SUSTAV ZA UMREŽAVANJE IT PROFESIONALACA

Rennert, Matej

Master's thesis / Specijalistički diplomski stručni

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:589493

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**



Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College





VISOKO UČILIŠTE ALGEBRA

DIPLOMSKI RAD

SUSTAV ZA UMREŽAVANJE IT PROFESIONALACA

Matej Rennert

Zagreb, Kolovoz 2019.

Predgovor

Zahvaljujem se mentoru dr. sc. Vedranu Juričiću na stručnim savjetima i potpori pri izradi diplomskog rada te svojoj obitelji i prijateljima na potpori tijekom trajanja studija.

Sažetak

U radu je detaljno provedeno istraživanje praćenja performansi i vremena izvršavanja procesa te su prikazane prednosti i ograničenja korištenja modula u razvoju. Cilj rada je usporediti kako modulariziranje aplikacije utječe na sveukupni razvoj projekta u odnosu na klasični pristup razvoju i prikazati je li moguće rastaviti komponente aplikacije na module, a da se zadrži pravilan i nesmetan pristup i rad aplikacije. U svrhe istraživanja kreiran je projekt s temom sustava za umrežavanje IT profesionalaca. Projekt se sastoji od autentifikacije i validacije korisnika, komunikacije porukama, objavljivanju i djeljenju sadržaja te povezivanju korisnika. Rad detaljno opisuje tehnologije i alate koje projekt koristi za implementaciju komponenti sustava, uključujući primjer implementacije arhitekturalnog obrasca za razvoj programske podrške u modulima. Osim toga, prikazani su primjeri proširenja na ostale Android platforme i mogućnost razvoja novih komponenti aplikacije. Za kreiranje projekta koristi se Android Studio razvojno okruženje, a kod projekta pisan je u Java programskom jeziku.

Ključne riječi: Android, Mobilna aplikacija, Room baza podataka, MVP arhitektura, Cloud Firestore, moduli, Gradle

Abstract

This paper thoroughly explores performance monitoring and process execution time and presents the benefits and limitations of using modules in software development. The goal of the paper is to compare how modularization affects on the overall development of a project relative to the classic approach, and to show whether it is possible to divide application components into modules while maintaining proper and unobstructed access and operation of the application. For the purposes of the research, a project was created with the theme of networking systems for IT professionals. The project consists of user authentication and validation, messaging, posting and sharing content, and connecting users. The paper describes the technologies and tools used by the project to implement system components, including an example implementation of an architectural pattern for software development in modules. In addition, examples of extensions to other Android platforms and opportunities to develop new application components are presented. The Android Studio development environment was used to create the project, and the project code was written in the Java programming language.

Key words: Android, Mobile application, Room Database, MVP architecture, Cloud Firestore, modules, Gradle

Sadržaj

1. Uvod ... 1

2. Arhitektura i funkcionalnost sustava ... 2

2.1. Komponente i moduli aplikacije ... 2

2.2. Model View Presenter obrasac i implementacija ... 3

3. Istraživanje ... 7

3.1. Utjecaj modula na performanse aplikacije ... 7

3.2. Utjecaj modula na vrijeme i kompleksnost razvoja ... 9

3.3. Prednosti i ograničenja korištenja modula ... 13

4. Tehnologije i alati ... 17

4.1. Sigurnost i autentifikacija korisnika ... 17

4.2. Računalni oblak za spremanje i upravljanje podatcima ... 18

4.3. Komunikacija porukama ... 19

4.4. Lokalni rad s podatcima ... 21

5. Mogućnosti poboljšanja aplikacije ... 24

5.1. Dodatne funkcionalnosti ... 24

5.2. Proširenja na druge Android platforme ... 25

Zaključak ... 27

Popis kratica ... 28

Popis slika ... 29

Popis kôdova ... 31

Literatura ... 32

1. Uvod

Android programeri se pri planiranju razvoja projekta često susreću s problemima organiziranja strukture i arhitekture projekata. Zbog toga je modulariziranje aplikacije sve češća tema rasprave na Google I/O i Droidcon konferencijama.

Modulariziranje se koristi u svrhu rastavljanja aplikacije na komponente pružajući pouzdanost, skalabilnost, ponovnu iskoristivost i čitkost koda. Dobro oganizirana arhitektura modula omogućava robusnost, održivost, testiranje i održavanje komponente. Druga cjelina opisuje što su moduli, koje vrste modula postoje i kako ih se može kombinirati sa Model View Presenter (MVP) obrascem.

U trećoj cjelini u svrhe istraživanja rada razvijena je Android Nerdo aplikacija koja služi kao sustav za umrežavanje IT profesionalaca. Aplikacija se sastoji od trideset i četiri zasebno implementirane komponente. Cilj istraživačkog djela rada je prikazati kako rastavljanje komponenti na module, u odnosu na klasični pristup razvoja aplikacije, utjeće na kompleksnost razvoja. Na temelju projekta, korištenjem alata, provedeno je istraživanje o praćenju performansi i vremenu izvršavanja procesa te su prikazane prednosti i ograničenja korištenja modula u razvoju.

Četvrta cjelina opisuje glavne tehnologije koje aplikacija koristi, a to su: korištenje računalnog oblaka za spremanje i upravljanje podatcima u realnom vremenu, komunikacija porukama između korisnika, autentifikacija korisnika korištenjem otiska prsta, Facebook i Google sustav za prijavu u aplikaciju te spremanje i upravljanje lokalnim podatcima na uređaju.

Google sadrži niz platformi za razvoj Android aplikacija. U petoj cjelini vidljive su mogućnosti poboljšanja projekta implementirajući nove komponente i vrste platformi na koje se projekt može proširiti.

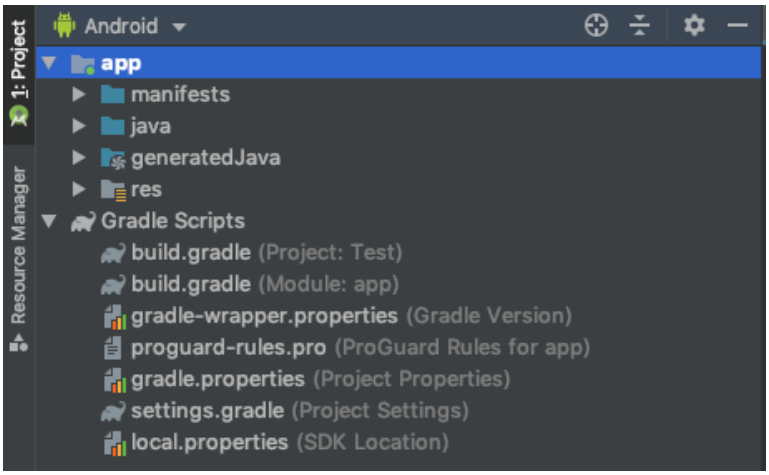
2. Arhitektura i funkcionalnost sustava

Ovaj dio rada objašnjava što su komponente, kako se komponetne mogu rastaviti na module, što je modul i koje vrste modula postoje. Također prikazuje što su obrasci, podjelu obrazaca po kategorijama, arhitekturu koju moduli koriste i opisuje primjer implementacije arhitekture u modulu.

2.1. Komponente i moduli aplikacije

Svaka aplikacija, ovisno o kompleksnosti sadržaja aplikacije, sastoji se od jedne ili više komponente. Komponenta predstavlja dio koda koji izvršava svoju specifičnu namjenu. Na primjer, prijava korisnika u aplikaciju koristeći Facebook alate za razvijanje softwera (engl. s*oftware development kit*, skraćeno SDK), predstavljala bi jednu komponentu aplikacije.

Ideja Nerdo projekta bazira se na rastavljanju komponenti aplikacije na module, što najbolje može asocirati na puzle. Modul je skup izvornih datoteka i postavki aplikacije koje omogućuju podjelu projekta na zasebne jedinice komponenti. Projekt može imati jedan ili više modula, a moduli mogu ovisiti jedan o drugome. Svaki modul se može samostalno razvijati, testirati i *debugirati*. Pri kreiranju projekta, Android Studio automatski kreira *app* modul sa svim potrebnim datotekama i resursima za razvoj aplikacije.



Slika 2.1 Prikaz app modula kod kreiranja novog projekta

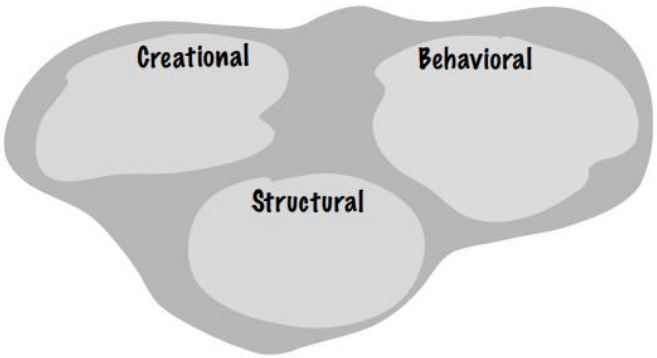
Prilikom kreiranja novog modula Android Studio nudi opciju za odabir vrste modula koji se želi implementirati. Android *app* moduli se odnose na implementaciju modula po Android platformama. *Library* modul sadrži sve datoteke i resurse kao i *app* module, s time da se isti *library* može iskoristiti u više modula i projekata. Google Cloud modul omogućava podršku za razvoj *backenda* unutar istog projekta gdje se razvija i mobilna aplikacija.

