Android aplikacija za praćenje prilagodbe djece u vrtićima

Orlić, Rastimir

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni diplomski studij

ANDROID APLIKACIJA ZA PRAĆENJE PRILAGODBE DJECE U VRTIĆIMA

Diplomski rad

Rastimir Orlić

Osijek, 2017.

SADRŽAJ

1. UVOD

1.1. Opis aplikacije i korištene tehnologije

Cilj ovoga rada je omogućiti predškolskim psiholozima bolje praćenje napretka djece u vrtićima te njihovu integraciju u vrtićke skupine. Kako bi se to ostvarilo osmišljena je mobilna aplikacija koja će psiholozima pružiti mogućnost bilježenja njihovih opservacija tijekom boravka u skupinama. Aplikacija treba biti povezana na udaljenu bazu podataka s kojom je neprestano u interakciji. Korisnik (psiholog) dobiva email i lozinku kojom se prijavljuje u aplikaciju te dohvaća sve podatke iz baze. Nakon što su podaci dohvaćeni, korisnik može pretraživati podatke o djeci, poremećajima, nalaze i vrtiće kako bi znao koja djeca se nalaze u određenim skupinama i vrtićima te da li je kod pojedinog djeteta uočen neki poremećaj ili poteškoća u razvoju i prilagodbi vrtićkom okruženju. Također psihologu je omogućeno unošenje nalaza za pojedino dijete te slanje tog nalaza na udaljenu bazu. Za te potrebe dizajniran je API u programskom jeziku PHP korištenjem Laravel *framework-a.* Mobilna aplikacija rađena je na Android platformi korištenjem Java programskog jezika, najnovijih biblioteka unutar Android SDK (*Retrofit, GSON, Dagger2, RxJava2, Butterknife, Stetho*), a izrađena je po MVP (*Model - View - Presenter*) arhitekturi.

Rad je strukturiran tako da u prvom, uvodnom poglavlju opiše ukratko aplikaciju te problematiku struke i potrebu za izgradnjom aplikacije. Drugo poglavlje opisuje razvojna okruženja, programske jezike, osnovne principe arhitekture aplikacije, određene komponente i biblioteke koje su korištene u izradi Android i API aplikacije. Treće poglavlje opisuje dizajn aplikacija, četvrto opisuje *backend* - skladištenje podataka na mobilnom uređaju i bazi te komunikaciju između API i mobilne aplikacije. Peto poglavlje sadrži upute za uporabu aplikacije, a šesto poglavlje je zaključak.

