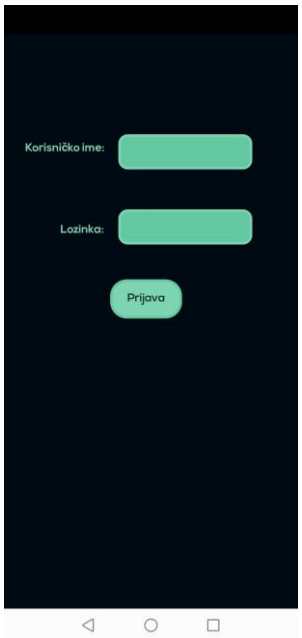
```
public String DohvatiPodatke(String link){
    try{
        URL url = new URL(link);
        HttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet();
        request.setURI(new URI(link));
        HttpResponse response = client.execute(request);
        BufferedReader in = new BufferedReader(new
        InputStreamReader(response.getEntity().getContent()));
        StringBuffer sb = new StringBuffer("");
        String line="";
        while ((line = in.readLine()) != null) {
            sb.append(line);
            break;
        }
        in.close();
        return sb.toString();
    } catch(Exception e){
        return "error";
    }
}
```

Isječak koda 36: Funkcija za dohvat podataka iz klase Konekcija [28]

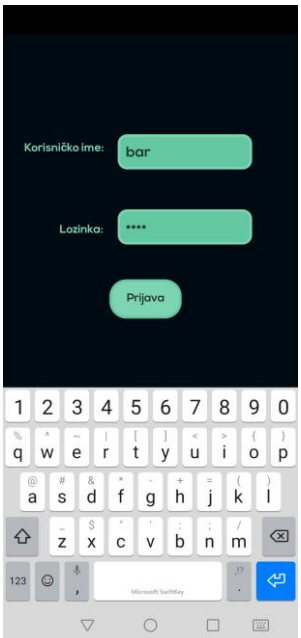
Ako su podaci uspješno dohvaćeni i pohranjeni, klasi Podaci vraća se potvrdni string. U klasi koja upravlja početnom aktivnošću (MainActivity) definiramo funkciju koja će se pozvati nakon što korisnik klikne na gumb za prijavu. Prvo se u funkciji dohvaćaju podaci iz elementa za unos teksta. Zatim se provjerava jesu li upisani podaci ispravni te ako nisu korisniku se ispisuje obavijest. Ako korisnik postoji u bazi pomoću klase Podaci dohvaća se popis svih klijenata i podaci svih dana koje je taj korisnik upisao.

```
public void GetTxtFromSQL(View view){
    String u = username.getText().toString();
    String p = password.getText().toString();
    String link1 =
"http://crofi.com/assets/assets/images/RasporedBaza/Logiranje.php?username="+u+
"&password="+p;
    String link2 =
"http://crofi.com/assets/assets/images/RasporedBaza/Svilijenti.php";
    Object result= null;
    String rezultat=null;
    Podaci p1=new Podaci(this,t1, 1);
    try {
        result=p1.execute(link1).get();
        rezultat=p1.result();
        if(rezultat.equals("error")||rezultat.equals("No")||rezultat==null) {
            Toast.makeText(this, "Krivi podaci", Toast.LENGTH_LONG).show();
        }
        else{
            int idKor=Integer.parseInt(rezultat);
            con.setUlogiranizaposlenik(idKor);
            String link3 =
"http://crofi.com/assets/assets/images/RasporedBaza/SviDani.php?id="+idKor;
            new Podaci(this, t1, 2).execute(link2).get();
            new Podaci(this, t1, 3).execute(link3).get();
            Intent intent = new Intent(this, KreiranjeDana.class);
            startActivity(intent);
        }
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }}
```

Isječak koda 37: Funkcija za preuzimanje podataka iz klase MainActivity [28]



Slika 81: Početno stanje sučelja za prijavu [28]



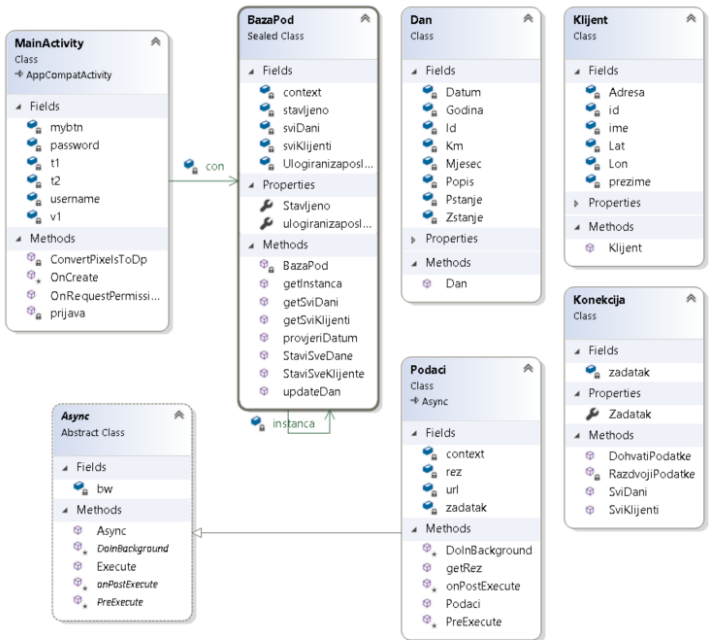
Slika 82: Upis podataka za prijavu [28]

Slika 83: Sučelje za unos podataka o danu [28]

Nakon što je preuzimanje podataka završilo korisnika se preusmjerava na sljedeće sučelje. U ovom slučaju preuzimanje ne kreću istovremeno, već se prvo preuzimaju podaci o svim klijentima te će aplikacija čekati da se preuzimanje završi i potom će krenuti preuzimati podatke o danima kao što je prikazano u isječku koda 37. Kada je aplikacija uspješno pokrenuta na mobilnom uređaju korisniku će se prikazati početno stanje sučelja za prijavu (slika 81) te će korisnik upisati svoje podatke (slika 82), ako su podaci točni korisniku će se prikazati početno stanje sučelja za unos podataka o radnom danu (slika 83).

5.7.1.2. Xamarin

Ispod je prikazan dijagram klasa prema kojem je implementirana funkcionalnost prijave.



Slika 84: Dijagram klasa implementacije funkcionalnosti prijave - Xamarin

Kao i u prethodnom primjeru prvo kreiramo opisne klase Klijent, Dan i Zaposlenik čiji su dijelovi prikazani u isječcima koda 38, 39 i 40.

```
public class Klijent {
        private String id;
        private String ime;
        private String prezime;
        private String Adresa;
        private String Lat;
        private String Lon; }
```

Isječak koda 38: Klasa Klijent [29]

```
public class Dan {
        private String Id;
        private String Datum;
        private List<Klijent> Popis = new List<Klijent>();
        private String Km;
        private int Mjesec;
        private int Godina;
        private String Pstanje;
        private String Zstanje;

       }
```

Isječak koda 39: Klasa Dan [29]

```
public class Zaposlenik {
        private int Id;
        private String Ime;
        private String Prezime;
        private String Auto;
        private String Registracija;
       }
```

Isječak koda 40: Klasa Klijent [29]

Nakon toga kreirat ćemo singleton klasu BazaPod u kojoj će nakon prijave biti pohranjeni podaci korisnika kao što je prikazano u isječku koda 41.

```
public sealed class BazaPod
    {
        private List<Klijent> sviKlijenti;
        private List<Dan> sviDani;
        private static BazaPod instanca = null;
        private Context context;
        private int stavljeno = 0;
        private int Ulogiranizaposlenik;
        public int Stavljeno { get => stavljeno;}
       private Zaposlenik UlogiraniZaposlenik;
        public List<Klijent> getSviKlijenti() { return sviKlijenti; }
        public List<Dan> getSviDani() { return sviDani; }
        public static BazaPod getInstanca()
        {
            if (instanca == null)
            {
                instanca = new BazaPod();
            }
            return instanca;
        }
        private BazaPod()
        {
            sviKlijenti = new List<Klijent>();
            sviDani = new List<Dan>();
        }
```

Isječak koda 41: Dio klase BazaPod [29]

Kako bi dohvatili podatke iz baze kreirat ćemo klase Podaci, Konekcija i Async. U apstraktnoj klasi Async definirat ćemo BackgroundWorker kao što je prikazano u isječku koda 42 koji će se koristiti u klasi Podaci kako bi se podaci iz baze mogli istovremeno preuzimati.

```
public abstract class Async
    {
        private BackgroundWorker bw;

        public  Async()
        {
            bw = new BackgroundWorker();
            bw.DoWork += (s, e) => { DoInBackground(); };
            bw.RunWorkerCompleted += (s, e) => {onPostExecute(); };
        }
        protected abstract void PreExecute(String link);
        protected abstract void DoInBackground();
        protected abstract string onPostExecute();

        public void Execute(String link)
        {
            PreExecute(link);
            bw.RunWorkerAsync();
        }
    }
}
```

Isječak koda 42: Abstrakna klasa Async [29]

Kao i u prethodnom primjeru kreirati klasa Podaci pomoću klase Konekcija dohvaća i čita podatke iz baze. U klasi Konekcija osim funkcije za dohvaćanje podataka definirane su i funkcije za razdvajanje i spremanje podataka u liste.

```
public String DohvatiPodatke(String link)
        {
            String odgovor = "test";
            try
            {
                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(link);
                HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                Stream dataStream = response.GetResponseStream();
                StreamReader reader = new StreamReader(dataStream);
                string responseFromServer = reader.ReadToEnd();
                odgovor = responseFromServer;
            }
            catch (Exception e)
            {
                odgovor= "error";
            }
            return odgovor;
        }
```

Isječak koda 43: Funkcija za dohvat podataka iz klase Konekcija [29]