U verziji 3.2.0 Android Studia, Google Play je predstavio novi oblik posluživanja aplikacije zvan *Dynamic Delivery* koji koristi *App Bundle* kako bi se korisniku isporučio kod i resursi koji su potrebni za rad aplikacije. U tu svrhu pojavljuje se *Dynamic feature* modul koji se odnosi na implementaciju komponenti koje će korisnicima biti ponuđene tek kada zatraže pristup navedenoj komponenti.[1,2]

2.2. Model View Presenter obrasac i implementacija

Oblikovni obrasci (engl. *Software Design Patterns*) potječu iz 1977/78. godine kada ih je Christopher Alexander predstavio kao arhitektonski koncept, a popularnost su stekli 1994. godine nakon izdavanja knjige *Design Patterns: Elements of Reusable Object-Oriented Software*.

Oblikovni obrazac se može definirati kao rješenje kojemu se teži kako bi se postigao cilj za situaciju u kojoj se obrazac koristi. Prilikom razvoja koriste kao najbolja praksa za rješenje uobičajenih problema pri planiranju softwera. Grupiraju se po kategorijama kao kreacijski, strukturni i bihevioralni obrasci.[3]

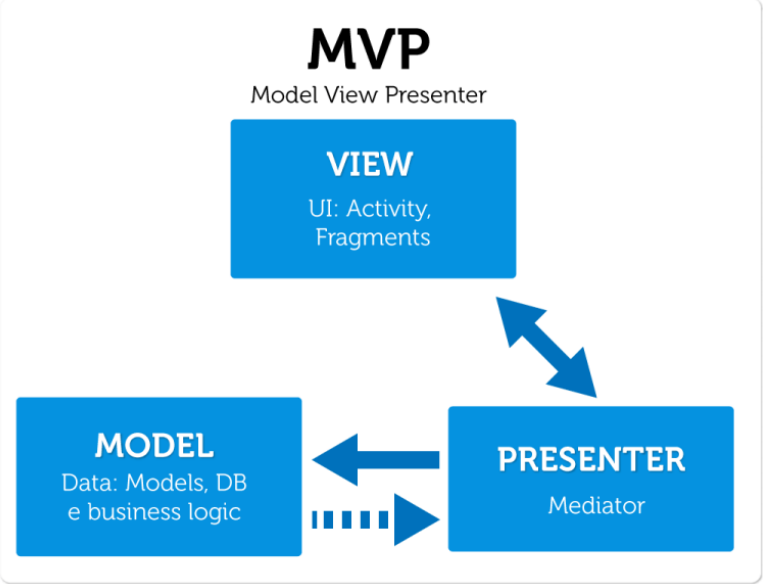


Slika 2.2 Prikaz sheme Design Pattern klasa

Kreacijski obrasci uključuju instanciranje objekata i svi obrasci u kreacijskoj grupi pružaju način da se klijent odvoji od objekta kojeg je potrebno instancirati. Strukturni obrasci se odnose na sastav klasa i objekata te omogućavaju sastavljanje u veće strukture.

Bihevioralni obrasci identificiraju zajedničke komunikacijske obrasce između objekata i povećavaju fleksibilnost komunikacije ostvarujući raspoređivanje odgovornosti. Neovisno odnose li se obrasci na klase ili objekte, mogu se kategorizirati u objektne obrasce (engl: *Object Patterns*), koji se bave kreiranjem objekata i klasne obrasce (engl. *Class Patterns*), koji se bave instanciranjem klasa.[4]

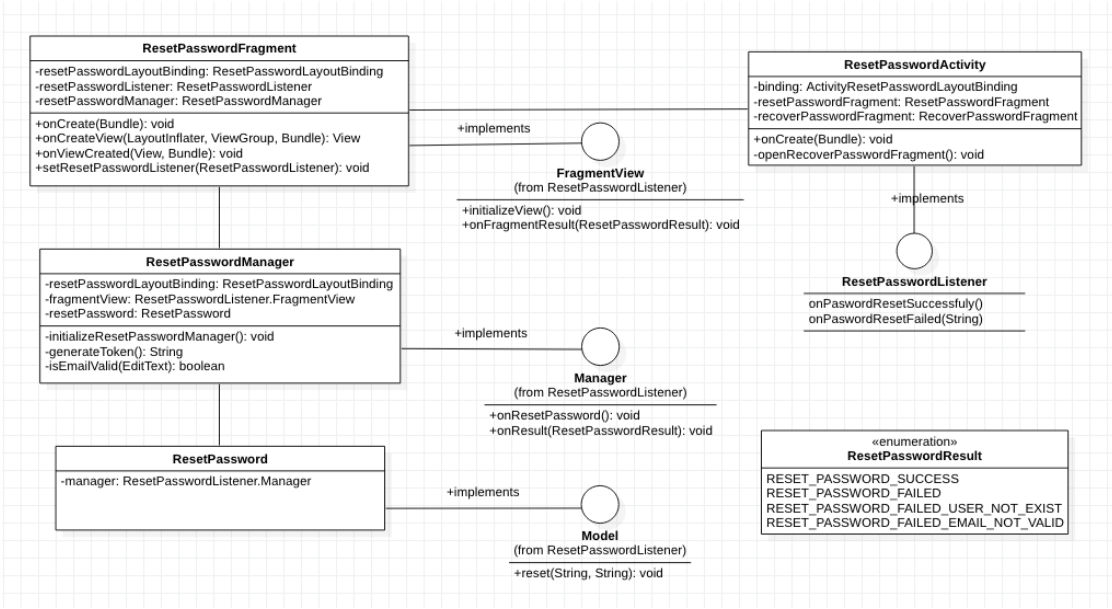
Kao klasifikacija predstavljena je ideja arhitektonskih obrazaca koji se mogu koristiti na softwerskoj arhitekturalnoj razini.[3] Arhitektonski obrasci sadrže širu sliku u rješavanju različitih pitanja softwerskog inženjerstva poput ograničenja performansi, visoke dostupnosti i minimiziranja poslovnog rizika.[5]



Slika 2.3 Prikaz sheme Model View Presenter obrasca

Model View Presenter (MVP) obrazac je nastao kao izvedba Model View Controller (MVC) obrasca, jednog od najpoznatijih obrazaca koji je korišten za implementaciju grafičkog korisničkog sučelja (engl. *graphical user interface*, skraćeno GUI) kojega je razvio Smalltalk 1970. godine.[6] MVP obrazac razdvaja *View* od *Modela* koristeći *Presenter* koji odlučuje što će se prikazati unutar *View*-a.[7]

Sve komponente unutar Nerdo aplikacije koriste MVP obrazac za dohvat i prikaz podataka. U nastavku je prikazan primjer implementacije komponente za resetiranje lozinke korisnika.



Slika 2.4 Primjer implementacije MVP arhitekture u komponenti za resetiranje lozinke

U primjeru implementacije MVP obrasca `ResetPasswordFragment` klasa predstavlja *View*. *View* ne sadrži implementiranu logiku, već se koristi samo za prikaz podataka korisniku. `ResetPasswordFragment` klasa implementira `FragmentView` sučelje (engl. *interface)* koje sadrži dvije metode, `initializeView` i `onFragmentResult`. `initializeView` metoda postavlja *event* na klik na gumb za generiranje tokena, a `onFragmentResult` metoda javlja glavnoj aktivnosti rezultat po kojemu se izvode sljedeće akcije. Token je specifično generirani osmeroznamenkasti broj koji služi za validaciju korisnika koji želi napraviti izmjenu lozinke.

`ResetPasswordManager` klasa predstavlja *Presenter* koji se ponaša kao posrednik između *Viewa* i *Modela*. Kada korisnik unese podatak o adresi elektroničke pošte i pritisne gumb za generiranje tokena, pokreće se `onPasswordReset` metoda koja procesira uneseni podatak. Ako je podatak validan kontaktira se *Model* klasa.

`ResetPassword` klasa predstavlja *Model*. *Model* sadrži logiku za dohvat podataka putem lokalne baze podataka ili sa servera. Kada `ResetPassword` klasa vrati rezultat pomoću instanciranog `manager` parametra poziva se `onResult` metoda kako bi se rezultat prosljedio `ResetPasswordManager` klasi.

Nakon što `ResetPasswordManager` od *Modela* zaprimi rezultat, procesira ga i na temelju definiranih konstanti pokuša kreirati poruku koja se prosljeđuje `ResetPasswordFragment` klasi koristeći `onFragmentResult metodu`.