1.2. Potreba za aplikacijom u struci predškolski psiholog

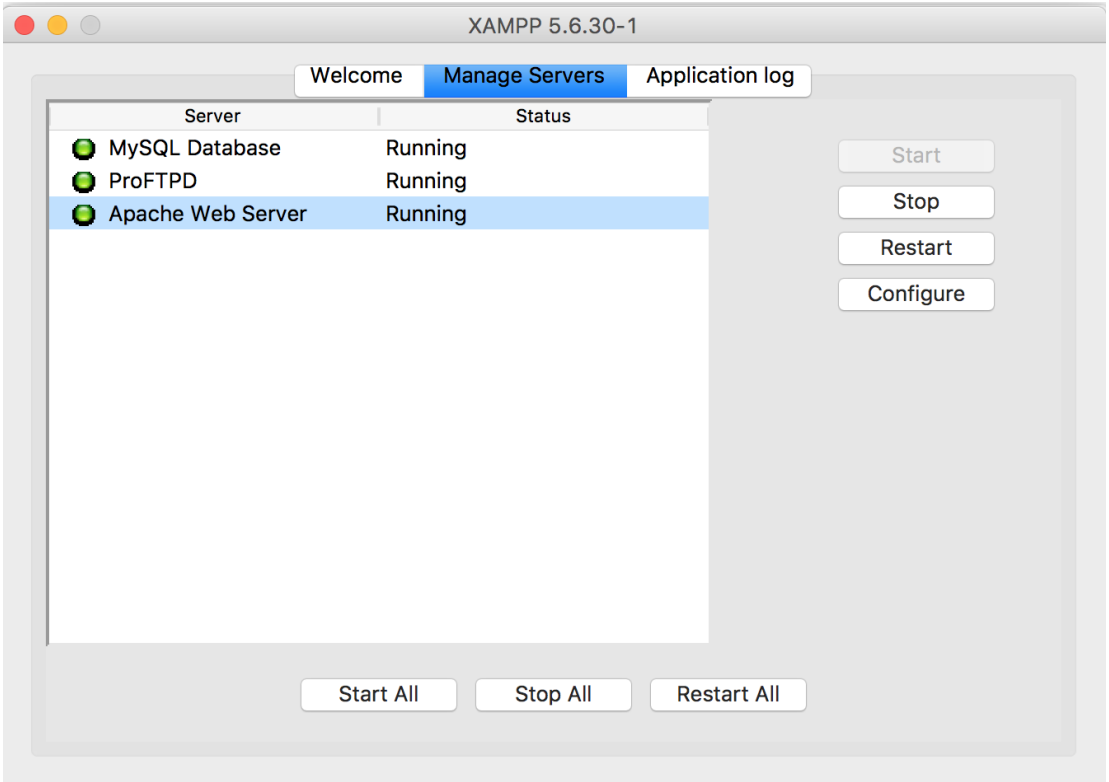
Rad predškolskih psihologa je najviše usmjeren na interakciju s djecom, roditeljima i odgojiteljima. Prema [1] posebno je bitna interakcija s djecom jer iz nje proizlazi interakcija s roditeljima i odgojiteljima. Kako bi ta interakcija postala kvalitetnijom potrebno ju je bilježiti i ostavljati pisani trag o njoj. Upravo za to služi aplikacija koja je predmet ovog rada. Ona omogućava psiholozima ostavljanje pisanog traga o svakoj interakciji s djecom kako bi se kasnije mogli bolje podsjetiti, bilježiti napredak ili nazadovanje i na temelju tih podataka donositi odluke o interakciji s roditeljima i odgojiteljima – na što ih usmjeriti, što pohvaliti, što prigovoriti itd. U razgovoru s psiholozima viđena je potreba za aplikacijom koja neće samo omogućavati pisanje nalaza za pojedino dijete, nego će pružati i mogućnost organizacije djece po skupinama i vrtićima zbog konkretnog slučaja u Osijeku gdje je jedan psiholog zadužen za više od 20 različitih vrtića te je teško pratiti pripadnost pojedinog djeteta određenom vrtiću, a kamo li skupini. Zato ova aplikacija uz unošenje novih podataka o djeci i njihovih nalaza omogućuje pregled vrtića, skupina i odgojiteljica zaduženih za pojedine skupine. Takav pristup radu izravno doprinosi produktivnosti i kvaliteti rada psihologa. [2] Također sve opservacije psihologa koriste i prilikom pripremanja roditeljskih sastanaka, uočavanja nekih globalnih problema u vrtićima ili skupinama koji se onda mogu na bolji način približiti roditeljima i odgojiteljima te ih na taj način bolje osvijestiti o postojanju tih problema te kako se s njima nositi (npr. jedno dijete ima problem s bijesom koje utječe na svu ostalu djecu i odgojiteljice te ih psiholog tada može informirati o drugačijim načinima nošenja s djetetovim bijesom).

2. OSNOVNE KOMPONENTE APLIKACIJE

2.1. Korišteni programski jezici i razvojna okruženja

Kod izrade obje aplikacije korišten je Git [3] – alat za verzioniranje koda. Kod je pohranjivan na Bitbucket mrežno - orijentiranom *hosting* servisu, a svo verzioniranje obavljeno je preko Git Bash sučelja. Kod je organiziran na dvije Git grane – *master* i *development*. Dok je aplikacija razvijana, sve nove izmjene spremane su na *development* granu, a nakon završetka razvoja, *development* grana je sjedinjena u *master* granu. Moglo se ići u više grananja pa recimo u mobilnoj aplikaciji imati i grana koje bi bile zadužene samo za promjene na korisničkom sučelju ili samo za povezivanje s API aplikacijom i komunikaciju s lokalnom bazom. Kako je na ovom projektu radila samo jedna osoba, radi jednostavnosti korištene su samo dvije prethodno navedene grane.