U početnoj aktivnosti (MainActivity) kada korisnik klikne na gumb za prijavu prvo se provjeravaju uneseni podaci. Ako su podaci netočni ispisuje se obavijest korisniku, u suprotnome dohvaćaju se svi klijenti i svi prethodno uneseni dani korisnika kao što je prikazano u isječku koda 44.

```
private async void prijava(object sender, EventArgs e) {
            String u = username.Text;
            String p = password.Text;
            String link1 =
"http://crofi.com/assets/assets/images/RasporedBaza/Logiranje.php?username="
+ u + "&password=" + p;
            String link2 =
"http://crofi.com/assets/assets/images/RasporedBaza/Svilijenti.php";
            String result = null;
            try {
                Podaci logiranje= new Podaci(this, 1);
                int r = 0;
                int l = 0;
                TimeSpan vrijeme = TimeSpan.FromMilliseconds(10);
                Task t = Task.Run(() =>
                {
                    logiranje.Execute(link1);
                });
                while (r == 0){
                    if (logiranje.getRez().Count ==0) {
                        await Task.Delay(vrijeme);
                    }
                    else {
                        result = logiranje.getRez()[0];
                        if (result == "error" || result == "No" || result == null) {
                            l = 0;
                        }
                        else {
                            con.PrijaviZaposlenika(result);
                            l = 1;
                        }
                        r = 1;
                    }
                }
                if (l == 0) {
                    Toast.MakeText(Application.Context, "Pogrešni podaci",
                    ToastLength.Short).Show();
                }
                else {
                    Podaci dohvatSvihKlijenta = new Podaci(this, 2);
                    Podaci dohvatSvihDana = new Podaci(this, 3);
                     String link3 =
                     "http://crofi.com/assets/assets/images/RasporedBaza/SviDani.php?id
                     ="+con.UlogiraniZaposlenik1.Id1;
                    r = 0;
                    Task t2 = Task.Run(() => {
                        dohvatSvihKlijenta.Execute(link2);
                        dohvatSvihDana.Execute(link3);
                    });
                    while (r == 0){
                     if (dohvatSvihKlijenta.getRez().Count ==0 ||
                            dohvatSvihDana.getRez().Count==0) {
```

```
                    await Task.Delay(vrijeme);
                }
                else {
                    List<String> Daniresult = dohvatSvihDana.getRez();
                    con.StaviSveDane(Daniresult);
                    List<Dan> sviKlijenti = con.getSviDani();
                    r = 1;
                }
            }
            var intent = new Intent(this, typeof(KreiranjeDana));
            StartActivity(intent);
        }
    }
    catch (Exception r){
        Console.WriteLine("greska");
    }
}
```

Isječak koda 44: Funkcija za prijavu korisnika u klasi MainActivity [29]

Aplikacija čeka da završi dohvaćanje podataka o klijentima kako bi mogla sortirati podatke o danima u listu, zbog toga podaci se mogu istovremeno preuzimati, ali prije sortiranja svi moraju biti preuzeti. Nakon što je preuzimanje i sortiranje podataka završeno korisnika se preusmjerava na sljedeće sučelje (KreiranjeDana). Nakon uspješne prijave korisnik je preusmjeren na sučelje za kreiranje dana.

Slika 85: Početno stanje sučelja za prijavu [29]

Slika 86: Upis podataka za prijavu [29]

Slika 87: Sučelje za unos podataka o danu [29]

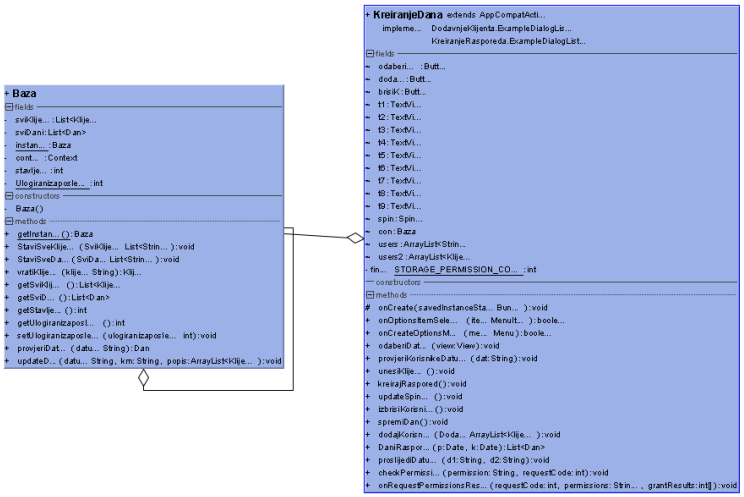
Kada je aplikacija pokrenuta na mobilnom uređaju korisnik će prvo vidjeti sučelje za prijavu (slika 85) i potom će upisati svoje korisničke podatke (slika 86), ako su podaci točni korisnik će biti preusmjeren na sučelje unos podataka o danu (slika 87).

5.7.2.Odabir Datuma

Kada korisnik klikne na gumb za promjenu datuma, otvorit će mu se kalendar na kojem će odabrati datum te će se odabrani datum potom ispisati u elementu za prikaz teksta. Aplikacija također mora provjeriti postoji li već definirani popis obiđenih klijenata.

5.7.2.1. Android Studio

U aktivnosti za kreiranje dana (KreiranjeDana) definiramo funkciju odaberiDatum() kao što je prikazano na dijagramu klasa (slika 88).



Slika 88: Dijagram klasa implementacije funkcionalnosti odabira datuma - Android Studio

Funkcija otvara kalendar u kojem korisnik može izabrati željeni datum kao što je prikazano u isječku koda 45.

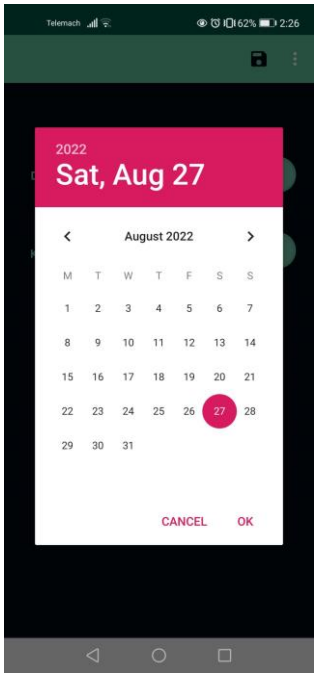
```
public void odaberiDatum(View view){
    Calendar calendar = Calendar.getInstance();
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
    DatePickerDialog datePickerDialog = new
    DatePickerDialog(KreiranjeDana.this,
    new DatePickerDialog.OnDateSetListener() {
     @Override
    public void onDateSet(DatePicker datePicker, int year, int month, int day){
                    month=month+1;
                    String datumDana=day + "/" + month + "/" + year;
                    t1.setText(datumDana);
                    provjeriKorisnikeDatuma(datumDana);
          } }, year, month, dayOfMonth);
    datePickerDialog.show(); }
```

Isječak koda 45: Funkcija za odabir datuma iz klaseKreiranjeDana [28]

Prije nego se ispiše odabrani datum pomoću funkcije provjeriKorisnikeDatuma() provjeravaju se podaci u bazi te ako postoji definiran popis obiđenih klijenata on se dohvaća i kreira se lista s prezimenima klijenata koja se pridružuje padajućem izborniku na sučelju kao što je prikazano u isječku koda 46.

```
public void provjeriKorisnikeDatuma(String dat) {
    Dan d=con.provjeriDatum(dat);
    users = new ArrayList<String>();
    users2= new ArrayList<Klijent>();
    for(int i=0;i<d.getPopis().size();i++){
        int x=i+1;
        String id=x+".";
        users.add(id+d.getPopis().get(i).getPrezime());
        users2.add(d.getPopis().get(i));
    }
    updateSpinner();
    String km=d.getKm();
    String ps=d.getPstanje();
    String zs=d.getZstanje();
    t2.setText(km);
    t8.setText(ps);
    t9.setText(zs);
}
```

Isječak koda 46: Funkcija za provjeru postojećih datuma iz klase KreiranjeDana [28]



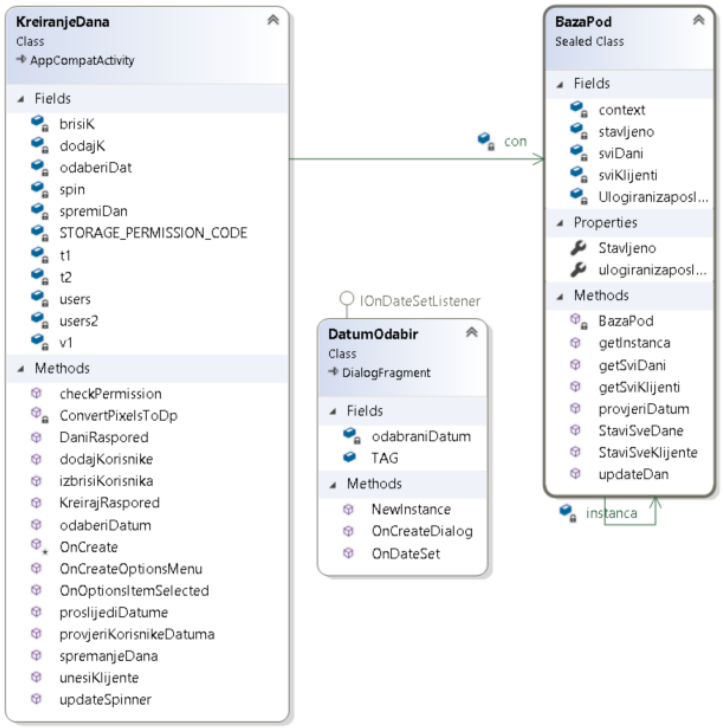
Slika 89: Odabir datuma pomoću kalendara [28]

Slika 90: Prikaz prethodno unesenih podataka za odabrani datum [28]

Također se dohvaćaju i ispisuju prijeđeni kilometri te stanje brojila za taj datum (ako su podaci uneseni) kao što je prikazano na slikama 89 i 90 na kojima korisnik odabire datum 27.8.2022. pomoću kalendara koji se otvara klikom na gumb „Promjeni“ te se potom na sučelju prikazuje prethodno definiran popis klijenata za taj datum i izračunati broj prijeđenih kilometara, u ovom slučaju za odabrani datum nisu definirana stanja brojila.

5.7.2.2. Xamarin

Kako bi se kalendar mogao uspješno prikazati prvo kreiramo klasu DatumOdabir u kojoj definiramo prikaz kalendara i vraćanje odabranog datuma kao što je prikazano na dijagramu klasa (slika 91).