Kada `ResetPasswordFragment` zaprimi rezultat, provjeravaju se dva stanja poruke. Ukoliko poruka postoji, poziva se `onPasswordResetFailed` metoda i prosljeđuje poruka da resetiranje lozinke nije uspješno. Ako se umjesto poruke zaprimi *null* vrijednost, poziva se `onPasswordResetSuccessfuly` metoda te se obavještava da je token za reset lozinke uspješno generiran te korisnik može postaviti novu lozinku.[8]

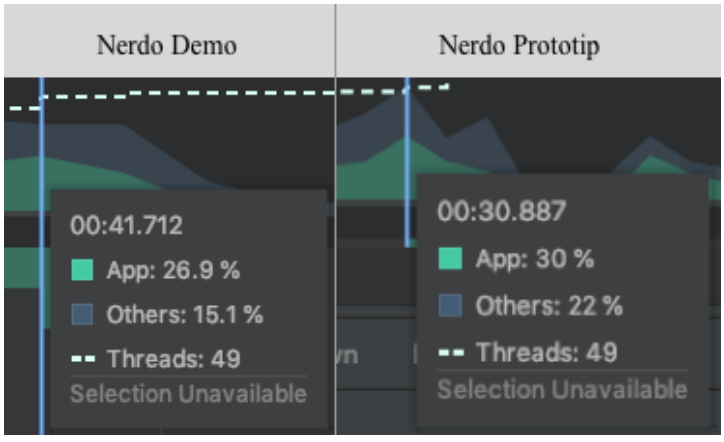
3. Istraživanje

U svrhu istraživanja razvijene su dvije vrste Nerdo projekata, Demo i Prototip. Demo projekt se odnosi na aplikaciju u kojoj su sve komponente implementirane unutar *app* modula. Prototip projekt odnosi se na aplikaciju u kojoj je svaka komponenta razvijena u zasebnom modulu. Kako bi se postigli bolji rezultati istraživanja, obje aplikacije sastoje se od jednakog broja komponenti iste implementacije. U nastavku je prikazano kako modulariziranje aplikacije utječe na performanse i razvoj aplikacije. Za istraživanje korišteno je prijenosno računalo MacBook Air (2015), Android Studio 3.4.1. razvojno okruženje i mobilni telefon Sony X-Peria Z5 Premium (Android 7).

3.1. Utjecaj modula na performanse aplikacije

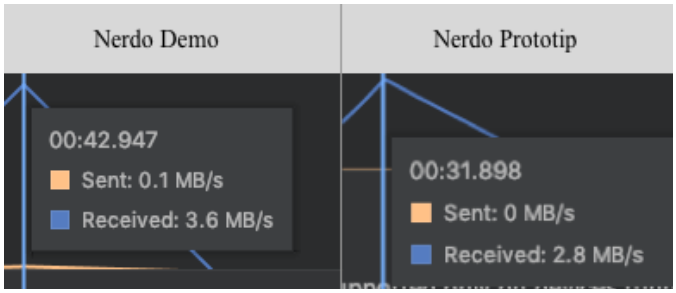
U Android Studio verziji 3.0 Google je predstavio novi alat za praćenje performansi aplikacije zvan Android Profiler. Alat u trenutnom radu aplikacije omogućava mjerenje procesora (engl. *central processing unit*, skraćeno CPU), mreže i praćenje memorije. Od Android 8 verzije OS-a dostupno je i praćenje trenutne potrošnje energije pri radu aplikacije koje se odnose na analizu CPU, mreža i praćenje lokacija.[9]

Mjerenje CPU koristi se za identifikaciju uskog grla u performansama aplikacije. Slika 3.1. prikazuje korištenje CPU-a u trenutnom vremenu pri korištenju aplikacije i broj dretvi koje aplikacija koristi. Polje ostalo (engl. others) prikazuje postotak korištenja CPU-a drugih aplikacija.[10]



Slika 3.1 Prikaz odnosa mjerenja CPU

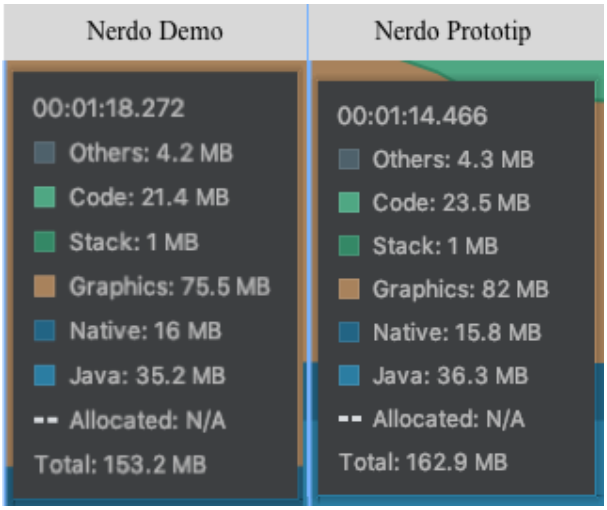
Mjerenje mreže koristi se kako aplikacija ne bi sadržavala prekomjerenu propusnost mreže. Na slici 3.2. moguće je vidjeti kada i koji podatci se šalju ili primaju. Također, unutar alata moguće je vidjeti broj konekcija na mrežu, poziv (engl. *request*) i odgovor (engl. *response*) za dohvat podataka.[11]



Slika 3.2 Prikaz odnosa mjerenja mreže

Mjerenje memorije koristi se za osiguranje da aplikacija koristi odgovarajuću količinu memorije iz resursa. Preveliko korištenje memorije može dovesti do problema sa prikazom korisničkog sučelja, zamrzavanjem aplikacije ili čak i rušenjem aplikacije. Slika 3.3. prikazuje korištenje memorije po određenim djelovima aplikacije, a to su:

- *Others* - odnosi se na memoriju koju sustav ne zna kategorizirati
- *Code* - odnosi se na memoriju koju aplikacija koristi za kod i resurse
- *Stack* - odnosi se na memoriju koliko dretvi aplikacija koristi
- *Graphics* - odnosi se na memoriju korištenu za prikaz piksela na ekranu
- *Native* - odnosi se na memoriju objekata alociranih u C ili C++ kodu
- *Java* - odnosi se na memoriju objekata alociranih u Java kodu[12]



Slika 3.3 Prikaz odnosa mjerenja potrošnje memorije

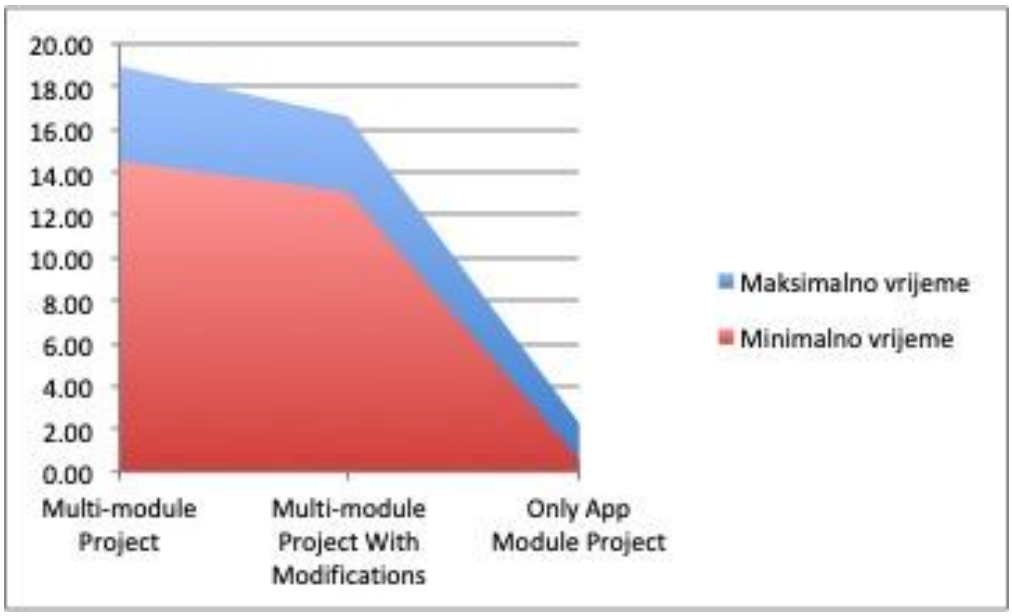
3.2. Utjecaj modula na vrijeme i kompleksnost razvoja

Osnovne i najčešće akcije pri razvoju android aplikacije su: proces čišćenja i ponovnog kreiranja (engl. *clean build*) projekta, kompajliranje aplikacije i kreiranje *debug* i *release* apk (engl. *Android Package*, skraćeno APK) datoteke.

U istraživanju navedenih procesa dodana je poboljšana verzija Nerdo Prototip aplikacije. Poboljšanja se odnose na međuovisnost između modula. Modul za komunikaciju s Firestore bazom podataka ne implementira bazni modul već se dohvaćeni podatci prenose kao mape. Svrha poboljšane verzije aplikacije je provjeriti utječu li prekomjerne ovisnosti o modulima na vrijeme izvršavanja procesa.