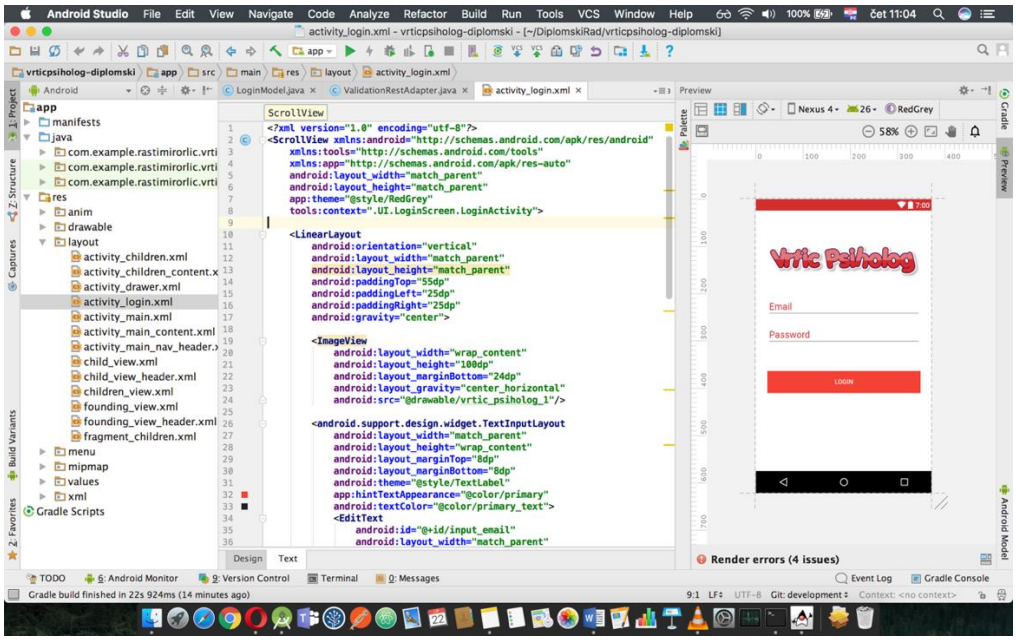
Za poslužitelj korišten je XAMPP [4] – besplatno poslužiteljsko rješenje otvorenog koda namijenjeno za rad na različitim platformama. Sastoji se od Apache HTTP poslužitelja, MariaDB (MySQL) baze podataka i tumača za skripte napisane u PHP i Perl programskim jezicima. Prikaz svih pokrenutih komponenti je na slici 2.1. Verzija baze podataka je: 10.1.21 – MariaDB, a mrežnog poslužitelja: Apache/2.45(Unix) OpenSSL/1.0.2j.



Slika 2.1. *XAMPP poslužiteljsko rješenje*

Izrada API aplikacije ostvarena je korištenjem programskog jezik PHP (engl. *Hypertext Preprocessor*) koji spada u porodicu skriptnih, objektno – orijentiranih jezika široke primjene. U ovome projektu korištena je 5.6.3. verzija PHP jezika. Kako PHP jezik ima široku primjenu, korištenje različitih razvojnih okvira (engl. *Framework*) bolje definira njegovu primjenu i olakšava je. Zato je korišten Laravel razvojni okvir verzije 5.4. koji je svojim ugrađenim funkcionalnostima i MVC (engl. *Model - View - Controller*) arhitekturom pojednostavio proces razvoja sučelja baze podataka. Laravel ima svoj vlastiti ORM (engl. *Object Relational Mapping*) mehanizam koji znatno ubrzava i olakšava pisanje, čitanje i pretraživanje nad skupom podataka u bazi. Za pisanje koda API aplikacije korišten je Atom – besplatno integrirano okruženje za razvoj aplikacija. Svojim PHP i Git ekstenzijama doprinosi lakom pisanja koda.

Mobilna aplikacija rađena je za Android platformu, točnije minimalni SDK (engl. *Software development kit*) na mobilnom uređaju mora biti Android 4.0.3. (Ice Cream Sandwich) – API 15. Za mobilne uređaje ispod ove razine, aplikacija neće dobro raditi. Odabirući API 15 za najnižu razinu, pogođeno je 97,4% svih Android uređaja što je više, nego prihvatljiv postotak. Za izradu mobilne aplikacije korišten je programski jezik Java – opće namjenski, objektno – orijentirani jezik, već jako dobro poznat velikom broju programera te XML (engl. *Extensible Markup Language*) – jezik za označavanje podataka korišten u dizajnu Android aplikacija te definiranju raznih drugih resursa (teksta, slika, dimenzija itd.). Za pisanje koda korišten je Android Studio – besplatno integrirano okruženje za razvoj Android aplikacija (verzije 2.3.3.). Prikaz korisničkog sučelja Android Studio-a je na slici 2.2.



Slika 2.2 *Android Studio razvojno okruženje*

Od korištenih stavki u izradi ovog rada valjalo bi još spomenuti i JSON (engl. *Javascript Object Notation*) – format u kojem se prenose podaci. Vrlo je jednostavan i čitljiv te ga je lako generirati s većinom korištenih programskih jezika. JSON format je moglo bi se reći industrijski standard kada je u pitanju prijenos podataka, a može sadržavati

- objekte sastavljene od atributa koji u sebi imaju podatke grupirane po principu ključ – vrijednost
- polja (koja se mogu sastojati od više objekata ili više vrijednosti)
- zasebne vrijednosti različitog tipa.