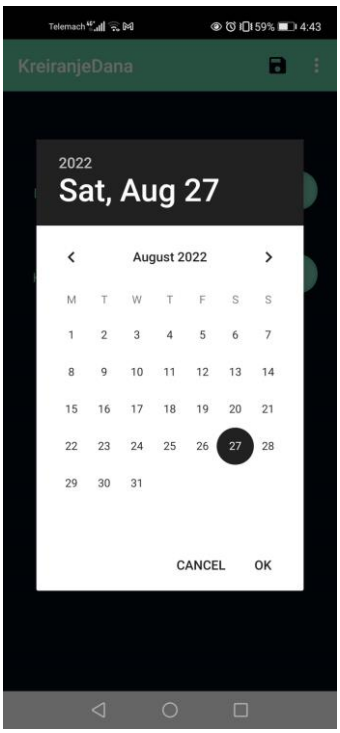


Slika 91: Dijagram klasa implementacije funkcionalnosti odabira datuma - Xamarin

Nakon što korisnik klikne na gumb za odabir datuma pozvat će se funkcija odaberiDatum() koja poziva klasu DatumOdabir te provjerava da li je taj datum već definiran kao što je prikazan u isječku koda 47.

```
public void odaberiDatum(object sender, EventArgs e) {
            spremanjeDana();
            var odabirDat = DatumOdabir.NewInstance(delegate (DateTime vrijeme) {
              string datumDana = vrijeme.Day + "/" + vrijeme.Month + "/" +
              vrijeme.Year;
                 t1.Text = datumDana;
                 provjeriKorisnikeDatuma(datumDana);
            });
            odabirDat.Show(FragmentManager, DatumOdabir.TAG);
        }
```

Isječak koda 47: Funkcija za odabir datuma iz klase KreiranjeDana [29]



Slika 92: Odabir datuma pomoću kalendara [29]

Slika 93: Prikaz prethodno unesenih podataka za odabrani datum [29]

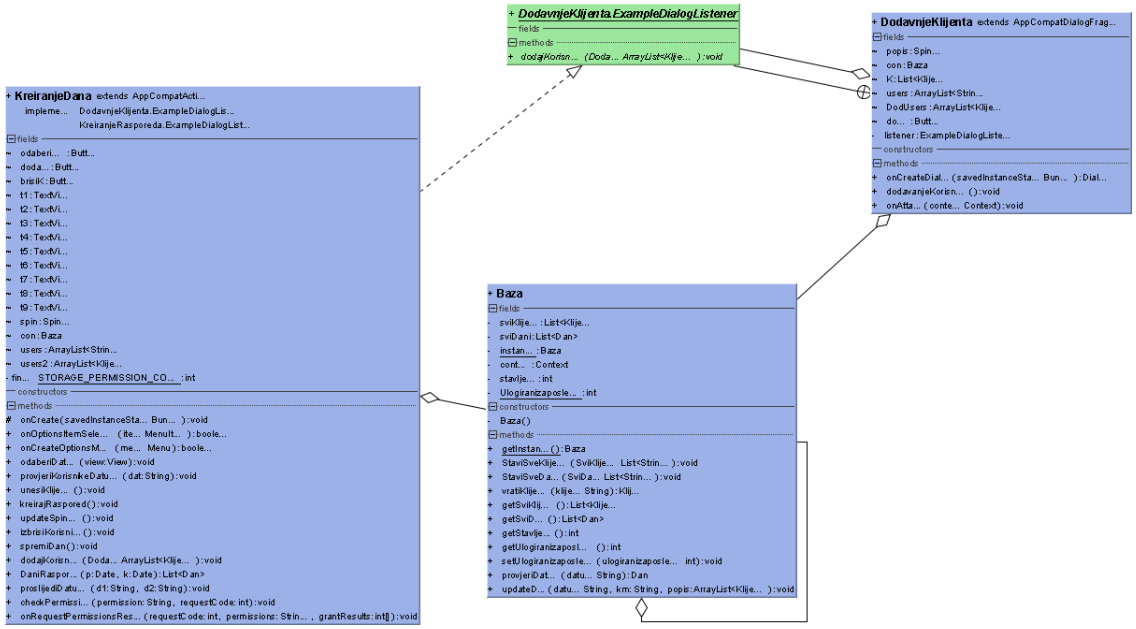
Kao što je prikazano na slici 92 korisnik odabire datum 27.8.2022. te se postojeći podaci ispisuju na sučelju (slika 93), također ispisuje se i izračunati broj prijeđenih kilometara. Korisnik može ponovno promijeniti datum klikom na gumb „Promjeni“ te će ako postoje novi podaci biti dohvaćeni i ispisani.

5.7.3. Dodavanje korisnika

Kada korisnik klikne na gumb za dodavanje klijenata aplikacija otvara prozor u kojem se nalazi padajući izbornik sa svim klijentima te korisnik može izabrati klijenta i dodati ga u popis za taj dan.

5.7.3.1. Android Studio

Kreirano sučelje (iskočnom prozoru za dodavanje korisnika) prikazivat ćemo pomoću klase DodavanjeKlijenta kao što je prikazano na dijagramu klasa (slika 94). Iz baze se dohvaćaju svi klijenti te kada korisnik odabere klijenta klikne na gumb za dodavanje i aktivira funkciju dodavanjeKorisnika() koja u zasebnu listu pohranjuje dodanog klijenta kao što je prikazano u isječku koda 48.



Slika 94: Dijagram klasa implementacije funkcionalnosti dodavanja korisnika - Android Studio

```
public void dodavanjeKorisnika(){
 Long prezime=popis.getSelectedItemId();
 int i=prezime.intValue();
 DodUsers.add(K.get(i));
}
```

Isječak koda 48: Funkcija za dodavanje korisnika iz klase DodavanjeKlijenta [28]

Slika 95: Odabir klijenta pomoću padajućeg izbornika [28]

Kada je korisnik dodao sve klijente mora kliknuti na gumb OK s kojim završava dodavanje te se zatvara iskočni prozori i prosljeđuje se lista dodanih klijenata (ako korisnik klikne na gumb Cancel lista se ne prosljeđuje). Iskočni prozor odnosno sučelje za dodavanje klijenata prikazuje se nakon što korisnik klikne na gumb za dodavanje koji aktivira funkciju unesiKlijente te korisnik može pretraživati sve dostupne klijente koji su abecedno sortirani u padajućem izborniku (slika 95).

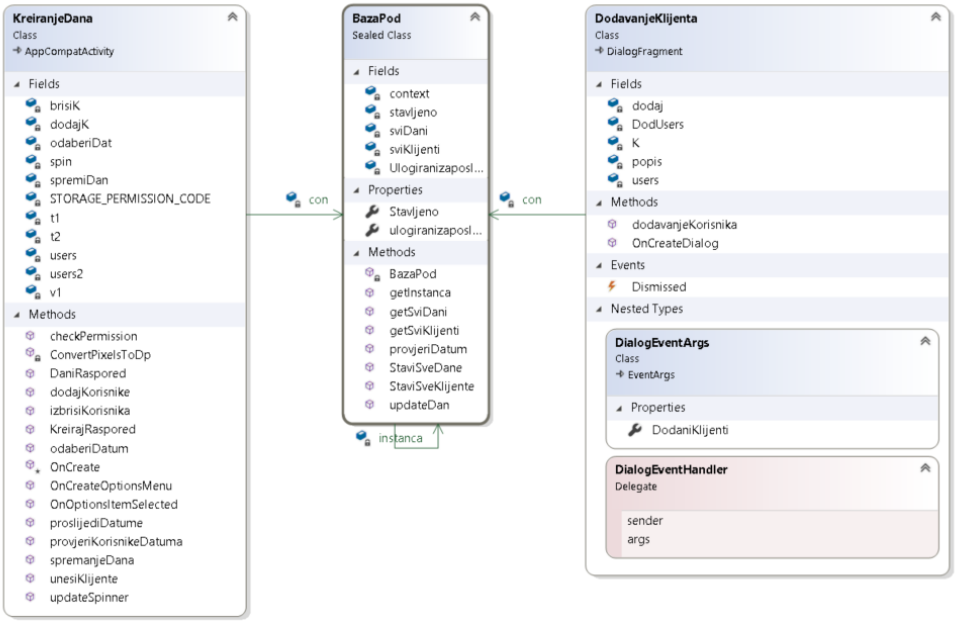
```
public void dodajKorisnike(ArrayList<Klijent> Dodani) {
    int pocetni=users.size();
     for(int i=0;i<Dodani.size();i++){
         pocetni=pocetni+1;
         String id=String.valueOf(pocetni);
         users.add(id+"."+Dodani.get(i).getPrezime());
         users2.add(Dodani.get(i));
     }
    updateSpinner();
    KorisniciLokacije kl= new KorisniciLokacije();
    String km=kl.TraziKorisnike(users2);
    t2.setText(km);
}
```

Isječak koda 49: Funkcija za dodavanje klijenata iz klase KreiranjeDana [28]

Nakon što korisnik klikne na gumb „SPREMI“ lista dodanih klijenata se prosljeđuje funkciji dodajKorisnike() koja osvježava padajući izbornik s popisom klijenata kao što je prikazano u isječku koda 49.

5.7.3.2. Xamarin

Prvo kreiramo klasu DodavanjeKlijenta kao što vidimo na dijagramu klasa (slika 96) koja će prikazivati prozor za dodavanje korisnika kao što je prikazano u isječku koda 50, u ovoj verziji nećemo dodane klijente prosljeđivati funkciji već će se nakon zatvaranja prozora dodani klijenti spremati u listu.

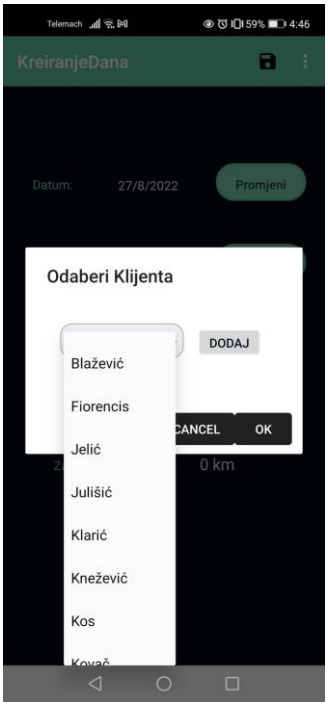


Slika 96: Dijagram klasa implementacije funkcionalnosti dodavanja korisnika - Xamarin

```
public override Dialog OnCreateDialog(Bundle savedInstanceState)
        {
            AlertDialog.Builder builder = new AlertDialog.Builder(Activity);
            LayoutInflater inflater = Activity.LayoutInflater;
            con = BazaPod.getInstanca();
            K = con.getSviKlijenti();
            for (int i = 0; i < K.Count; i++)
            {
                users.Add(K[i].Prezime);
            }
            View v = inflater.Inflate(Resource.Layout.dodavanje_klijenta, null);
            dodaj= v.FindViewById<Button>(Resource.Id.dod);
            popis = v.FindViewById<Spinner>(Resource.Id.sviKorisnici);
              ArrayAdapter<string> adapter = new ArrayAdapter<string>(Activity,
              Android.Resource.Layout.SimpleSpinnerDropDownItem, users);

              adapter.SetDropDownViewResource(Android.Resource.Layout.SimpleSpinnerDrop
              DownItem);
            popis.Adapter = adapter;
            dodaj.Click += dodavanjeKorisnika;
```

```
        builder.SetView(v)
               .SetTitle("Odaberi Klijenta")
               .SetPositiveButton("Ok", (sender, args) =>
                {
                    if (null != Dismissed)
                                    Dismissed(this, new DialogEventArgs {
          DodaniKlijenti = DodUsers });
                })
               .SetNegativeButton("Cancel", (sender, args) =>
               {
                   Console.WriteLine("Korisnik kliknuo Cancel");
               });
        return builder.Create();

    }
```

Isječak koda 50: Prikaz iskočnog prozora za dodavanje klijenata iz klase DodavanjeKlijenta [29]



Slika 97: Odabir klijenta pomoću padajućeg izbornika [29]

Prozor za dodavanje klijenata otvara se nakon što korisnik klikne na gumb za unos klijenta te korisnik može dodati novog klijenta pomoću padajućeg izbornika koji sadrži abecedno sortirani popis klijenata (slika 97). Nakon što su svi klijenti dodani i korisnik je potvrdio popis klikom na gumb „OK“, dohvaća se popis svih dodanih klijenata i prosljeđuje se funkciji dodajKorisnike() koja osvježava padajući izbornik s popisom klijenata.