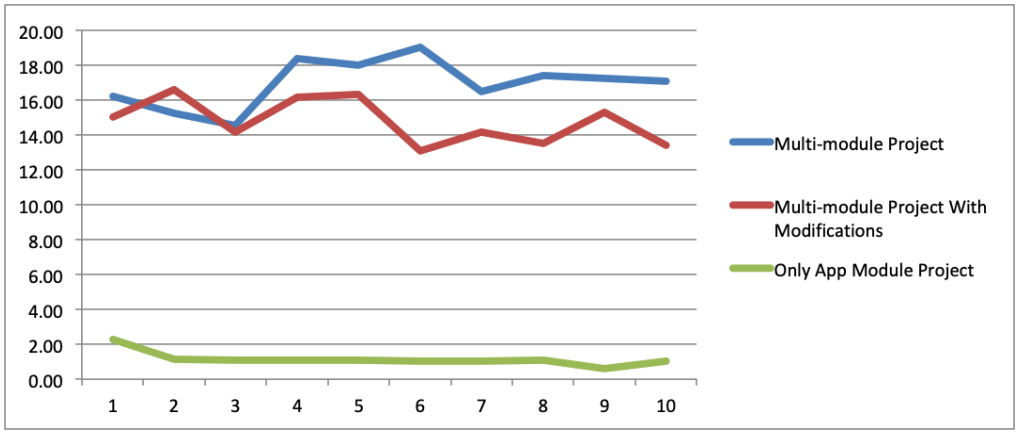
Proces čišćenja i ponovnog kreiranja projekta spada u osnovnu akciju koja se ne koristi često prilikom razvoja, ali je u određenim situacijama vrlo korisna. Prema Gradle dokumentaciji, proces čišćenja se odnosi na brisanje *build* direktorija što rezultira uklanjanju unaprijed sastavljenih *outputa* te povećanju vremena izvršavanja sljedećih zadataka. Proces ponovnog kreiranja odnosi se na ponovno sastavljanje svih *outputa* i izvršavanje svih provjera. Unutar konzole procesi se mogu pokretati skupa ili zasebno. Za pokretanje procesa čišćenja koristi se naredba `gradlew clean`, za pokretanje procesa ponovnog kreiranja koristi se naredba `gradlew build`, a za istraživanje se koristi zajednička naredba `gradlew clean build`.[13]

Odnos minimalnog i maksimalnog vremena trajanja procesa po projektu pokazuje velika odstupanja između aplikacija u rezultatima istraživanja procesa čišćenja i ponovnog kreiranja. (Slika 3.4.)



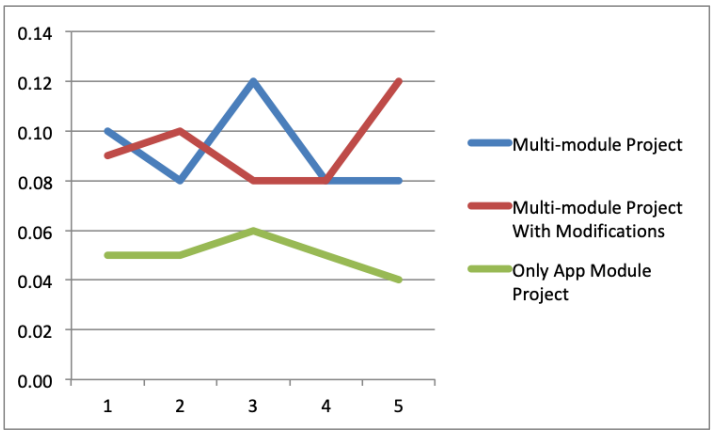
Slika 3.4 Prikaz izvršavanja vremena procesa čišćenja i ponovnog kreiranja po projektu

Navedeni podataci prikazuju da Demo projektu treba najmanje vremena za izvršavanje procesa. Prosjećno vrijeme za deset pokrenutih procesa Demo projekta iznosilo je 1 min i 26 s. U odnosu na modularizirane projekte, vrijeme izvršavanja je impresivno. Prosjećno vrijeme Prototipa kreće se oko 16 min i 47 s, dok modificirana verzija ima malo bolje vrijeme od 14 min i 51s. (Slika 3.5.)



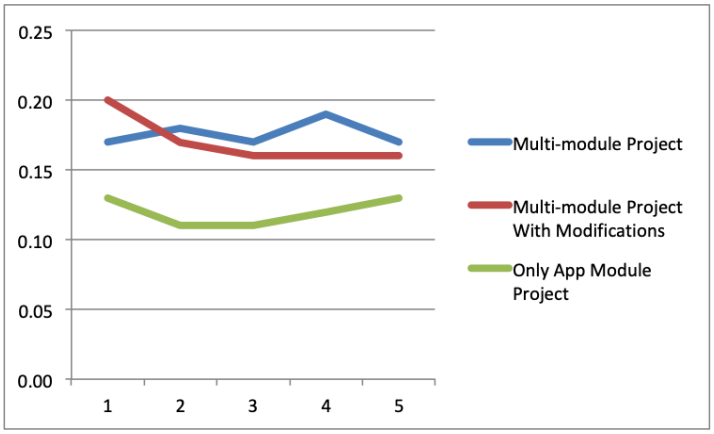
Slika 3.5 Prikaz izvršavanja vremena procesa čišćenja i ponovnog kreiranja u odnosu na broj procesa

Sljedeći proces odnosi se na kreiranje *debug* APK datoteke. Najčešće se koristi u testne svrhe ili u slučaju kada je potrebno samostalno instalirati aplikaciju na fizički uređaj. Proces započinje upisivanjem `assembleDebug` naredbe u konzolu. Na slici 3.6. može se vidjeti vrijeme trajanja procesa kada se radi promjena unutar jedne komponente aplikacije.



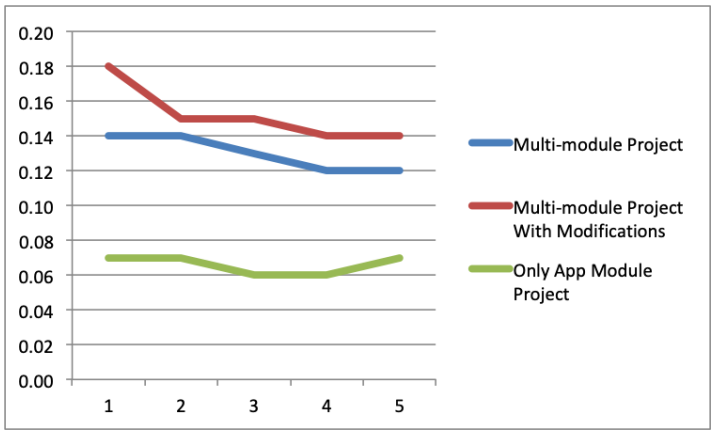
Slika 3.6 Prikaz izvršavanja vremena procesa kreiranja debug APK datoteke pri izmjeni jedne komponente

Iako se radi o sekundama, unutar pet pokretanja Demo projekt i dalje ima najbolji rezultat sa prosjećnim vremenom od 5 s, dok Prototip i modificirana Prototip aplikacija imaju jednako prosjećno vrijeme od 9 s. Pri mjerenju rezultata izmjene u više komponenti aplikacije, može se primjetiti da je vrijeme u svim projektima duplo veće. (Slika 3.7.)



Slika 3.7 Prikaz izvršavanja vremena procesa kreiranja debug APK datoteke pri izmjeni u više komponenti

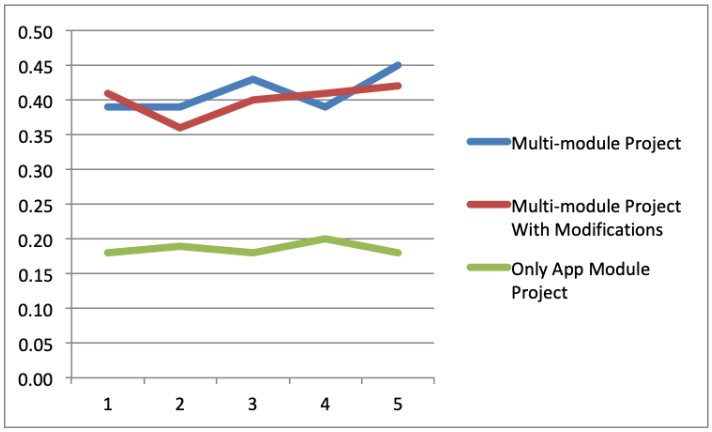
Proces za kreiranje release APK datoteke predstavlja veliku važnost u razvoju Android aplikacija. Prilikom pokretanja procesa Android nudi opciju korištenja obfuskacije koda, što omogućava preimenovanje klasa i metoda u kodu. APK datoteka koja se kreira postavlja se na Google Play trgovinu te je dostupna korisnicima diljem svijeta. Kao i u prethodnom istraživanju, može se vidjeti utjecaj vremena na proces kreiranja release APK datoteke nakon promjene u jednoj funkcionalnosti. (Slika 3.8)



Slika 3.8 Prikaz izvršavanja vremena procesa kreiranja release APK datoteke pri izmjeni jedne komponente

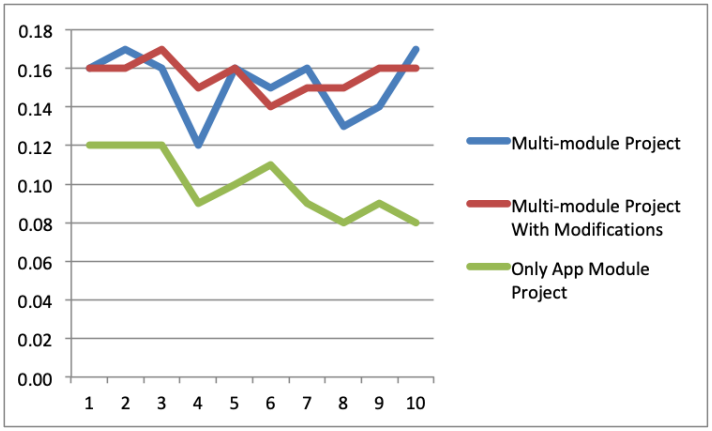
U pet pokretanja, Demo projekt je najbrži s prosjekom od 7 s, Prototip projekt završava s 13 s, a Prototip projekt s modifikacijama s 15 s. Prethodna dva slučaja ishodila su boljim rezultatima modificiranog projekta u odnosu na Prototip.