2.2. Programski jezik Java

Prema [5] Java je jednostavan, objektno – orijentiran, prijenosan, neovisan o platformi, siguran, robustan, distribuiran, višenitni programski jezik razvijen od strane James Goslinga davne 1995. Godine [6]. Njegova vjerojatno najbitnija i najspecifičnija značajka je WORA (engl. Write once run anywhere) – sposobnost da kada se jednom prevede (obično) u *bytecode* može biti pokretan na bilo kojem Java VM (engl. *Virtual Machine*) neovisno o arhitekturi računala. To omogućava veliku rasprostranjenost Jave što na različitim uređajima (osobna računala, bankomati, poslužitelji, *set-top* kutije itd.), što u različitim područjima (web, baze, mobilne aplikacije, ugrađeno (engl. *Embedded*) programiranje itd.). Što se tiče sintakse jezika, Java većinu svoje

sintakse vuče iz C/C++ jezika, ali ima puno manje značajki programiranja niske razine od njih. Trenutna verzija Java jezika je Java 8.

2.3. Programski jezik PHP

Prema [7] PHP je poslužiteljski, skriptni, objektno – orijentirani programski jezik opće namijene s naglaskom na razvoj web aplikacija. Napisao ga je Rasmus Lerdof 1994. godine, a značenje mu je tada bilo "Personal Home Page". PHP kod može se ugraditi u HTML ili može biti korišten u kombinacijama s razno raznim web sustavima predložaka, web CMS-ovima (engl. Content Menagement Systems) i razvojnim okvirima (Laravel, Symphony, CodeIgniter, itd.). PHP kod je obično procesiran s PHP interpreterom koji je implementiran kao modul u poslužitelju ili kao CGI (engl. *Common Gateway Interface*). Poslužiteljski software kombinira rezultate interpretiranog PHP koda koji može biti bilo koji tip podataka (u PHP programskom jeziku ne postoje tipovi podataka) u generiranu web stranicu. PHP kod također može biti pokretan kroz CLI (engl. *Command-Line Interface*), a može biti implementiran i kao samostalna grafička aplikacija.

2.4. Povijest Androida

Prema [8] Android je rođen 05. Studenog 2007. Od tada izrastao je u jedan od najdominantnijih operacijskih sustava u svijetu, no to se nije dogodilo preko noći. Android je prošao kroz brjegove i doline kako bi došao na poziciju na kojoj se nalazi. Njegov najveći konkurent iOS mu putovanje nije olakšao svojim konstantnim inovacijama, no s druge strane, velike cijene Apple proizvoda olakšale su Androidu lakše probijanje na globalno tržište s obzirom da je on od samog početka bio *open source*, a cijene uređaja znatno niže od Apple-ovih. Ipak, različitost uređaja, zaslona i Android verzija koju određeni uređaji podržavaju znatno su zakomplicirale i otežale proces razvoja aplikacija, nego što je to slučaj kod iOS-a koji ima uniformirane uređaje i verzije. U tablici 2.1., prema [9], prikazane su verzije Androida sa najbitnijim novim svojstvima. Trenutna najnovija verzija Android operacijskog sustava je Android Oreo 8.0 (API 26). Kako za razvoj specifičnih aplikacija često postoji i specifično razvojno okruženje (IDE) tako je i u slučaju Androida. Do kraja 2014. godine glavni IDE za razvoj Android aplikacija bio je Eclipse, no od onda je u potpunosti nadvladao Android Studio koji je od tada uvelike napredovao i postao vrlo stabilno i puno optimiziranije razvojno okruženje.

2.5. Razvoj aplikacija za Android platformu

Razvoj aplikacija za Android platformu [10] moguć je korištenjem Java programskog jezika kao što je to slučaj u ovom projektu, no Android je ove godine integrirao novi jezik koji je postao materinji poput Jave – Kotlin. Kotlin je statički napisan programski jezik široke promjene, pokreće se na Java VM, može se prevoditi i u JavaScript, a postao je 100% interoperabilan s Javom i Android SDK-om. Android SDK je skup opsežnih alata za razvoj aplikacija. One uključuju alat za pronalaženje pogrešaka, biblioteke, emulator koji je baziran na QEMU (engl. *Quick Emulator*), dokumentaciju, uzorke koda i tutorijale. Osim Jave i Kotlina, Android omogućava korištenje i jezika bližeg Android jezgri – C++. To je omogućeno kroz Android NDK (engl. *Native Development Kit*). Androidova jezgra je bazirana na jednom od Linux dugoročno podržavanih grana, a od ove godine Android uređaji većinom koriste verzije 3.18 ili 4.4 Linux jezgri.