5.7.4. Brisanje korisnika

Korisnik ima opciju da iz padajućeg izbornika može odabrati korisnika i obrisati ga iz popisa klijenata koji su obiđeni u radnom danu.

5.7.4.1. Android Studio

Kada korisnik odabere klijenta kojeg želi maknuti iz popisa, klikne na gumb Izbriši koji aktivira funkciju izbrisiKorisnika() koja je prikazan u isječku koda 51.

```
public void izbrisiKorisnika(){
    Long broj=spin.getSelectedItemId();
    int id=broj.intValue();
    if(users.size()!=1 || users.size()!=0) {
        for (int i = 0; i < users.size(); i++) {
            if (i == id) {
                users2.remove(i);
                users.remove(i); }}
        for (int i = 0; i < users.size(); i++) {
            int noviBroj = i + 1;
            String kor = users2.get(i).getPrezime();
            String novi = noviBroj + "." + kor;
            users.set(i, novi); }}
    else{
        users = new ArrayList<String>();
        users2 = new ArrayList<Klijent>();}
    updateSpinner();
}
```

Isječak koda 51: Funkcija za brisanje korisnika iz klase KreiranjeDana [28]

Slika 98: Odabir korisnika za brisanje [28]

Cilj funkcije je izbrisati klijenta kojeg je korisnik odabrao iz padajućeg izbornika kao što je prikazano na slici 98 i ostale klijente pomaknuti na odgovarajuće mjesto u popisu, nakon što je klijent izbrisan i korisnici pomaknuti osvježava se padajući izbornik.

5.7.4.2. Xamarin

Kao i u prethodnoj verziji nakon što korisnik odabere klijenta klikne na gumb za brisanje koji poziva funkciju izbrisiKorisnika() koja je prikazana u isječku koda 52.

```
public async void izbrisiKorisnika(object sender, EventArgs e) {
            long broj = spin.SelectedItemId;
            int id = Convert.ToInt32(broj);
            if (users.Count != 1 || users.Count != 0) {
                for (int i = 0; i < users.Count; i++){
                    if (i == id){
                        users2.RemoveAt(i);
                        users.RemoveAt(i);
                    }
                }
                for (int i = 0; i < users.Count; i++){
                    int noviBroj = i + 1;
                    String kor = users2[i].Prezime;
                    String novi = noviBroj + "." + kor;
                    users[i] = novi;
                }
            }
            else{
                users = new List<String>();
                users2 = new List<Klijent>();
            }
            updateSpinner();
}
```

Isječak koda 52: Funkcija za brisanje korisnika iz klase KreiranjeDana [29]

Slika 99: Odabir korisnika za brisanje [29]

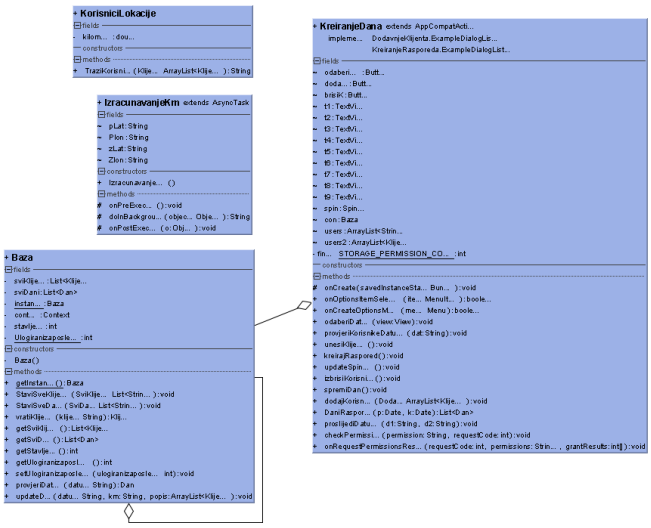
Kao što je prikazano na slici 99 korisnik iz padajućeg izbornika odabire jednog klijenta te potom klikće na gumb „Izbriši“, potom se preostali klijenti u popisu pomiču za jedno mjesto i padajući izbornik se osvježava.

5.7.5. Izračun i ispis prijeđenih kilometara

Prijeđeni broj kilometara će se izračunavati pomoću google api servisa distance matrix kojem će se prosljeđivati koordinate adresa klijenata i on će na temelju podataka vraćati udaljenost u kilometrima.

5.7.5.1. Android Studio

Kreiramo novu klasu pod nazivom IzracunavanjeKm koja će komunicirati sa servisom. Klasa prvo prima koordinate polazišta i odredišta te ih prosljeđuje servisu.



Slika 100: Dijagram klasa implementacije funkcionalnosti izračuna i ispisa kilometara - Android Studio

Nakon što servis primi podatke izračunava se udaljenost te izračunava udaljenost između 2 mjesta.

```
protected String doInBackground(Object[] objects) {
    this.pLat=objects[0].toString();
    this.Plon=objects[1].toString();
    this.zLat=objects[2].toString();
    this.Zlon=objects[3].toString();
    String link=
"https://maps.googleapis.com/maps/api/distancematrix/json?destinations="+zLat+"
%2C"+Zlon+"&origins="+pLat+"%2C"+Plon+"&key=AIzaSyCiLTKcVOoWFXqSe-
TpI0jyUty7dG4zFJw";
    String rez="";
    try {
        URL url=new URL(link);
        HttpURLConnection con= (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        con.connect();
        int statuscode=con.getResponseCode();
```

```
        if(statuscode==HttpURLConnection.HTTP_OK) {
            BufferedReader br=new BufferedReader(new
            InputStreamReader(con.getInputStream()));
            StringBuilder sb=new StringBuilder();
            String line=br.readLine();
            while(line!=null) {
                sb.append(line);
                line=br.readLine();
            }
            String json=sb.toString();
            JSONObject root=new JSONObject(json);
            JSONArray array_rows=root.getJSONArray("rows");
            JSONObject object_rows=array_rows.getJSONObject(0);
            JSONArray array_elements=object_rows.getJSONArray("elements");
            JSONObject  object_elements=array_elements.getJSONObject(0);
            JSONObject
            object_distance=object_elements.getJSONObject("distance");
            rez= object_distance.getString("text"); }
        } catch (MalformedURLException e) {
            rez="er1";
        } catch (IOException e) {
            rez="er1";
        } catch (JSONException e) {
            rez="er1";
        } return rez;
}
```

Isječak koda 53: Izračunavanje udaljenosti u kilometrima u klasi IzracunavanjeKm [28]

Udaljenost se ispisuje u obliku Json objekta koji se dohvaća i čita pomoću BufferedReader-a. Nakon što je pročitana udaljenost između mjesta, klasa vraća rezultat kao što je prikazano u isječku koda 53. Ako se dogodi greška kod komunikacije sa servisom vrati će se string s tekstom koji označava grešku. Nakon toga kreiramo klasu KorisniciLokacije koja će prolaziti kroz popis klijenata, dohvaćati njihove koordinate i pomoću klase IzracunavanjeKm izračunavati udaljenosti između klijenata koji su na popisu kao što je prikazano u isječku koda 54.

```
for(int i=0;i<Klijenti.size()-1;i++){
     String pLat=Klijenti.get(i).getLat();
     String Plon=Klijenti.get(i).getLon();
     String zLat=Klijenti.get(i+1).getLat();
     String Zlon=Klijenti.get(i+1).getLon();
        try {
        duljina=new IzracunavanjeKm().execute(pLat, Plon, zLat, Zlon).get();
        String km=duljina.toString().replace("km","");
        double d=Double.parseDouble(km);
        kilometri=kilometri+d;
        } catch (ExecutionException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }}
```

Isječak koda 54: Prolazak kroz listu klijenata u klasi KorisniciLokacije [28]

Kada je funkcija prošla kroz cijeli popis klijenata vrati će ukupnu udaljenost u kilometrima. Ovu klasu ćemo ukomponirati u funkcije za dodavanje (dodajKorisnike) i brisanje klijenata (izbrisiKorisnika) kao što je prikazano u isječku koda 55.

```
KorisniciLokacije kl= new KorisniciLokacije();
String km=kl.TraziKorisnike(users2);
t2.setText(km);
```

Isječak koda 55: Primjena klase KorisniciLokacije [28]

Slika 101: Prijeđeni broja kilometara prije dodavanja novih klijenata [28]

Slika 102: Prijeđenih broja kilometara nakon dodavanja novih klijenata [28]

Slika 103: Popis klijenata nakon dodavanja [28]

Slika 104: Prijeđeni broja kilometara prije brisanja klijenta [28]

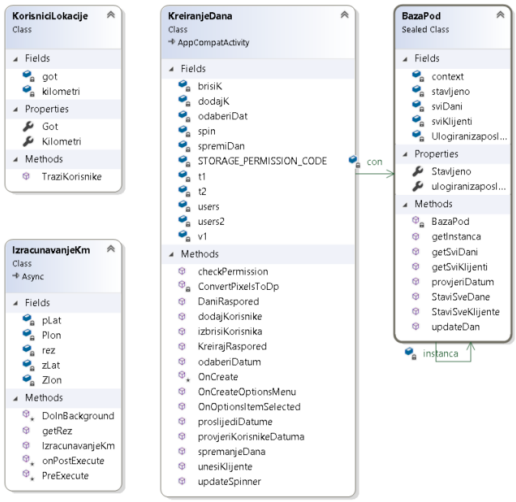
Slika 105: Prijeđenih broja kilometara nakon brisanja klijenta [28]

Slika 106: Popis klijenata nakon brisanja [28]

Svaki puta kad korisnik dodaje (postupak prikazan na slikama 101, 102 i 103) ili briše (postupak prikazan na slikama 104, 105 i 106) klijente iz popisa, klasi KorisniciLokacije se prosljeđuje nova lista klijenata i automatski se osvježava ukupno prijeđena udaljenost te se ispisuje na sučelje u odgovarajući element za prikaz teksta.

5.7.5.2. Xamarin

Kreiramo klasu IzracunavanjeKm kao što vidimo na dijagramu klasa (slika 107) koja će primati koordinate odredišta i polazišta i pomoću google api servisa izračunavati i vraćati udaljenost u kilometrima.