Rezultati procesa vremena kreiranja release APK datoteke pri promjeni u više komponenti aplikacije porasli su do tri puta u odnosu na izmjenu na jednoj komponenti. (Slika 3.9)



Slika 3.9 Prikaz izvršavanja vremena procesa kreiranja release APK datoteke pri izmjeni jedne komponente

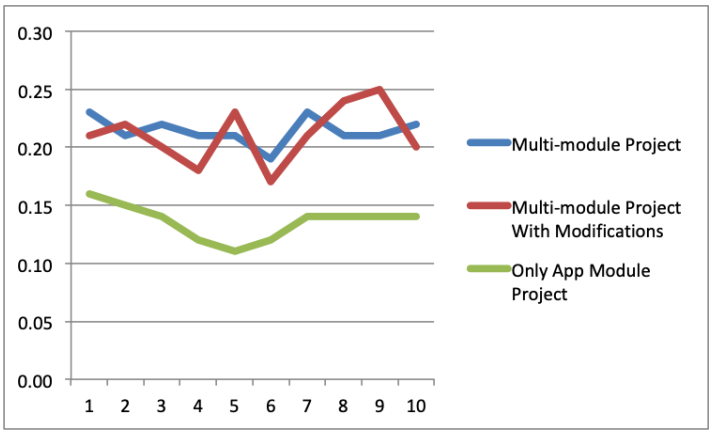
Zadnji proces je kompajliranje, najkorištenija akcija u razvoju aplikacija, a odnosi se ne prikaz promjena u kodu.



Slika 3.10 Prikaz izvršavanja vremena procesa kompajliranja aplikacije pri izmjeni jedne komponente

U deset pokrenutih procesa Demo projekt je izvršen u prosjeku od 10 s, a Prototip projekt u prosjeku od 15 s te je u odnosu na modificirani projekt bio bolji za 1 s.

Pri izmjeni u više komponenti aplikacije, vrijeme kompajliranja Demo projekta naraslo je na prosjek od 14 s, a za Prototip i modificirani projekt na 21 s. (Slika 3.11)



Slika 3.11 Prikaz izvršavanja vremena procesa kompajliranja aplikacije pri izmjeni u više komponenti

3.3. Prednosti i ograničenja korištenja modula

Najveća prednost rastavljanja komponenti na module vidljiva je u prikazu arhitekture projekta. Svaki modul kreiran je kao zasebna cjelina te razvoj ili izmjena arhitekture implementacije određene komponente ne predstavlja veliki problem. Dodatnu prednost predstavlja mogućnost reiskoristivosti istih funkcionalnosti u više projekata. Prilikom

razvoja Android Studio nudi opciju mogućnosti prikaza i skrivanja modula. Trenutno se prilikom skrivanja moduli uklone sa liste strukture projekta. Navedena funkcionalnost korisna je ukoliko aplikacija sadrži veliki broj modula što omogućava programerima lakšu navigaciju i pronalazak potrebnih datoteka za rad. Skrivanje modula bilo bi od velike koristi kada bi se skriveni moduli uklonili iz projekta što bi moglo utjecati na smanjenje vremena kompajliranja projekta.

Svaki modul može sadržavati određene resurse koji su potrebni za rad. Neki od resursa mogu biti datoteke: sučelja, boja, dimenzija, jezika, slika, crteža. Prilikom postavljanja resursa potrebno je voditi računa o imenovanju jer se prilikom kompajliranja aplikacije svi resursi spajaju u jedan. Isto vrijedi za *AndroidManifest* datoteku pri postavljanju dopuštenja ili definiranju servisa.

Ukoliko se određena funkcionalnost u aplikaciji više ne planira koristiti te je odlučeno da se ukloni iz projekta, modulariziranje aplikacije nudi brzo rješenje. Potrebno je unutar *settings.gradle* datoteke ukloniti ovisnost projekta o modulu. Zatim je potrebno ukloniti ovisnosti u svim ostalim modulima koji su koristili komponentu i sinkronizirati aplikaciju s gradle datotekama. U zadnjem koraku sinkronizacija će javiti grešku i prikazati klase u kojima se navedena funkcionalnost koristila. U klasičnom pristupu razvoja aplikacije korištenjem *app* modula, brisanje komponente unutar koda može biti puno kompliciranije. Brisanju se može pristupiti na dva načina, uklanjati metodu po metodu dok se niti jedna metoda iz komponente više ne koristi ili brisati kompletne komponente i pokušati kompajlirati i brisati dok se svi pozivi metoda ne uklone.

Jedno od ograničenja korištenja modula predstavlja Gradle. Gradle je automatizirani skup alata za izradu aplikacija. Prilikom kreiranja novog *Library* modula Android Studio kreira Gradle datoteku i postavlja osnovne postavke kako bi se aplikacija mogla kompajlirati i pokrenuti. Datoteka započinje sa postavljanjem *plugina* za korištenje *android librarya*. Zatim se postavlja *android* odlomak koji sadrži *defaultConfig* sekciju za osnovne informacije o projektu kao što je postavljanje SDK verzije te *BuildTypes* sekcije koja sadrži informacije hoće li se u *release* verziji aplikacije koristiti obfuskacija koda za preimenovanje klasa i metoda. Nadalje se postavlja odlomak sa ovisnostima (engl. *dependencies*) aplikacije o lokalnim java datotekama (engl. *Java ARchive*, skraćeno JAR) i vanjskim bibliotekama.

Ograničenja se pojavljuju u situaciji kada se kreira više modula te se ovisnosti dupliraju. Na primjer, dva različita modula koja implementiraju *RecyclerView* listu. Prilikom

kompajliranja aplikacije može doći do problema zbog dupliciranja ovisnosti. Uz *defaultConfig* polje u kojemu se postavlja SDK verzija, ovisnosti također koriste verziju biblioteke stoga može doći do problema pri održavanju i unapređenju komponenti aplikacije. Navedeni problemi se mogu riješiti pomoću kreiranja baznog modula koji će koristiti sve verzije i ovisnosti koje koriste i ostali moduli. Navedeno se može postići tako da se u gradle datoteci umjesto postavljanja *plugina* za *android library* postavi putanja do gradle datoteke baznog modula.

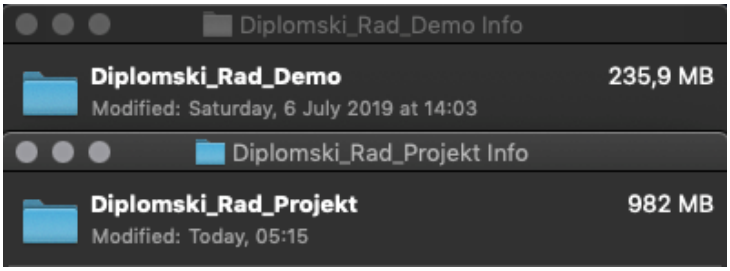
```
apply from: "$rootProject.projectDir/core/build.gradle"
```

Takva implementacija onemogućava korištenje različitih verzija bibloteka u više različitih modula. Ukoliko neki modul zahtjeva korištenje dodatnih ovisnosti ili postavki, moguće ih je dodati u gradle datoteku.

U postavkama projekta zadana opcija paralelnog izvršavanja procesa je isključena što znatno usporava vrijeme kompajliranja aplikacije. Da bi se omogućilo paralelno izvršavanje procesa za svaki modul, u datoteci za Gradle postavke projekta `gradle.properties` potrebno je omogućiti opciju za paralelno izvršavanje. Navedena opcija omogućava istovremeno izvršavanje procesa za više modula.

```
org.gradle.parallel=true
```

Prilikom rada s modulima primjećen je nedostatak vezan za resurse aplikacije. Kod kreiranja modula Android Studio automatski kreira i *Library* mapu koja sadrži sve potrebne resurse za razvoj. Do *Library* mape može se navigirati tako da se odabere prikaz paketa (engl. *packages*), otvori modul paket i proširi *Library* mapa. Iako se, kao i ostali resursi prilikom kompajliranja projekta spajaju i ne utječu na veličinu APK datoteke, može se primjetiti da se veličine Demo i Prototip projekta spremljenog na disku znatno razlikuju.