Kada su u pitanju operacijski sustavi, razvijati za Android moguće je na sva tri vodeća operacijska sustava

1. Linux (bilo koja moderna Linux distribucija)
2. Mac OS X (od verzija 10.5.8 na dalje)
3. Windows (Windows 7 na dalje)

Tablica 2.1. *Android verzije kroz godine*

Verzija	Datum	API nivo	Svojstva
Cupcake 1.5	27. travanj, 2009.	3	*Widget*-i Različite tipkovnice Animiranje tranzicija
Donut 1.6	15. rujan, 2009.	4	Brzo pretraživanje Različite veličine ekrana Android trgovina
Eclair 2.0. – 2.1.	26. listopad, 2009.	5 – 7	Google karte Prilagodba početnog zaslona Konverzija govora u tekst
Froyo 2.2. – 2.3.	20. svibanj, 2010.	8	Glasovne akcije Prijenosni *hotspot* Performanse
Gingerbread 2.3. – 2.3.7.	06. prosinac, 2010.	9 – 10	API za igrice NFC Menadžment baterije
Honeycomb 3.0. – 3.2.6	22. veljača, 2011.	11 – 13	Dizajn za tablet Sustavna traka Brze postavke
Ice cream sandwich 4.0. – 4.0.4	18. listopad, 2011.	14 – 15	Prilagodba početnog zaslona Kontrola korištenja podatkovne mreže Android *beam*
Jelly Bean 4.1 – 4.3.1.	09. lipanj, 2012.	16 – 18	Google *now* Akcijske notifikacije Prebacivanje računa
Kit kat 4.4. – 4.4.4.	31. listopad, 2013.	19 – 20	*Voice: OK Google* Pametno pozivanje
Lollipop 5.0. – 5.1.1.	12. studeni, 2014.	21 – 22	*Material* dizajn *Multiscreen* Notifikacije
Marshmallow 6.0. – 6.0.1	05. listopad, 2015.	23	*Now on tap* Dozvole za pristup Baterija
Nougat 7.0. – 7.1.2.	22. kolovoz, 2016.	24 – 25	*Multi-window view* Grupirane notifikacije *VR mode*
Oreo **8.0**	21. kolovoz, 2017.	26	*Picture in picture mode* Dizajn *Persistent notifications*

2.6. Povijest Laravel *framework*-a

Prema [11] i [12] Laravel je razvojni okvir za PHP nastalo 2011. godine u svrhu lakšeg, konzistentnijeg i preglednijeg razvoja web aplikacija korištenjem MVC arhitekture. Trenutna verzija je 5.4, a od 1. verzije Laravel razvojnog okruženja prošlo je dosta vremena, a uz to i dosta promjena. Laravel razvojno okruženje po verzijama, godinama i značajkama:

1. Laravel 1 (lipanj, 2011.) – Ugrađena podrška za autentikaciju, jezičnu lokalizaciju, modele, *view*-ove, sesije i rute, ali mu je falila podrška za *controller*-e što je sprječavalo Laravel da postane istiniti MVC razvojni okvir.
2. Laravel 2 (rujan, 2011.) – Uvedene dvije vrlo bitne značajke – podrška za *controller*-e koja je omogućila Laravelu da postane MVC razvojni okvir i *Blade* – sustav za predloške (dopušta lakše ubacivanje PHP varijabli u HTML predloške)
3. Laravel 3 (veljača, 2012.) – vjerojatno najbitnija značajka uvedena u trećoj verziji je *Artisan* CLI (engl. *Command line interface*), sučelje za Laravel kroz *command prompt/*terminal. Također u 3. verziji dodana je i podrška za više sustava za menadžment baza (DBMS – engl. *Database management systems*)
4. Laravel 4 (svibanj, 2013.) – Za ovu verziju napravljena je potpuna prerada postojećeg koda, migracija maketa u odvojene pakete koji su distribuirani kroz *Composer* koji služi kao menadžer za pakete na razini cijele aplikacije.
5. Laravel 5 (veljača, 2015.) – verzija nastala kao rezultat internih promjena koje su završile u promijeni numeriranja Laravel razvojnog okruženja. Nove značajke uključuju: podršku za zakazivanje periodički izvršavanih zadataka kroz paket imenom *Scheduler*, abstrakcijski sloj zvan *FlySystem* koji omogućuje udaljenom spremanju korištenje kao da je lokalni datotečni sustav, poboljšano rukovanje paketima kroz dodatak zvan *Elixir* i pojednostavljena vanjski rukovana autentikacija kroz opcionalni *Socialite* paket.