Slika 107: Dijagram klasa implementacije funkcionalnosti izračuna i ispisa kilometara - Xamarin

```
protected override void DoInBackground(){

     string link =
@"https://maps.googleapis.com/maps/api/distancematrix/json?destinations=" + zLat +
"%2C" + Zlon + "&origins=" + pLat + "%2C" + Plon + "&key=AIzaSyCiLTKcVOoWFXqSe-
TpI0jyUty7dG4zFJw";
      try{
           WebRequest request = WebRequest.Create(link);
           WebResponse response = request.GetResponse();
           Stream data = response.GetResponseStream();
           StreamReader reader = new StreamReader(data);
           string responseFromServer = reader.ReadToEnd();
           response.Close();
           JObject json = JObject.Parse(responseFromServer);
           string rezultat;
           rezultat =json.SelectToken("rows[0].elements[0].distance.text").ToString();
           rez = rezultat;
            }
            catch (Exception ex)
            {
                rez= "error";
            }

       }
```

Isječak koda 56: Izračun udaljenosti u klasi IzracunavanjeKm [29]

Ako se dogodi greška vraća se string „error“ kao što je prikazano u isječku koda 56. Ovu klasu ćemo implementirati u klasu KorisniciLokacije koja će primati listu klijenata i dohvaćati njihove koordinate kao što je prikazano u isječku koda 57. Kada dohvati koordinate proslijediti će ih klasi IzracunavanjeKm koja će vratiti udaljenost između dvije lokacije.

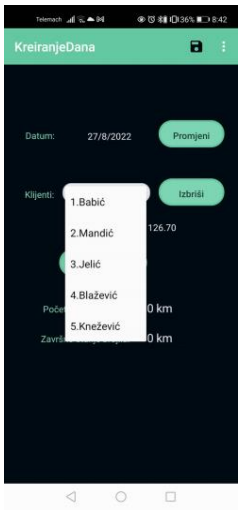
```
public async void TraziKorisnike(List<Klijent> Klijenti) {
            double km=0;
            for (int i = 0; i < Klijenti.Count - 1; i++){
                String pLat = Klijenti[i].Lat1;
                String Plon = Klijenti[i].Lon1;
                String zLat = Klijenti[i+1].Lat1;
                String Zlon = Klijenti[i+1].Lon1;
                try
                {
                IzracunavanjeKm duljina = new IzracunavanjeKm(pLat, Plon, zLat, Zlon);

                    int r = 0;
                    int l = 0;
                    TimeSpan vrijeme = TimeSpan.FromMilliseconds(1);
                    Task t = Task.Run(() =>
                    {
                        duljina.Execute("");
                    });
                    while (r == 0)
                    {
                        if (duljina.getRez()=="No"){
                            await Task.Delay(vrijeme);
                        }
                        else
                        {
                           String result= duljina.getRez();
                           if (result == "error" || result == "No" || result == null){
                                l = 0;
                            }
                           else{
                                string brojkm=result.Replace("km", "");
                                km = km+Convert.ToDouble(brojkm);
                                l = 1;
                            }
                            r = 1;
                        }
                    }
                }
                catch (Exception e)
                {
                }
            }
            kilometri = String.Format("{0:0.00}", km);
            got = 1;

        }
```

Isječak koda 57: Funkcija za pretragu popisa klijenata iz klase KorisniciLokacije [29]

Slika 108: Prikaz prijeđenih broja kilometara prije dodavanja novih klijenata [29]

Slika 109: Prikaz prijeđenih broja kilometara nakon dodavanja novih klijenata [29]

Slika 110: Prikaz popisa klijenata nakon dodavanja [29]



Slika 111: Prikaz prijeđenih broja kilometara prije brisanja klijenta [29]

Slika 112: Prikaz prijeđenih broja kilometara nakon brisanja klijenta [29]

Slika 113: Prikaz popisa klijenata nakon brisanja [29]

Kao i u prošloj verziji klasa KorisniciLokacije koristit će se u funkcijama izbrisiKorisnika() i u funkciji dodajKorisnike() što znači da će se prijeđeni kilometri preračunavati svaki puta kada korisnik dodaje (postupak prikazan na slikama 108, 109 i 110) ili briše (postupak prikazan na slikama 111, 112 i 113) klijente iz popisa.

5.7.6. Spremanje podataka u bazu

Podaci će se spremati ako korisnik klikne na gumb za spremanje i ako korisnik promjeni datum (automatski će se spremati na promjenu datuma).

5.7.6.1. Android Studio

U klasi KreiranjeDana kreiramo funkciju spremiDan() kao što je prikazano u isječku koda 58 koja će promijeniti podatke dana u lokalnoj listi i u bazi podataka koja se nalazi na serveru.

```
public void spremiDan(){
    String datum=t1.getText().toString();
    String km=t2.getText().toString();
    con.updateDan(datum, km, users2);
    String lista="";
    for(int i=0;i<users2.size();i++){
        lista=lista+users2.get(i).getId();
        if(i!=users2.size()-1){
            lista=lista+",";
        }
    }
    String Datum = datum;
    String Poc = t8.getText().toString();
    String Zav = t9.getText.toString();
    String Pkm = km;
    String Popis = lista;
    Integer Mjesec = 9;
    Integer Godina = 2022;
    Integer Zaposlenik = con.getUlogiranizaposlenik();
    String link =
"http://crofi.com/assets/assets/images/RasporedBaza/UpdateDan.php?Datum="+Datum
+"&Poc="+Poc+"&Zav="+Zav+"&Pkm="+Pkm+"&Popis="+Popis+"&Mjesec="+Mjesec+"&Godina
="+Godina+"&Zaposlenik="+Zaposlenik;
    Object result= null;
    try {
        result = new Podaci(this,null, 4).execute(link).get();
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Isječak koda 58: Funkcija za spremanje dana iz klase KreiranjeDana [28]

Slika 114: Obavijest o spremanju promjena [28]

Funkcija se poziva kada korisnik klikne na gumb za spremanje i kod odabira datuma te će se korisniku pojaviti obavijest kao što je prikazano na slici 114.

5.7.6.2. Xamarin

Kreirat ćemo funkciju spremanjeDatuma() koja će spremati sve promjene u lokalnim listama i u bazi podataka te je prikazana u isječku koda 59.

```
public async void spremanjeDana() {

            string datum = t1.Text;
            string km = t2.Text;
            con.updateDan(datum, km, users2);
            String lista = "";
            for (int i = 0; i < users2.Count; i++)
            {
                lista = lista + users2[i].Id;
                if (i != users2.Count - 1)
                {
                    lista = lista + ",";
                }
            }
            string Datum = datum;
            string Poc = t3.Text;
            string Zav = t4.Text;
            string Pkm = km;
            string Popis = lista;
            int Mjesec = 10;
            int Godina = 2021;
            int Zaposlenik = con.ulogiranizaposlenik;
            string link =
"http://crofi.com/assets/assets/images/RasporedBaza/UpdateDan.php?Datum=" + Datum +
"&Poc=" + Poc + "&Zav=" + Zav + "&Pkm=" + Pkm + "&Popis=" + Popis + "&Mjesec=" +
Mjesec + "&Godina=" + Godina + "&Zaposlenik=" + Zaposlenik;
```

```
            Podaci update = new Podaci(this, 1);
            int r = 0;
            int l = 0;
            TimeSpan vrijeme = TimeSpan.FromMilliseconds(2);
            Task t = Task.Run(() =>
            {
                update.Execute(link);
            });
            while (r == 0)
            {
                if (update.getRez().Count == 0)
                {
                    await Task.Delay(vrijeme);
                }
                else
                {
                    Console.WriteLine("Spremljeno");
                    r = 1;
                }
            }
}
```

Isječak koda 59: Funkcija za spremanje dana iz klase KreiranjeDana [29]

Slika 115: Obavijest o spremanju promjena [29]

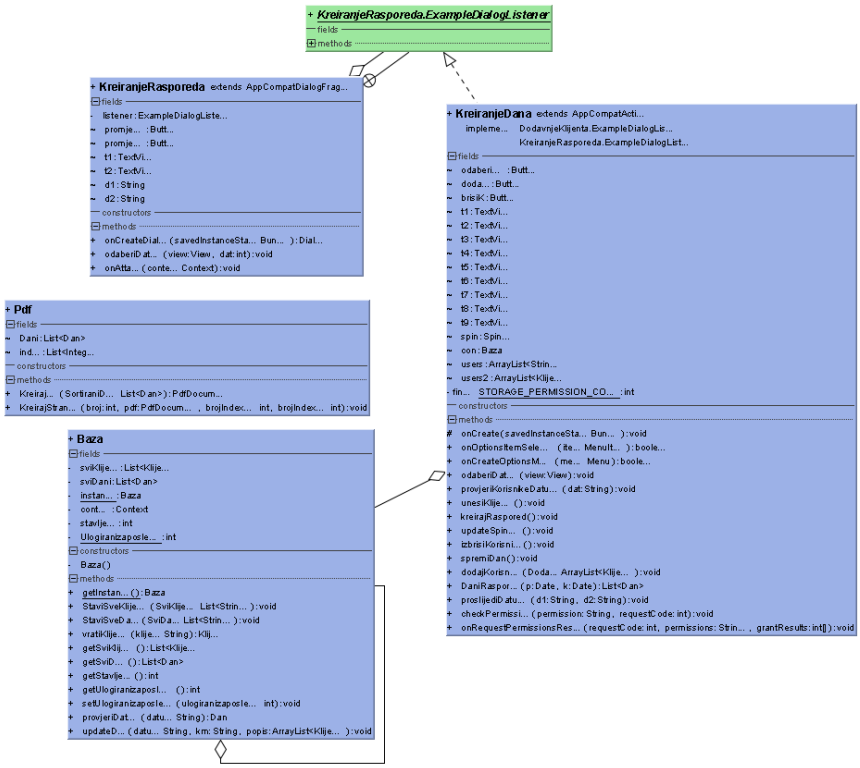
Funkcija se poziva kod promjene datuma i kod klika gumba za spremanje i korisniku će se ispisati obavijest o uspješnom spremanju podataka (slika 115).

5.7.7. Generiranje izvještaja

Kada korisnik klikne na opciju Kreiraj Raspored koja se nalazi u izborniku otvorit će se posebni prozor u kojem korisnik odabire od kojeg do kojeg datuma će se generirati izvještaj i zatim se generira pdf izvještaj o zadanim danima. Izvještaj će se generirati u pdf obliku pomoću zasebne klase u kojoj će biti definiran dizajn izvještaja.

5.7.7.1. Android Studio

Kreiramo novu klasu pod nazivom Pdf kao što je prikazano na dijagramu klasa (slika 116). Prvo moramo odrediti koliko dana će izvještaj imati te koliko dana stane na jednu stranicu.



Slika 116: Dijagram klasa implementacije funkcionalnosti generiranja izvještaja – Android Studio