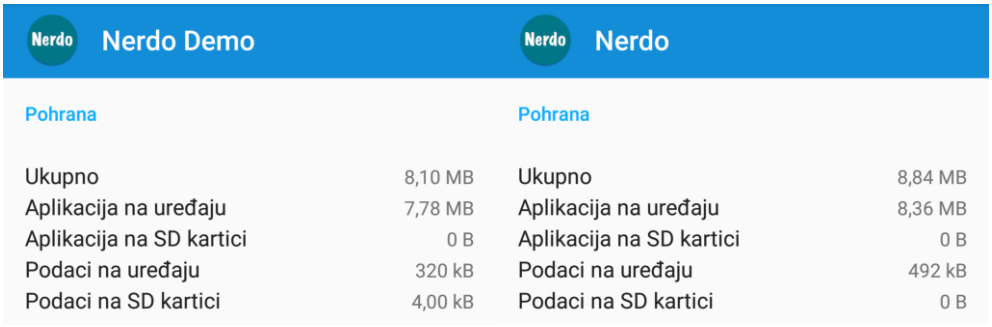


Slika 3.12 Prikaz usporebe veličine projekta na disku

Zbog velike razlike u veličini projekta na disku, potrebno je provjeriti odnos veličina datoteka na uređaju na kojemu je aplikacija instalirana. Demo aplikacija zauzima nešto

manje mjesta sa količinom od 7.78 MB dok Prototip zauzima količinu od 8.36 MB. (Slika 3.13)

Prema think with Google istraživanju jedan od glavnih razloga zbog kojih korisnici deinstaliraju aplikaciju je zauzimanje ili oslobađanje memorije na uređaju.[14]



Slika 3.13 Prikaz usporebe veličine instalirane aplikacije na uređaju

4. Tehnologije i alati

Ovaj dio rada prikazuje glavne tehnologije koje su korištene u Nerdo projektu. Opisuje koji su tipovi autentifikacije korisnika za pristup sadržaju aplikacije i siguran način validacije koji omogućava pristup osjetljivim podatcima. Nadalje se opisuje što je računalni oblak za spremanje i upravljanje podatcima u realnom vremenu, kako se podatci spremaju, što su funkcije i kako se koriste za izmjenu poruka te svrha i rad s lokalnim podatcima.

4.1. Sigurnost i autentifikacija korisnika

Kako bi korisnik pristupio sadržaju aplikacije mora dokazati da je on zaista osoba za koju se predstavlja. Iako Firestore sadrži sustav za autentifikaciju korisnika, unutar Nerdo projekta za klasični login koristi se prilagođeni način za prijavu korisnika. Za prijavu putem društvenih mreža koristi se Facebook SDK, a za Google prijavu koristi se Google Sign-In.

Klasična prijava u aplikaciju sastoji se od unosa adrese elektroničke pošte i lozinke korisnika. Ovisno o dužini i kompleksnosti podataka prijava može potrajati i dovesti do pogreški prilikom unosa. Ukoliko korisnik nema kreiran profil potrebno ga je registrirati. Registracija korisnika najčešće zahtjeva nekoliko polja više od same prijave u aplikaciju.

Implementacija Facebook i Google prijave može znatno pojednostaviti i ubrzati proces registracije i prijave korisnika u aplikaciju. Korisnik se u svega dva klika može registrirati ili prijaviti u aplikaciju i nastaviti koristiti sav sadržaj. Također pomoću takvih sustava mogu se zatražiti dopuštenja i dobiti validne informacije o korisnicima bez traženja unosa podataka.[15,16]

Ukoliko se koristi klasični login, a ne postoji web podrška za aplikaciju, potrebno je razviti sustav za resetiranje lozinke. Za resetiranje lozinke potrebno je unijeti adresu elektroničke pošte na koju se šalje pošta sa tokenom koji vrijedi 10 minuta. Za slanje elektroničke pošte brine se SendGrid aplikacijsko programsko sučelje (engl. *application programming interface*, skraćeno API) uz pomoć Cloud Firestore funkcija. Nakon generiranja tokena korisnik može unijeti zaprimljeni token u aplikaciju. Ukoliko je token aktivan aplikacija se preusmjerava na ekran za postavljanje nove lozinke.

U današnje vrijeme privatnost i sigurnost podataka za korisnike predstavlja iznimnu važnost. Stoga se pri otvaranju postavki aplikacije, profila ili izvršavanjem specifične akcije može postaviti validacija kako nitko ne bi mogao pristupiti osjetljivim podatcima osim korisnika. Kako se Android operativni sustav (engl. *operation system*, skraćeno OS) i Android uređaji konstantno razvijaju tako nastaje sve više mogućnosti za autentifikaciju i validaciju korisnika. Validacija se može napraviti prikazom sistemskog otključavanja zaslona putem pina, uzorka, otiska prsta ili prepoznavanjem lica.[9] Podrška za autentifikaciju korisnika pomoću otiska prsta omogućena je u Android 6 verziji OS-a implementacijom `FingerprintManager` klase.[17] Od Android 9 OS Google je predstavio korištenje `BiometricPrompt` klase za autentifikaciju te je time implementaciju `FingerprintManager` iz verzije 6 postavio kao zastarjelu. `BiometricPrompt` klasa u odnosu na `FingerprintManager` znatno olakšava implementaciju brinući se o prikazu dijaloga koji vodi korisnika kroz proces autentifikacije, izvršavajući autentifikaciju i vračajući rezultat aplikaciji. Ukoliko dođe do pogreške kod autentifikacije, `BiometricPrompt` klasa blokira ponovnu autentifikaciju na određeno vrijeme.[9]

4.2. Računalni oblak za spremanje i upravljanje podatcima

Projekt ovisi o dijeljenju i prikazu korisničkog sadržaja zbog čega je potrebno postaviti Cloud Firestore bazu podataka za spremanje i upravljanje podatcima. Cloud Firestore je računalni oblak razvijen od strane Firebasea i Google Cloud Platforme koji nudi fleksibilnu i skalabilnu bazu podataka za web, mobilni i serverski razvoj. Kao i Realtime Database, Cloud Firestore nudi mogućnost sinkronizacije podataka u realnom vremenu i mogućnost offline podrške za mobilne i web platforme što znači da će podatci biti pohranjeni neovisno o brzini mreže korisnika i povezanosti na internet.[18]

Cloud Firestore sprema podatke kao kolekcije dokumenata te se svi dokumenti mogu dodatno hijerarhijski proširiti na podkolekcije dokumenata. Unutar projekta, Cloud Firestore je implementiran kao zasebni modul aplikacije. Podatci koji se dohvaćaju iz baze podataka prikazuju se kao mape, stoga modul za komunikaciju s bazom ne ovisi niti o jednom drugom modulu. Svaki modul koji koristi komunikaciju s bazom mora

implementirati FirestoreLibrary modul. Pri prijenosu podataka između modula radi se parsiranje mape u objekt.

Neki od nedostataka korištenja Cloud Firestore baze podataka su:

- Unos testnih podataka - za svaki podatak potrebno je odabrati ključni naziv, tip podataka koji se unosi i unijeti vrijednost, vodeći računa o hijerarhijskom prikazu podataka.
- Filtriranje podataka - trenutno se liste mogu filtrirati samo po jednom parametru što znatno otežava kompleksniji prikaz podataka.[19]

4.3. Komunikacija porukama

Funkcionalnost komunikacije porukama neizostavni je dio društvenih mreža. Uz objavljivanje i dijeljenje sadržaja potrebno je omogućiti korisnicima međusobnu komunikaciju stoga Cloud Firestore nudi podršku za slanje poruka korištenjem Firebase Cloud Messaging (FCM) funkcionalosti.

Ključne sposobnosti FCM-a su:

- Slanje notifikacijskih poruka koje mogu biti prikazane korisniku ili slanje podataka o poruci koje se procesiraju i prikazuju unutar koda aplikacije
- Slanje poruka prema jednom uređaju, grupi uređaja ili uređajima koji su pretplaćeni na temu
- Slanje poruka s klijentskih aplikacija kroz FCM pouzdani komunikacijski kanal

FCM sadrži dvije glavne komponente za slanje i primanje poruka. Pouzdanu okolinu kao što su Cloud Funkcije ili server na kojemu se kreiraju, targetiraju i šalju poruke i klijentsku aplikaciju za zaprimanje poruka.[20]

Unutar Nerdo aplikacije komunikacija porukama napravljena je korištenjem Cloud Funkcija. Funkcije se pišu u Node.js okolini i pohranjuju na server korištenjem Firebase komandnog sučelja (engl. *command line interface*, skraćeno CLI) kako bi se mogle uspješno koristiti. Cloud Funkcije omogućavaju izvršavanje *backend* koda izazvano određenim akcijama neovisno o radu aplikacije. [21]

Korištenjem Cloud Funkcija uz Cloud Firestore bazu podataka, moguće je pratiti promjene u bazi podataka i izvršiti potrebnu akciju. Akcije mogu biti definirane kao: `onCreate` kada se podatak prvi put zapisuje u bazu podataka, `onUpdate` kada podatak postoji, ali

mu se vrijednost promjeni, `onDelete` kada se podatak izbriše i `onWrite` koji se odnosi na sva tri prethodna stanja.