2.7. Korištene biblioteke, proširenja i arhitektura u izradi mobilne aplikacije

Kod izrade mobilne aplikacije korištena je MVP arhitektura. Android *framework* nema ugrađenu arhitekturu kao što je to slučaj kod Laravel *framework*-a, nego je programer prepušten sebi u odlučivanja koju arhitekturu koristiti i na koji način organizirati aplikaciju što najčešće rezultira neučinkovitim i nepreglednim kodom bez pravog arhitekturalnog dizajna. MVP omogućava razdvojenost aplikacijskih komponenti u različite programske cjeline kako bi olakšao implementaciju poslovne logike, prikaza podataka, komunikacije s API aplikacijom itd. MVP se sastoji od:

- modela – klasa koja predstavlja tablicu u bazi podataka
- pogleda (engl. V*iew*) – XML datoteka u kojoj je definiran dizajn određenog ekrana/segmenta aplikacije

- prezentera – klasa koja je zadužena za svu logiku vezanu uz dohvaćanje, skladištenje i prezentiranje podataka u poglede

Strukturiranosti aplikacije i njezinom efikasnijem radu doprinijele su razne vanjske biblioteke koje se u *gradle* datoteci unutar Android SDK povezuju s aplikacijom.

Retrofit [13] je biblioteka korištena u razvoju mobilne aplikacije koja omogućava lakše slanje zahtjeva na poslužitelj, a uz Retrofit korištena je i *GSON* biblioteka koja omogućava serijalizaciju i deserijalizaciju Java objekata u JSON i obrnuto. To olakšava proces prevođenja podataka modela u transportni tip JSON koji se onda šalje poslužitelju. Isto tako pojednostavljuje i dohvaćanje JSON tipa podataka koji se onda serijaliziraju u Java objekte. Prikaz korištenja Retrofit biblioteke nalazi se na slici 2.3.

```
public interface ChildrenApi {

    @Multipart
    @POST("diplomski_api/api/public/dijete")
    Call<ChildPOJO> createNewchild(@Part("ime") RequestBody name,
                                   @Part("prezime") RequestBody lastname,
                                   @Part("dob") RequestBody age,
                                   @Part("skupina") RequestBody group);

    @Multipart
    @POST("diplomski_api/api/public/dijete/{dijete}")
    Call<ChildPOJO> updateChild(@Path("dijete") String dijeteId,
                                @Part("ime") RequestBody name,
                                @Part("prezime") RequestBody lastname,
                                @Part("dob") RequestBody age,
                                @Part("skupina") RequestBody group);

}
```

Slika 2.3. *Prikaz korištenja Retrofit-a*

Dagger2 [14] biblioteka zadužena je za *dependency injection* - ubacivanje ovisnosti o drugim objektima što uvelike pojednostavljuje ubacivanje klasa koje zahtijevaju npr. kontekst cijele aplikacije ili pojedinog dijela aplikacije.

Butterknife [15] biblioteka pojednostavljuje povezivanje elemenata korisničkog sučelja definiranih u XML datotekama s Java klasama. Primjer korištenja biblioteke prikazan je na slici 2.4. Umjesto da se prvo definira element te onda u *onCreate* metodi pronađe traženi *view* i pretvori u traženi element, Butterknife to sve odradi pozivom metode "*@BindView*".

```
@BindView(R.id.children_recycler)
RecyclerView childrenRecyclerView;
@BindView(R.id.fab)
FloatingActionButton fab;
@BindView(R.id.search_view)
MaterialSearchView searchView;
@BindView(R.id.text_empty_result)
TextView emptyResultTxt;
```

Slika 2.4. *Prikaz korištenja Butterknife-a*

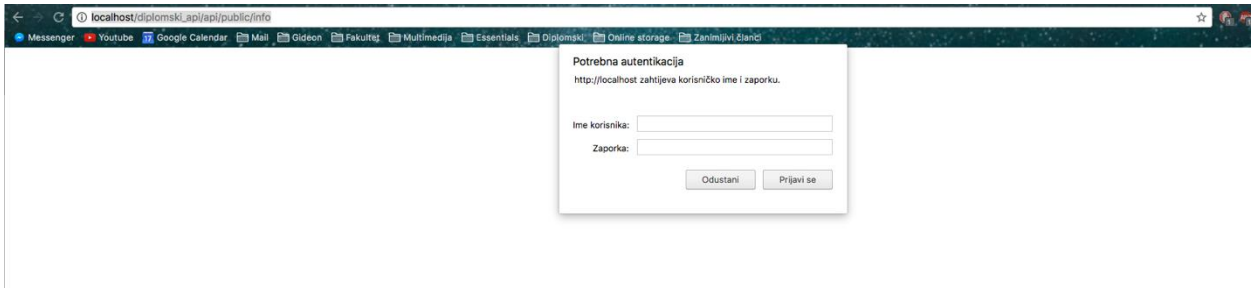
Stetho [16] predstavlja most za otklanjanje pogrešaka u Android aplikacijama. To je biblioteka koja omogućava otklanjanje pogrešaka aplikacije unutar Google Chrome preglednika. Korištenje Android monitora za nadzor baze podataka ponekad zna biti zamorno i nepregledno pa je u te svrhe korištena Stetho biblioteka.