```
public PdfDocument Kreirajpdf(List<Dan> SortiraniDan){
    System.out.println("broj dana: "+SortiraniDan.size());
    Dani=SortiraniDan;
    PdfDocument pdf= new PdfDocument();
    int b=0;
    int bDani=0;
    for(int i=0;i<Dani.size();i++){
```

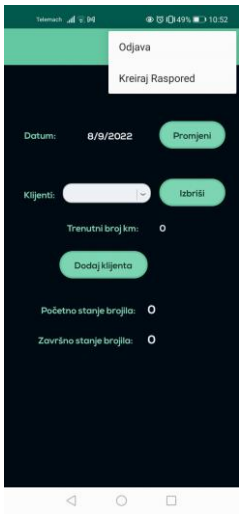
```
            int brojLjudi=b+Dani.get(i).getPopis().size();
            System.out.println("broj ljudi u danu : "+Dani.get(i).getDatum()+"
            "+brojLjudi);
            if(brojLjudi<55) {
                b = b + Dani.get(i).getPopis().size();
                if(Dani.get(i).getPopis().size()==0){
                    b=b+3; }
                bDani=bDani+1;
            }
            else{
                b=0;
                bDani=0;
                System.out.println("dodani index: "+i);
                index.add(i); }
        }
        if(index.size()<1){
            index.add(Dani.size());
        }
        int stranice=index.size()+1;
        System.out.println("broj strainica: "+stranice);
        for(int i=0;i<index.size();i++) {
            if(i!=0) {
                KreirajStranicu(i + 1, pdf, index.get(i-1), index.get(i)); }
            else{
                KreirajStranicu(i + 1, pdf, 0, index.get(i)); }}
        KreirajStranicu(stranice, pdf, index.get(index.size()-1), Dani.size());
        return pdf;
    }
```

Isječak koda 60: Određivanje broja dana i ljudi na stranici Pdf dokumenta u klasi Pdf [28]

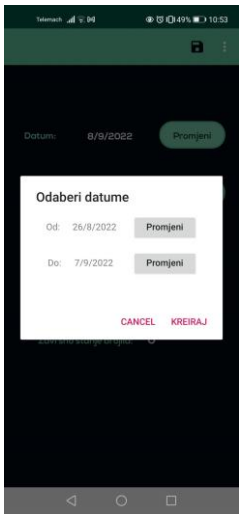
Potom kreiramo svaku stranicu u izvještaju i vraćamo kreirani izvještaj kao što je prikazano u isječku koda 60. Zatim kreiramo klasu KreiranjeRasporeda koja će prikazivati i upravljati sučeljem za odabir datuma pomoću funkcije odaberiDatum() koja je prikazana u isječku koda 61.

```
public void odaberiDatum(View view, int dat){
    Calendar calendar = Calendar.getInstance();
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
    final int x=dat;
    DatePickerDialog datePickerDialog = new DatePickerDialog(this.getContext(),
            new DatePickerDialog.OnDateSetListener() {
                @Override
                public void onDateSet(DatePicker datePicker, int year, int
                month, int day) {
                    month=month+1;
                    String datumDana=day + "/" + month + "/" + year;
                    if(x==1){
                        t1.setText(datumDana); }
                    else{
                        t2.setText(datumDana); }
                }
                }, year, month, dayOfMonth);
    datePickerDialog.show();}
```

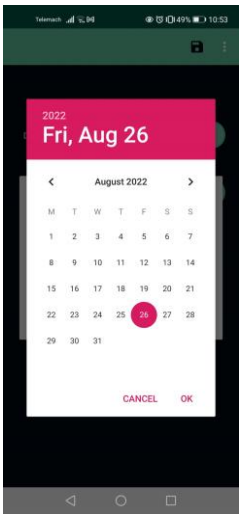
Isječak koda 61: Funkcija za odabir datuma iz klase KreiranjeRasporeda [28]



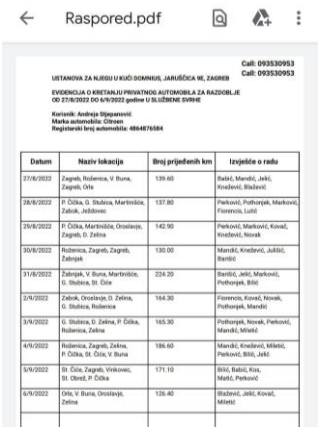
Slika 117: Izbornik [28]



Slika 118: Sučelje za odabir raspona datuma u rasporedu [28]



Slika 119: Primjer odabira datuma [28]

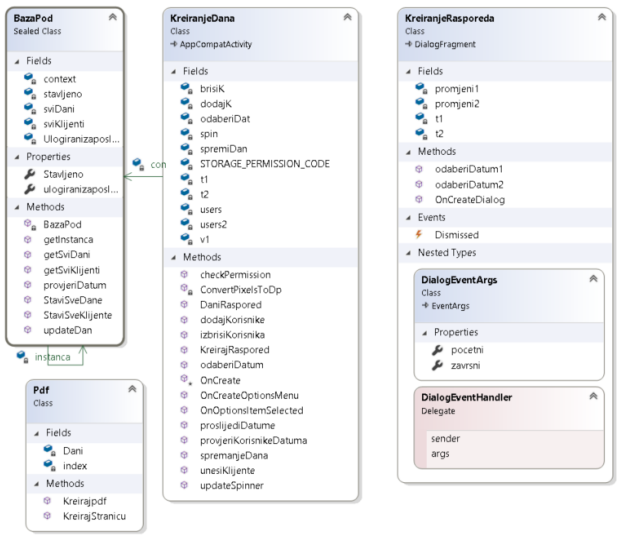


Slika 120: Kreirana pdf datoteka na temelju odabranih datuma [28]

Korisnik prvo klikće na ikonu izbornika u gornjem desnom kutu te odabire opciju „Kreiraj Raspored“ kao što je prikazano na slici 117. Kada korisnik klikne na gumb za promjenu određenog datuma otvara mu se kalendar u kojem odabire željeni datum (slika 119) te se zapisuje u element za prikaz teksta kao što se vidi na slici 118, korisnik mora odrediti početni i završni datum rasporeda. Kada je korisnik završio s odabirom datuma mora kliknuti na gumb Kreiraj u otvorenom prozoru (ako klikne na Cancel datumi se poništavaju). Potom se odabrani datumi prosljeđuju funkciji proslijediDatume() koja se nalazi u klasi KreiranjeDana Nakon što se odabrani datumi proslijede funkciji ona ih prvo sortira pomoću funkcije DaniRaspored() koja vraća listu sortiranih dana na temelju početnog i završnog datuma te se u memoriji uređaja kreira pdf datoteka koja sadrži podatke o radnim danima korisnika kao što je na slici 120.

5.7.7.2. Xamarin

Kreiramo klasu Pdf u kojoj se nalaze funkcije Kreirajpdf() i KreirajStranicu().Funkciji Kreirajpdf() prosljeđuje se lista sortiranih dana od čijih podataka će se kreirati izvještaj.



Slika 121: Dijagram klasa implementacije funkcionalnosti generiranja izvještaja - Xamarin

S obzirom na broj dana kreiraju se stranice izvještaja te svaka stranica ima određen broj podataka, svaka stranica kreira se pomoću funkcije KreirajStranicu() koja je prikazana u isječku koda 62. Na svakoj stranici prvo se moraju ispisati osnovne informacije o izvještaju te se potom podaci o danima ispisuju u tablici.

```
public void KreirajStranicu(int broj, PdfDocument pdf, int brojIndexaP, int
                            brojIndexaZ) {
            PdfDocument.PageInfo pi = new PdfDocument.PageInfo.Builder(1200, 2010,
            broj).Create();
            PdfDocument.Page myPage = pdf.StartPage(pi);
            con = BazaPod.getInstanca();
            Zaposlenik ulogiraniZap = con.UlogiraniZaposlenik1;
            int pageWidth = 1200;
            int pageHeight = 2010;
            //naslov
            Paint paint = new Paint();
            Paint title = new Paint();
            Canvas c = myPage.Canvas;
            title.TextAlign = Paint.Align.Left;
            title.SetTypeface(Typeface.DefaultBold);
            title.TextSize = 25;
            c.DrawText("USTANOVA ZA NJEGU U KUĆI DOMNIUS, JARUŠČICA 9E, ZAGREB", 150,
            100, title);
            c.DrawText("EVIDENCIJA O KRETANJU PRIVATNOG AUTOMOBILA ZA RAZDOBLJE", 150,
            160, title);
            c.DrawText("OD " + Dani[0].Datum1 + " DO " + Dani[Dani.Count - 1].Datum1 +
            " godine U SLUŽBENE SVRHE", 150, 190, title);
            c.DrawText("Korisnik: " +ulogiraniZap.Ime1+" "+ulogiraniZap.Prezime1, 150,
```

```
250, title);
c.DrawText("Marka automobila: "+ulogiraniZap.Auto1, 150, 280, title);
c.DrawText("Registarski broj automobila: "+ulogiraniZap.Registracija1,
150, 310, title);
//informacije u desnom kutu
paint.SetTypeface(Typeface.DefaultBold);
paint.TextSize = 30f;
paint.TextAlign = Paint.Align.Right;
c.DrawText("Call: 093530953", 1160, 40, paint);
c.DrawText("Call: 093530953", 1160, 80, paint);
//tablica
title.SetStyle(Paint.Style.Stroke);
title.StrokeWidth = 3;
c.DrawRect(20, 400, pageWidth - 20, 480, title);
title.TextAlign = Paint.Align.Left;
title.SetStyle(Paint.Style.Fill);
paint.TextAlign = Paint.Align.Left;
paint.SetStyle(Paint.Style.Fill);
c.DrawText("Datum", 60, 450, paint);
c.DrawText("Naziv lokacija", 260, 450, paint);
c.DrawText("Broj prijeđenih km", 570, 450, paint);
c.DrawText("Izvješće o radu", 890, 450, paint);
c.DrawLine(20, 400, 20, pageHeight - 200, title);
c.DrawLine(180, 400, 180, pageHeight - 200, title);
c.DrawLine(550, 400, 550, pageHeight - 200, title);
c.DrawLine(830, 400, 830, pageHeight - 200, title);
c.DrawLine(pageWidth - 20, 400, pageWidth - 20, pageHeight - 200, title);
//podaci
paint.TextSize = 25f;
paint.SetTypeface(Typeface.Default);
int y = 510;
int pocetak = brojIndexaP;
int zavrsetak = brojIndexaZ;
for (int i = pocetak; i < zavrsetak; i++){
    y = y + 10;
    String datumDana = Dani[i].Datum1;
    String kilometri = Dani[i].Km1;
    c.DrawText(datumDana, 30, y, paint);
    c.DrawText(kilometri, 580, y, paint);
    Console.WriteLine("veliciina popisa: " + Dani[i].Popis1.Count);
    if (Dani[i].Popis1.Count > 3) {
        int red = 0;
        String lokacije = "";
        String text = "";
        for (int j = 0; j < Dani[i].Popis1.Count; j++){
            if (red < 2 && j < Dani[i].Popis1.Count - 1) {
                lokacije = lokacije + Dani[i].Popis1[j].Adresa1 + ", ";
                text = text + Dani[i].Popis1[j].Prezime + ", ";
                red++;
            }
            else{
                red = 0;
                lokacije = lokacije + Dani[i].Popis1[j].Adresa1;
                text = text + Dani[i].Popis1[j].Prezime;
                if (j != Dani[i].Popis1.Count - 1)
                {
                    lokacije = lokacije + ", ";
                    text = text + ", ";
                }
                c.DrawText(lokacije, 190, y, paint);
```