Kada korisnik unutar aplikacije pošalje poruku, poruka se sprema u bazu podataka i pokreće izvršavanje zadane funkcije. Navedena funkcija detektira unos novog ili izmjenjenog dokumenta unutar `Chats` kolekcije koja sadrži informacije o sudionicima razgovora i podkolekcije `Messages` koja sadrži informacije o poslanoj poruci.

```
exports.sendNotification =
functions.firestore.document("Chats/{chat_id}/Messages/{messa
ge_id}").onWrite((change,context) => {
      const chat_id = context.params.chat_id;
      const message_id = context.params.message_id;
});
```

Kôd 4.1 Primjer koda u funkcijama za osluškivanje promjena u bazi podataka i postavljanja varijabli

Dohvaćanje informacija o sudionicima razgovora omogućuje dohvat korisničkog imena i slike profila pošiljatelja te token za uređaj primatelja na koji će se poslati poruka.

```
return
admin.firestore().collection("Chats").doc(chat_id).collection
("Messages").doc(message_id).get().then(queryResult => {
      const from_id = queryResult.data().from;
      const to_id = queryResult.data().to;
      const message = queryResult.data().message;
      const from_data =
admin.firestore().collection("Users").doc(from_id).get));
      const to_data =
admin.firestore().collection("Users").doc(to_id).get();
      return Promise.all([from_data, to_data]).then(result =>
{
            const from_name = result[0].data().username;
            const to_name = result[1].data().username;
            const image_path = result[1].data().imagePath;
            const token_id = result[1].data().token;
      });
});
```

Kôd 4.2 Primjer koda u funkcijama za dohvat informacija o sudionicima razgovora

Dohvaćene informacije o korisnicima i informacije o poslanoj poruci potrebno je grupirati u podatke za slanje prema klijentskoj aplikaciji.

```
const payload = {
     notification: {
          title : "You have new message from " + from_name,
          body : message,
          icon : "default",
     },
     data: {
          "type" : "NOTIFICATION_MESSAGE",
          "imagePath" : image_path
     }
};
```

Kôd 4.3 Primjer koda u funkcijama za postavljanje payloada koji se šalje klijentskoj aplikaciji

Nakon što se uspješno grupiraju podatci za slanje potrebno je poslati poruku na uređaj koristeći token primatelja.

```
return admin.messaging().sendToDevice(token_id,
payload).then(notificationResult => {
      console.log("Notification sent...");
});
```

Kôd 4.4 Primjer koda u funkcijama za slanje poruke na jedan uređaj sa specifičnim tokenom

Na klijentskoj strani potrebno je kreirati servis koji nasljeđuje `FirebaseMessagingService` i nasljediti `onMessageReceived(RemoteMessage remoteMessage)` metodu kako bi se moglo upravljati zaprimljenim podatcima i napraviti specifična akcija.

Prema službenoj dokumentaciji Cloud Funkcije su u beta verziji te su moguće poteškoće pri izvršavanju funkcija. Trenutne poznate poteškoće su potrebno vrijeme do 10 sekundi da se izvrši funkcija kada dođe do promjene u bazi, pravilno organiziranje redosljeda izvršavanja funkcija te osiguravanje jednog izvršavanja funkcije.[22]

4.4. Lokalni rad s podatcima

U određenim situacijama rad s lokalnim podatcima može biti od velike koristi za aplikaciju. Neke od situacija mogu biti kada korisnik pokušava pristupiti sadržaju aplikacije bez mobilne mreže, kada dođe do pogreške s komunikacijom između aplikacije i

baze podataka i kada se spremanjem osnovnih informacija o korisniku želi rasteretiti bazu podataka i smanjit broj poziva prema bazi.

Za spremanje podataka u lokalnu bazu može se koristiti SQLite i Room. SQLite je lagan i otvoreni sustav upravljanja relacijskim bazama podataka koji omogućava spremanje i upravljanje kompleksnim podatcima. Kreirana baza podataka je privatna, ugrađena u aplikaciju i može joj se pristupiti samo iz aplikacije koja ju je kreirala.[23] Prema Googleovim smjernicama SQLite za vrijeme kompajliranja nema provjeru validnosti SQL upita stoga do greške dolazi tek kada se izvršava određeni upit korištenjem aplikacije. Kod promjene tipova ili proširivanjem objekata potrebno ih je ručno mjenjati što zahtjeva puno vremena i moguće su pogreške stoga je potrebno jako puno koda za pretvaranje između SQL upita i tipova objekata.[24] Google preporuća korištenje Room koji pruža apstrakcijski sloj preko SQLitea kako bi se omogućio pristup bazi.

Room se sastoji od 3 glavne komponente:

- Baze podataka (engl. *database*) - služi kao glavna pristupna točka za konekciju s podatcima
- Entiteta (engl. *entity*) - klasa koja predstavlja i opisuje tablicu
- Objekta za pristup podatcima (engl. *data access object*, skraćeno DAO) - sadrži metode za pristup bazi podataka

Unutar Nerdo projekta u lokalnu bazu podataka spremaju se posljednji podatci o korisniku, zadnje dohvaćeni postovi i zaprimljene notifikacije. Implementacija Room baze podataka vrlo je jednostavna te se može napraviti u svega nekoliko koraka.

Na objekt za kojeg želimo kreirati tablicu potrebno je postaviti anotaciju *@Entity* s nazivom tablice te na parametre objekta postaviti anotaciju *@ColumnInfo* kako bi se točno odredio naziv stupca. Za parametre koji se žele ignorirati može se postaviti anotacija *@Ignore*. Složeniji parametri poput lista zahtjevaju pretvarače kako bi se mogli uspješno spremiti u bazu podataka.

Potrebno je kreirati *@Dao* objekt koji predstavlja sučelje te postaviti metode za rad s tablicom. Pri postavljanju metoda Room nudi opciju iskorištavanja anotacija za dodavanje, brisanje, dohvat i izmjenu podataka ili se mogu pisat zasebni upiti za upravljanje podatcima, primjerice ako postoji podatak s određenom vrijednosću koji se želi izbrisati.

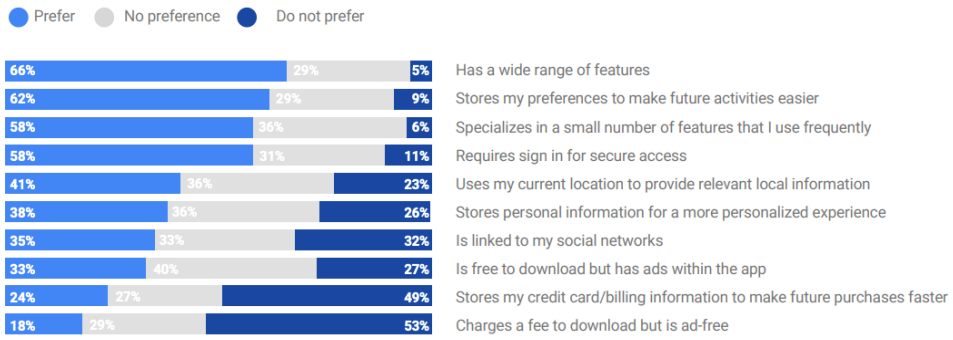
Na klasi koja predstavlja bazu podataka potrebno je postaviti anotaciju *@Database* sa entitetima koji se koriste i verziju baze podataka. Nadalje, potrebno je instancirati Dao objekt i napraviti instancu klase kako bi se moglo pristupiti bazi.[25]

5. Mogućnosti poboljšanja aplikacije

Ovaj dio rada opisuje kako implementacija novih funkcionalnosti može utjecati na aplikaciju i korisnike. Predložena su poboljšanja aplikacije u skladu sa sadržajem aplikacije. U nastavku se opisuje za koje se sve platforme mogu raditi aplikacije i koje se funkcionalnosti iz projekta mogu iskoristiti za razvoj na drugim platformama.

5.1. Dodatne funkcionalnosti

U današnje vrijeme vrlo je teško zadržati aktivne korisnike unutar aplikacije, stoga je potrebno postepeno proširivati aplikaciju pružajući korisnicima potrebu za korištenjem sadržaja i vodeći računa o memoriji koju aplikacija zauzima. Pri planiranju funkcionalnosti potrebno je da implementirana funkcionalnost bude u skladu sa sadržajem aplikacije, ciljanom skupinom korisnika i da navedena funkcionalnost ispunjava potrebe korisnika. Također prema think with Google istraživanju, za korisnike nema bitnije značajke od jednostavnosti korištenja i navigacije sadržaja.



Slika 5.1 Prikaz utjecaja širokog raspona funkcionalnosti i personaliziranih opcija na korisnike aplikacije

Kako bi nova funkcionalnost bila u skladu s istraživanjem i kako bi se postiglo što bolje korisničko iskustvo, potrebno je da korisnik u što manje koraka odradi željenu akciju. Projekt Nerdo sadrži polje za pregled oglasa za posao ili praksu te bi funkcionalnost brze prijave na natječaj prikupila korisnikove informacije, kreirala dokument u kojem se nalazi životopis korisnika, ovisno o ispunjenosti korisničkog profila i prosljedila ga na oglas za posao.

Cilj i vizija projekta je omogućiti korisnicima pristup informacijama koje im pomažu u osobnom i profesionalnom razvoju. Uz dijeljeni sadržaj drugih korisnika, funkcionalnost za online učenje omogućila bi korisnicima pristup video i audio sadržaju i drugim oblicima e-learning materijala kojima bi korisnici mogli razvijati svoje znanje.

Kao dodatak na funkcionalnost za online učenje, korisnicima bi od velike koristi bila i funkcionalnost za provjeru znanja i certificiranja koji bi se mogli provoditi kao kratki kvizovi.[13]

5.2. Proširenja na druge Android platforme

Uz mobilnu platformu Google nudi veliki opseg platformi koje podržavaju Android aplikacije kao što su: Wear OS, Android TV, Android for Cars, Android Things i Chrome OS. Svaka platforma ima svoju specifičnu primjenu te se svaka aplikacija ne može sadržajem prilagoditi svim platformama.

Wear OS je pametni sat koji olakšava život korisnicima nudeći kratke akcije kojima mogu izvršiti planirane radnje. Neki od primjera su: praćenje sportskih rezultata, bezkontaktno plaćanje, informacije o pozivima, porukama i elektoničkoj pošti. Aplikacije na pametnim satovima su u najčešćem slučaju sinkronizirane s mobilnim aplikacijama.