Postoji još par biblioteka koje su korištene u izgradnji mobilne aplikacije, no nisu spomenute u ovom poglavlju jer se više odnose na dizajn te će biti detaljnije opisane u četvrtom poglavlju.

3. DIZAJN APLIKACIJE

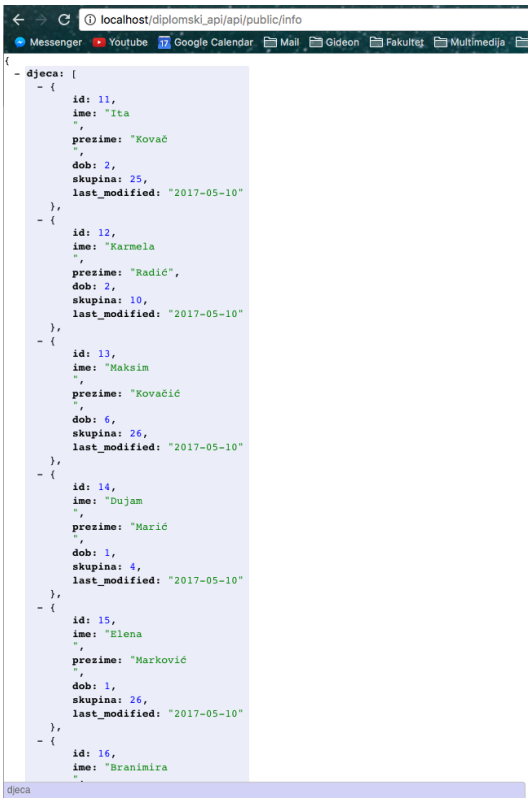
3.1. Dizajn API aplikacije

Prva stavka koju je potrebno definirati kada je u pitanju API [17] aplikacija jest sigurnost. Kako je moguće implementirati raznolike razine sigurnosti bilo je nužno opredijeliti se za onu koja pruža dovoljnu razinu sigurnosti, a da pritom znatno ne komplicira implementaciju aplikacije i na mobilnom i na poslužiteljskom dijelu. Zato je korištena osnovna (engl. *Basic*) autentikacija prema [18]. Osnovna autentikacija sastoji se od slanja korisničkog imena i zaporke u zaglavlju svakog zahtjeva. Takvom implementacijom onemogućen je pristup bazi bilo kome, tko nema korisničko ime i lozinku aplikacije. S druge strane, olakšana je implementacija jer se ne koriste autentikacijski tokeni koji zahtijevaju obnavljanje s vremena na vrijeme te bi bilo potrebno dodatno implementirati i taj mehanizam. Mana ovog pristupa je u tome što su korisničko ime i lozinka pohranjeni lokalno na mobilnoj aplikaciji, ali o tome će biti više govora u četvrtom poglavlju. Na slikama 3.1. i 3.2. prikazano je kako se aplikacija nosi sa neautoriziranim korisnicima koji dolaze na zaštićene adrese te slučajeve kada se korisnik autorizira unesenim podacima.