```
                        c.DrawText(text, 840, y, paint);
                        text = "";
                        lokacije = "";
                        y = y + 35;
                    }
                }
            }
            else{
                String lokacije = "";
                String text = "";
                for (int j = 0; j < Dani[i].Popis1.Count; j++){
                    lokacije = lokacije + Dani[i].Popis1[j].Adresa1;
                    text = text + Dani[i].Popis1[j].Prezime;
                    if (j != Dani[i].Popis1.Count - 1) {
                        lokacije = lokacije + ", ";
                        text = text + ", ";
                    }
                }
                c.DrawText(lokacije, 190, y, paint);
                c.DrawText(text, 840, y, paint);
                y = y + 35;
            }
            c.DrawLine(20, y, pageWidth - 20, y, title);
            y = y + 20;
        }
        pdf.FinishPage(myPage);
    }
}
```

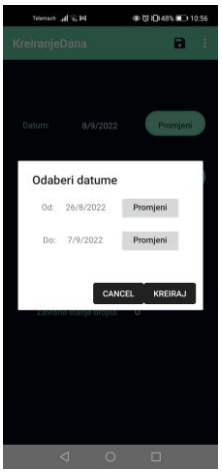
Isječak koda 62: Funkcija za kreiranje stranice u Pdf dokumentu iz klase Pdf [29]

Kada su sve stranice završene vraća se pdf dokument. Zatim kreiramo novu klasu KreiranjeRasporeda s kojom ćemo upravljati sučeljem za odabir datuma koji će se koristiti u kreiranju izvještaja. Klasa će imati dvije varijable pocetni i zavrsni u koje će se spremati odabrani datumi. Svaki puta kad korisnik klikne na gumb za promjenu datuma, otvorit će mu se kalendar te kada odabere datum on će se spremiti u jednu od varijabli(ovisno koji datum korisnik odabire) pomoću funkcije koje je prikazana u isječku koda 63. Zatim u klasi kreiranjeDana kreiramo funkciju KreirajRaspored() u kojoj će se kreirati objekt klase KreiranjeRasporeda.

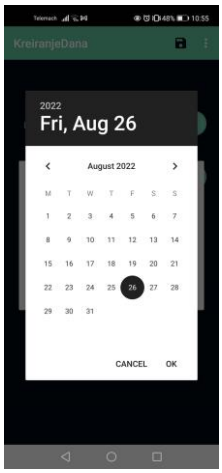
```
public void odaberiDatum1(object sender, EventArgs e) {
       var odabirDat = DatumOdabir.NewInstance(delegate (DateTime vrijeme) {
          string datumDana = vrijeme.Day + "/" + vrijeme.Month + "/" + vrijeme.Year;
                    t1.Text = datumDana;
          });
          odabirDat.Show(FragmentManager, DatumOdabir.TAG);