Android TV nudi prikaz multimedijalnog sadržaja korisniku na velikom platnu.

Android for Cars nudi mogućnost korisnicima kontroliranje sadržaja glasovnim kontrolama.

Android Things daje podršku za razvoj aplikacija na popularnim hardwerskim platformama poput Raspberry Pi 3.

Chrome OS daje podršku za distribuiranje aplikacija na uređajima koji podržavaju Google Chrome operativni sustav, poput Chromebooka.

U projektu Nerdo, na Wear OS platformi po implementiranim funkcionalnostima najviše se mogu prilagoditi notifikacije. Notifikacije bi poslužile za: prikaz primljenih poruka, mogućnost brzog odgovora na poruke, odgovor na zahtjev za povezivanje s drugim korisnicima, prikaz novih poslovnih ponuda i prilika te prikaz komentara na dijeljeni sadržaj korisnika.

Implementacijom funkcionalnosti za pregled i slušanje E-learning sadržaja iz prethodne cjeline, aplikacije se može prilagoditi i Android TV platformi. E-learning sadržaj bi se mogao gledati kao video ili slušati kao audio sadržaj uz prezentaciju koda.[26]

Zaključak

Istraživanje utjecaja modula na vrijeme i kompleksnost razvoja dovodi do zanimljivih rezultata. Iako se u većini procesa radi o sekundama, na temelju rezultata može se zaključiti da je aplikacija s jednim modulom i dalje efikasnija u odnosu na modulariziranu aplikaciju. S druge strane, kao što je spomenuto, modulariziranje znatno utječe na Gradle datoteke i Gradle postavke za projekt. Kombinacija modula i Gradlea s novim verzijama konstanto se poboljšava te bi već u sljedećim verzijama odnos performansi mogao biti znatno bolji.

U pogledu kompleksnosti razvoja, modulariziranje aplikacije ima veći utjecaj na projekte s više komponenti što omogućava lakšu organizaciju i strukturu projekta. Svi moduli u kreiranom projektu koriste MVP arhitekturu. Rastavljajući logiku i prikaz sučelja na tri sloja omogućavaju lakšu implementaciju dohvata i prikaza podataka korisniku. Projekt se koristi bibliotekama koje olakšavaju razvoj aplikacija te postavljaju standard za izradu kompleksnijih i skalabilnijih aplikacija. Primjenom modula olakšano je planiranje novih funkcionalnosti i proširenje na nove platforme, a postojeće komponente moguće je prilagoditi platformi i iskoristit već kreiranu logiku.

Popis kratica

SDK	*Software Development Kit*	skup alata za razvoj programa
FCM	*Firestore Cloud Messaging*	platforma za komunikaciju porukama
GUI	*Graphical User Interface*	grafičko korisničko sučelje
API	*Application Programming Interface*	aplikacijsko programsko sučelje
OS	*Operation Systems*	operativni sustavi
CLI	*Command Line Interface*	komandno sučelje
DAO	*Data Access Object*	objekt za pristup podatcima
JAR	*Java ARchive*	java datoteka
AS	*Android Studio*	softwer za razvoj android aplikacija
CPU	*Central Processing Unit*	procesor
APK	*Android Package*	format datoteke za android aplikacije

Popis slika

Slika 2.1 Prikaz app modula kod kreiranja novog projekta 2

Slika 2.2 Prikaz sheme Design Pattern klasa 3

Slika 2.3 Prikaz sheme Model View Presenter obrasca 4

Slika 2.4 Primjer implementacije MVP arhitekture u komponenti za resetiranje lozinke ... 5

Slika 3.1 Prikaz odnosa mjerenja CPU 7

Slika 3.2 Prikaz odnosa mjerenja mreže 8

Slika 3.3 Prikaz odnosa mjerenja potrošnje memorije 8

Slika 3.4 Prikaz izvršavanja vremena procesa čišćenja i ponovnog kreiranja po projektu 10

Slika 3.5 Prikaz izvršavanja vremena procesa čišćenja i ponovnog kreiranja u odnosu na broj procesa 10

Slika 3.6 Prikaz izvršavanja vremena procesa kreiranja debug APK datoteke pri izmjeni jedne komponente 11

Slika 3.7 Prikaz izvršavanja vremena procesa kreiranja debug APK datoteke pri izmjeni u više komponenti 11

Slika 3.8 Prikaz izvršavanja vremena procesa kreiranja release APK datoteke pri izmjeni jedne komponente 12

Slika 3.9 Prikaz izvršavanja vremena procesa kreiranja release APK datoteke pri izmjeni jedne komponente 12

Slika 3.10 Prikaz izvršavanja vremena procesa kompajliranja aplikacije pri izmjeni jedne komponente 13

Slika 3.11 Prikaz izvršavanja vremena procesa kompajliranja aplikacije pri izmjeni u više komponenti 13

Slika 3.12 Prikaz usporebe veličine projekta na disku 15

Slika 3.13 Prikaz usporebe veličine instalirane aplikacije na uređaju 16

Slika 5.1 Prikaz utjecaja širokog raspona funkcionalnosti i personaliziranih opcija na korisnike aplikacije 24

Popis kôdova

Kôd 4.1 Primjer koda u funkcijama za osluškivanje promjena u bazi podataka i postavljanja varijabli..... 20

Kôd 4.2 Primjer koda u funkcijama za dohvat informacija o sudionicima razgovora 20

Kôd 4.3 Primjer koda u funkcijama za postavljanje payloada koji se šalje klijentskoj aplikaciji 21

Kôd 4.4 Primjer koda u funkcijama za slanje poruke na jedan uređaj sa specifičnim tokenom 21

Literatura

[1] Android Developers, User Guide, https://developer.android.com/studio/projects, Kolovoz. 2019.

[2] Android Developers, Whats New, https://developer.android.com/studio/releases#3-2-0, Kolovoz. 2019

[3] Wikipedia, Software Design Pattern, https://en.wikipedia.org/wiki/Software_design_pattern#Classification_and_list, Kolovoz. 2019

[4] Freeman, E., Robson, E., Sierra, K., Bates, B. (2004) Head First Design Patterns: A Brain-Friendly Guide. Sebastopol: O'Reilly Media, Inc.

[5] Wikipedia, Architectural patter, https://en.wikipedia.org/wiki/Architectural_pattern, Kolovoz. 2019

[6] Potel, M. VP & CTO Taligent, Inc. MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java, https://www.wildcrest.com/Potel/Portfolio/mvp.pdf, Kolovoz. 2019

[7] Android Notes For Professionals, https://goalkicker.com/AndroidBook/AndroidNotesForProfessionals.pdf, Kolovoz. 2019

[8] Mainkar, P. Expert Android Programming: Master skills to build enterprise grade Android applications, Birmingham-Mumbai, 2017, Packt Publishing, https://books.google.hr/books?id=7plGDwAAQBAJ&pg=PA91&hl=hr&source=gbs_toc_r&cad=4#v=onepage&q&f=false, Kolovoz. 2019

[9] Smith, N. (2019) Android Studio 3.4 Development Essentials - Java Edition. Payload Media, Inc.

[10] Android Developers, User Guide, https://developer.android.com/studio/profile/cpu-profiler, Kolovoz. 2019

[11] Android Developers, User Guide, https://developer.android.com/studio/profile/network-profiler, Kolovoz. 2019

[12] Android Developers, User Guide, https://developer.android.com/studio/profile/memory-profiler, Kolovoz. 2019

[13] Think with Google, How People Discover, Use And Stay Engaged With Apps, https://www.thinkwithgoogle.com/_qs/documents/331/how-users-discover-use-apps-google-research.pdf, Kolovoz. 2019

[14] Gradle Docs 5.6.1, Command Line Interface, https://docs.gradle.org/current/userguide/command_line_interface.html#cleaning_outputs, Kolovoz. 2019

[15] Facebook for Developers, Docs, https://developers.facebook.com/docs/android/graph, Kolovoz. 2019

[16] Google Sign-In For Android, Guide, https://developers.google.com/identity/sign-in/android/people, Kolovoz. 2019

[17] Android Developers, Platform, https://developer.android.com/about/versions/marshmallow/android-6.0, Kolovoz.2019

[18] Firebase, Documentation Guide, https://firebase.google.com/docs/firestore/rtdb-vs-firestore, Kolovoz. 2019

[19] Firebase, Documentation Guide, https://firebase.google.com/docs/firestore/, Kolovoz. 2019

[20] Firebase, Documentation Guide, https://firebase.google.com/docs/cloud-messaging, Kolovoz. 2019

[21] Firebase, Documentation Guide, https://firebase.google.com/docs/functions/get-started, Kolovoz. 2019

[22] Firebase, Documentation Guide, https://firebase.google.com/docs/functions/firestore-events, Kolovoz. 2019

[23] Meier, R. (2012) Professional Android 4 Application Development. Indianapolis: John Wiley & Sons, Inc.

[24] Android Developers, User Guide, https://developer.android.com/training/data-storage/sqlite, Kolovoz. 2019

[25] Android Developers, User Guide, https://developer.android.com/training/data-storage/room, Kolovoz. 2019

[26] Android Developers, Preview, https://developer.android.com/about, Kolovoz. 2019

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, datum.