       }
```

Isječak koda 63: Funkcija za dabir jednog od datuma iz klase KreiranjeRasporeda [29]

Slika 122: Izbornik [29]



Slika 123: Sučelje za odabir raspona datuma u rasporedu [29]



Slika 124: Primjer odabira datuma [29]

Slika 125: Kreirana pdf datoteka na temelju odabranih datuma [29]

Postupak kreiranja rasporeda će se pokrenuti kada korisnik klikne na gumb Kreiraj raspored u izborniku kao što je prikazano na slici 121. Potom se otvara sučelje za odabir datuma (slika 123) i korisnik klikom na gumb za promjenu datuma odabire početni i završni datum vremenskog okvira rasporeda (slika 124). Nakon što je korisnik potvrdio odabir datuma, sučelje se zatvara i datumi se prosljeđuju se funkciji proslijediDatume(). Funkcija proslijediDatume() prvo prosljeđuje početni i završni datum funkciji DaniRaspored() koja na temelju odabranih datuma dohvaća i sortira sve dane koji spadaju u zadani vremenski okvir. I zatim vraća listu sortiranih dana. Potom se vraćena lista prosljeđuje u funkciju Kreirajpdf() te kada funkcija vrati kreirani pdf izvještaj sprema se pod nazivom Raspored2.pdf kao što je prikazano na slici 125.

6. Usporedba

Usporedbu ću napraviti pomoću nekoliko kriterija te će usporedba biti iz osobnog iskustva paralelnog kreiranja android aplikacije u oba razvojna okruženja s oba programska jezika. Kriteriji usporedbe bit će sljedeći:

- Programski jezik – ovaj kriterij će biti presudan nekome tko nije upoznat s oba programska jezika već samo s jednim te će odabrati alat koji ga koristi. U ovom slučaju Java odnosno Android Studio bi mogao imati prednost, jer po ovogodišnjoj statistici [24] ipak više ljudi koristi Javu
- Dokumentacija – mislim da je ovo jedan od najbitnijih kriterija, jer dobro napisana dokumentacija razvojnog okruženja može uvelike olakšati i skratiti vrijeme razvoja samog proizvoda. U ovom slučaju koristit ću službenu dokumentaciju od Android Studiju [1] i Xamarin- a [6] te ću kroz razvoj same aplikacije vidjeti koja je kvalitetnije napisana
- Emulatori – kada korisnik želi testirati aplikaciju bez vanjskog uređaja, koristit će emulatore, oba razvojna okruženja omogućuju korištenje emulatora, ali cilj usporedbe je vidjeti koji emulator radi brže i efikasnije
- Alati za generiranje dijagrama klasa – bit će uspoređeni alati za automatsko generiranje dijagrama klasa koji se mogu koristiti u razvojnim okruženjima
- Veličina aplikacije – nakon što je aplikacija završena, bit će testirano koliko zauzima prostora na uređaju
- Brzina pokretanja – ovaj kriterij za razliku od emulatora prikazivat će potrebno vrijeme za pokretanje aplikacije na samom mobilnom uređaju
- Pojava grešaka – tijekom usporednog razvoja aplikacije obratit ću pozornost na pojavu grešaka kod uključivanja vanjskih biblioteka ili same instalacije na fizički uređaj

Kriteriji	Xamarin	Android Studio	Pojašnjenje
Programski jezik	C#	Java	U ovom projektu osobno mi nije predstavljao problem korištenja jednog od ovih jezika jer sam relativno upoznata s oba, ali nekome tko nije vjerojatno će ovaj kriterij biti ključan kod izbora.

Dokumentacija	10/10	9/10	Bitno je napomenuti da oba alata imaju vrlo detaljno razrađenu dokumentaciju, ali ono što sam primijetila kod korištenja dokumentacije Android Studija je da više puta neke stvari su objašnjene korak po korak ali nedostaje neki ključan korak kako bi funkcionalnost proradila pa je potrebno potražiti dodatnu dokumentaciju u knjigama, člancima ili forumima, a kod korištenja Xamarin-a to se događalo puno rjeđe.
Emulatori	5/10	8/10	Pretpostavljam da korisnici koji imaju računala s više RAM-a, nemaju ovakvih problema ali nažalost pošto ja nemam preporučuje m svima koji su u istoj situaciji da aplikaciju testiraju na fizičkom mobilnom uređaju(ili više njih) jer iako je Android Studio u ovom slučaju odnio pobjedu svejedno se može reći da je emulator spor, a u slučaju Xamarin-a može se reći da je bolno spor. Tehnički oba emulatora rade i aplikacija se može testirati na više uređaja ali mojem slučaju polako i jako polako.
Alati za generiranje dijagrama klasa	10/10	7/10	Android Studio odnosno Visual Paradigm u ovom slučaju odnosi pobjedu, generirani dijagrami su jasni i detaljni te korisnik ima opciju biranja prikaza veza između klasa, dok Visual Studio nema tu opciju već generirani dijagram prikazuje samo strukturu klase te korisnik ručno mora dodavati veze između klasa.
Veličina aplikacije	6/10	10/10	Kada su aplikacije instalirane na istom uređaju(ponavljam, koriste iste resurse i imaju iste funkcionalnosti), Android Studio verzija aplikacije zauzima 2.86 MB interne memorije uređaja, a Xamarin verzija zauzima 99.48MB. Je li to u današnje vrijeme uistinu problem je za raspravu pošto prosječni mobilni uređaj ima minimalno 32GB(u većini slučajeva i više) interne memorije. Osobno na mobilnom uređaju(128GB) imam instalirano 42 aplikacije, 2508 slika, 230 videa, 481 ostalih datoteka i tek imam potrošeno 34GB memorije.

Brzina pokretanja	8/10	10/10	Ako aplikacija nije instalirana na mobilnom uređaju, i kliknete na gumb za pokretanje, Android Studio verziji trebat će otprilike 10-12 sekundi za pokretanje ili ako je već instalirana 3-4 sekunde, a Xamarin verziji 16-19 sekundi ili 4-6 ako je prethodno instalirana.
Pojava grešaka	9/10	7/10	Xamarin je u ovom slučaju (barem što se tiče mog iskustva) pobijedio. Tijekom izrade Xamarin verzije aplikacije svi vanjski dodaci aplikaciji(većinom NuGet paketi) integrirali su se s puno manje grešaka dok je u Android Studio verziji znalo doći do puno više problema, pogotovo kod usklađivanja Android verzija aplikacije. Također dosta puta mi se dogodilo da nije prepoznata neka metoda ili klasa iz dodanih biblioteka(ponekad i klase i metode iz samog projekta) te je trebalo par puta isključiti te ponovno pokrenuti Android Studio kako bi funkcionalnost proradila.

Tablica 5: Usporedba prema kriterijima

Završni rezultat usporedbe prema navedenim kriterijima koji su primijenjeni na moje osobno iskustvo u kreiranju mobilnih aplikacija i prikazani pomoću bodovne ljestvice od jedan do deset u šest kategorije (što znači da je maksimalan broj bodova 60) je 48 ukupnih bodova koji su dodijeljeni Xamarinu i 51 ukupnih bodova dodijeljenih Android Studiju. Objektivno gledajući (po kriterijima usporedbe) u ovom slučaju pobjeđuje Android Studio, ali moram uzeti u obzir da je ovaj slučaj jako specifičan. Programer (ja) poznaje i zna se koristiti s oba programska jezika (java i C#) i ima iskustva s oba razvojna okruženja, također ovdje se radilo o android aplikaciji koja trenutno nije namijenjena niti će vjerojatno biti drugim platformama (npr. iOS). Ali ako razmotrimo i druge slučajeve onda gore navedeni kriteriji padaju u vodu. Na primjer ako tvrtka posjeduje licencu za Visual Studio te u njoj rade .NET programeri i primi narudžbu za kreiranje mobilne aplikacije (može biti kao dodatni zahtjev originalne narudžbe) logičnije bi bilo koristiti Xamarin u takvoj situaciji, jer čak i ako programeri nisu upoznati sa strukturom mobilne aplikacije poznaju programski jezik i razvojno okruženje te uz edukaciju ne bi im trebao biti problem kreirati mobilnu aplikaciju, a ako se odluče koristiti Android Studio morat će zaposliti nove programere ili uložiti puno više vremena i financija u edukaciju trenutnih. Također još jedan mogući scenarij je da tvrtka dobije narudžba za Android aplikaciju, ali nije sigurno hoće li ta aplikacija u budućnosti trebati

funkcionirati i na drugim platformama. Ovo je malo kompliciraniji slučaj jer ako se tvrtka specijalizirala za izradu mobilnih aplikacija i ima iskusne timove od kojih je svaki specijaliziran za neku od platformi, neće im biti problem u budućnosti kreirati aplikaciju za drugu platformu i trenutno će vrlo vjerojatno odabrati Android Studio. Ako tvrtka nije specijalizirana za izradu mobilnih aplikacija na drugim platformama onda bi po mojem mišljenju bilo bolje odabrati Xamarin jer tim programera koji će raditi aplikaciji neće biti problem prilagoditi je i za druge platforme (90% koda možda i više koji je napisan pomoću Xamarin.Android alata može se bez problema koristiti i u Xamarin.Forms alatima koji služe za kreiranje aplikacija za više platformi ili koristiti druge alate kao Xamarin.iOS za izradu aplikacije namijenjene specifičnoj platformi) i nije potrebno imati različite programere odnosno timove programera za druge platforme jer će se koristiti isti programski jezik i isto razvojno okruženje. Što se tiče scenarija u kojem programer koji je upoznat sa samo jednim od navedenih programskih jezika dobije zadatak za kreirati Android aplikaciju a prije nikad nije radio mobilne aplikacije, preporučujem da odabere alat u kojem se koristi programski jezik s kojim je upoznat. I sada dolazimo do konačnog scenarija u kojem programer nije upoznat s niti jednim od navedenih jezika, nema iskustva s razvojnim okruženjima ni korištenim alatima i nije upoznat sa strukturom mobilne aplikacije ali ima želju u budućnosti kreirati mobilne aplikacije te se postavlja pitanje što mu preporučiti kao početniku? Odgovor koji bi osobno dala na to pitanje jest da nauči oba, odnosno da krene s izradom aplikacije upravo kako je predstavljeno u ovom radu, paralelno kreirajući dvije aplikacije i tako će se upoznati s oba programska jezika, razvojnim okruženjima i u budućnosti moći će odabrati način razvoja aplikacije koji će najbolje odgovarati zahtjevima.

7. Zaključak

Iz prvih poglavlja ovog rada gdje su među ostalim pojašnjene tehnologije razvoja mobilnih aplikacija prema predstavljenim primjerima mogu zaključiti da između mobilnih aplikacija kreiranih u javi i aplikacija kreiranih u C# programskom jeziku postoji mnogo sličnosti. Upravo ovaj zaključak postao mi je sve jasniji što sam dalje napredovala s radom jer i prije nego što sam počela sa samim razvojem mobilnih aplikacija iz teoretskog dijela rada moglo se naslutiti da u oba razvojna procesa predlaže se slijediti iste korake kao i koristiti iste arhitekture dizajna.

Kada sam krenula sa samim razvojem mobilnih aplikacija više nije bilo sumnje o mome početnom zaključku, pogotovo jer su aplikacije mogle koristiti istu bazu podataka kao i neke dijelove dizajna sučelja, čiji se XML kod mogao doslovno kopirati iz Android Studija u Visual Studio ili obrnuto i dizajn bi se prikazivao bez poteškoća. Moram priznati da kada sam tek krenula s radom, mislila sam da će mi biti problem što ću svaku funkcionalnost morati razvijati 2 puta u različitim razvojnim okruženjima koristeći različite programske jezike, ali tijekom samog procesa nije mi uopće bio problem paralelan razvoj. U nekim slučajevima kao što je bilo kod izračunavanja kilometara ili spremanja podataka pomoglo mi je što sam morala isti kod pisati u različitim jezicima jer sam lakše i brže prepoznala ako je došlo do logičkih grešaka u algoritmima. Također još jedna zanimljiva stvar koja mi se događala kada sam programirala previše sati bez pauze i kada mi je koncentracija počela padati je ta da sam počela pisati kod u programskom jeziku koji nije bio namijenjen toj verziji aplikacije i trebalo mi je nekoliko sekundi da shvatim zašto se javljaju greške. Kada bi projekte otvorili na GitHub-u (poveznice navedene u literaturi [28] i [29]) na prvi pogled jako su različiti, ali ako se kroz kodove prolazi kao u ovom radu u kojem je izvršena paralelna usporedba implementacije funkcionalnosti u različitim razvojnim okruženjima te su prikazani dijagrami klasa u obje verzije jasno je da projekti imaju mnogo zajedničkih stvari. Kao što sam navela u usporedbi ne može se jednostavno odrediti koji je način bolji za razvoj aplikacije, mogu se uvijek navesti kriteriji usporedbe i pomoću njih odrediti objektivnog pobjednika, ali u nekim slučajevima ti kriteriji neće svima odgovarati i imati istu važnost. Krajnji zaključak ovog rada je bez obzira na koji ste se programski jezik i razvojno okruženje odlučili, nemojte zanemariti korake prije samog pisanja koda, što znači detaljno definirajte funkcionalnosti aplikacije, isplanirajte arhitekturu, kreirajte kvalitetnu bazu i rasporedite na dijelove sami postupak razvoja jer bez obzira na znanje ili neznanje ovi koraci će vam uvelike olakšati kreiranje kvalitetne mobilne aplikacije.

Popis literature

[1] Anroid Developers, „Developer Guides“ list. 2021. [Na internetu] Dostupno: https://developer.android.google.cn/guide/, [pristupano 2.10.2021]

[2] Android, “Notes for proffesionals”, izd. 1, str. 1-1281, listopad-2021

[3] Neil Smith, „Android Studio 3.0 Development Essentials - Android 8 Edition“, izd 1, str 1-864, svibanj-2022.

[4] Altexsoft, „ The Good and The Bad of Xamarin Mobile Development“ stud. 2020. [Na internetu] Dostupno: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/ [pristupano 6.12.2021]

[5] Peter Martinez, „Can Xamarin Run on Linux and Everything abaout Xamarin Linux“ rujan 2020. [Na internetu] Dostupno: https://mockitt.wondershare.com/develop/xamarin-linux.html [pristupano 2.5.2022]

[6] Microsoft, „Xamarin.Android Documentation“ kol.2020. [Na internetu] Dostupno: https://docs.microsoft.com/en-us/xamarin/android/, [pristupano 5.11.2021]

[7] Benjamin Perikns, Jacob Vibe Hammer, Jon D. Reid, „Beginning Visual C# 2015 Programming“, izd 1, str 1-1226, svibanj-2022.

[8] Tutorials point „Xamarin-Android Activity Lifecycle“ [Na internetu] Dostupno: https://www.tutorialspoint.com/xamarin/xamarin_android_activity_lifecycle.htm [pristupano 10.5.2022.]

[9] Tutorials point „Android“ [Na internetu] Dostupno: https://www.tutorialspoint.com/android [pristupano 2.5.2022.]

[10] Rakesh Trivedi „Understand Content Provider In Xamarin With Visual Studio“ [Na internetu] Dostupno: https://www.c-sharpcorner.com/article/understand-content-provider-in-xamarin-with-visual-studio/ [pristupano 30.4.2022]

[11] Xamarin Android Guide - Peruzal, „Android Views“ [Na internetu] Dostupno: https://guides.peruzal.com/xamarin-android/android-views/ [pristupano 17.5.2022]

[12] Google Developer Training Team, „Android Developer Fundamentals Course, izd 1, str. 1-491, pros-2021

[13] Charles Petzol, “Creating Mobile Apps with Xamarin.Forms“ , izd. 1, str. 1-1122, studeni-2021

[14] Peerbits „6 Crucial Rules for Creating Award – Winning App Designs“ [Na internetu] Dostupno:https://www.peerbits.com/blog/mobile-app-designing-rules.html [pristupano 20.5.2022]

[15] RIP Tutorial „Basic Toast Message“ [Na internetu] Dostupno: https://riptutorial.com/xamarin-android/example/12241/basic-toast-message[pristupano 20.5.2022]

[16] C# Corner „Toast Message for Android“ [Na internetu] Dostupno:https://www.c-sharpcorner.com/article/working-with-xamarin-toast-message-for-android/[pristupano 21.5.2022]

[17] PHPPOT „Rest API for Android“ [Na internetu] Dostupno:https://phppot.com/php/php-mysql-rest-api-for-android/ [pristupano 21.5.2022]

[18] Appinventiv „22 Best Android Libraries for 2022“ [Na internetu] Dostupno: https://appinventiv.com/blog/best-android-libraries/ [pristupano 26.5.2022]

[19] GeeksforGeeks „MVC (Model View Controller)“ [Na internetu] Dostupno: https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/ [pristupano 28.5.2022]

[20] Scalable path „A Guide to Google' s Recommended Android Apps Architecture“ [Na internetu] Dostupno:https://www.scalablepath.com/android/android-apps-architecture [pristupano 28.5.2022]

[21] Microsoft „MVVM (Model-View-ViewModel) Pattern“ [Na internetu] Dostupno: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm [pristupano 30.5.2022]

[22] Invonto „Mobile App Development Process“ [Na internetu] Dostupno: https://www.invonto.com/insights/mobile-app-development-process/ [pristupano 1.6.2022]

[23] Visual Paradigm Online „Dijagram baze podataka“ [Na internetu] Dostupno: https://online.visual-paradigm.com/ [pristupano 2.7.2022]

[24] Statista „Most used programming languages among developers worldwide as of 2021“ [Na internetu] Dostupno: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ [pristupano 5.7.2022]

[25] InVision Studio „InVision Studio download“ [Na internetu] Dostupno: https://projects.invisionapp.com/d/studio/download [pristupano 18.7.2022]

[26] Visual Paradigm „How to integrate Visual Paradigm with Android Studio“ [Na internetu] Dostupno:https://circle.visual-paradigm.com/docs/ide-integration/android-studio-integration/how-to-integrate-visual-paradigm-with-android-studio/ [pristupano 20.7.2022]

[27] Visual Studio „Design and view classes and types with Class Designer“ [Na internetu] Dostupno: https://docs.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2022 [pristupano 21.7.2022]

[28] Github „Kod Aplikacije kreirane u Android Studiju“ Dostupno: https://github.com/marfioren/Diplomski_AndroidStudio

[29] Github „Kod Aplikacije kreirane u Visual Studiju“ Dostupno: https://github.com/marfioren/Diplomski_VisualStudio

Popis slika

Popis tablica

Popis isječaka koda