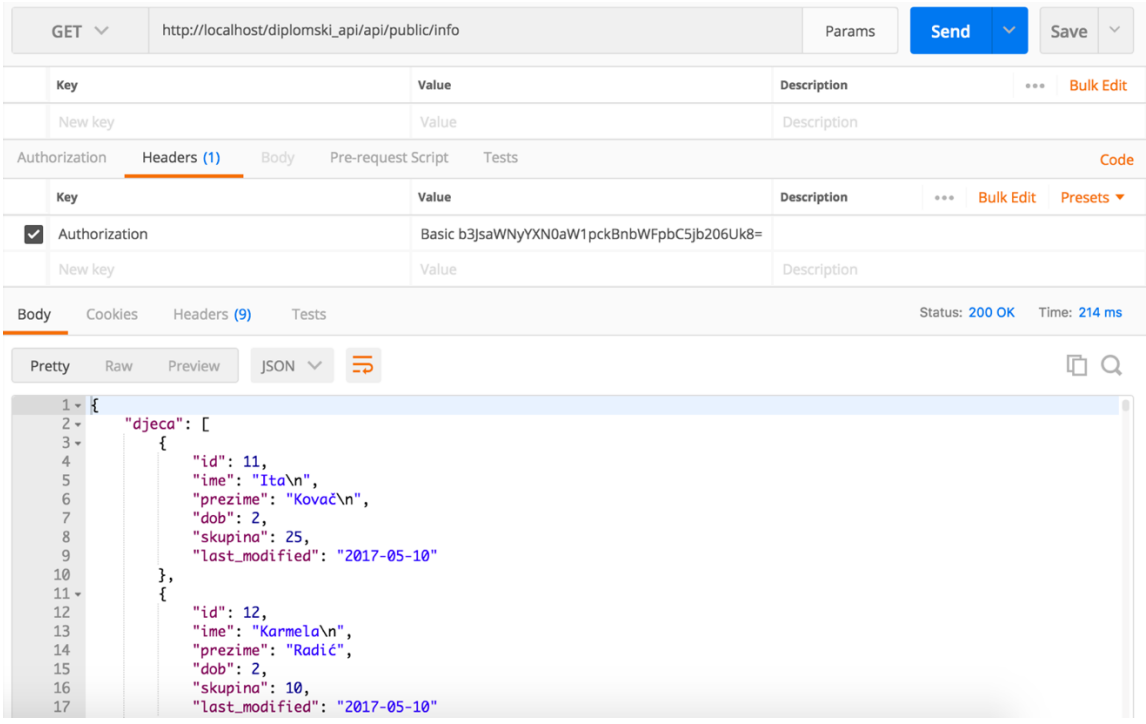


Slika 3.1. *Prikaz dolaska neregistriranog korisnika na putanju aplikacije*

Slika 3.3. također prikazuje slanje zahtjeva na rutu aplikacije, no korištenjem Postman aplikacije. Postman je ustvari proširenje Google Chrome preglednika koje omogućava lakše slanje zahtjeva na poslužitelj, a ima mogućnosti spremanja zahtjeva u kolekcije kako bi se preglednije mogli pohraniti i razdvojiti zahtjevi različitih aplikacija ili npr. Ukoliko imamo istu rutu, a na njoj različite funkcionalnosti (*GET* i *POST*), lako je razdvojiti te zahtjeve. To ubrzava dizajn aplikacije jer se ne mora svaki puta odlaziti na različite adrese i unositi autentikacijske podatke, nego je dovoljno spremiti svaku rutu te ih pozvati onda kada je to potrebno.



Slika 3.2. *Prikaz putanje nakon što je korisnik unio svoje podatke*



Slika 3.3. *Prikaz slanja autoriziranog zahtjeva korištenjem Postman aplikacije*

Bilo je potrebno napraviti rute na poslužitelju [18] koje imaju svoje vlastite kontrolere te prihvaćaju zahtjeve s mobilne aplikacije. Zahtjevi koje API aplikacija prihvaća su *GET*, *PUT*, *POST* i *DELETE* ovisno o tome što je potrebno učiniti. Na slici 3.4. vidljivi su primjeri ruta, tipovi zahtjeva koje rute prihvaćaju i funkcije koje bivaju pozvane u kontroleru. Transportni podaci su JSON formata.

```
MacBook-Pro-od-Rastimir:api rastimirorlic$ php artisan route:list
+--------+-----------+-----------------------------+------------------+-----------------------------------------------------------------+---------------+
| Domain | Method    | URI                         | Name             | Action                                                          | Middleware    |
+--------+-----------+-----------------------------+------------------+-----------------------------------------------------------------+---------------+
|        | GET|HEAD  | api/user                    |                  | Closure                                                         | api,auth:api  |
|        | GET|HEAD  | dijete                      | dijete.index     | App\Http\Controllers\DijeteController@index                     | web,auth.basic |
|        | POST      | dijete                      | dijete.store     | App\Http\Controllers\DijeteController@store                     | web,auth.basic |
|        | GET|HEAD  | dijete/create               | dijete.create    | App\Http\Controllers\DijeteController@create                    | web,auth.basic |
|        | PUT|PATCH | dijete/{dijete}             | dijete.update    | App\Http\Controllers\DijeteController@update                    | web,auth.basic |
|        | GET|HEAD  | dijete/{dijete}             | dijete.show      | App\Http\Controllers\DijeteController@show                      | web,auth.basic |
|        | DELETE    | dijete/{dijete}             | dijete.destroy   | App\Http\Controllers\DijeteController@destroy                   | web,auth.basic |
|        | GET|HEAD  | dijete/{dijete}/edit        | dijete.edit      | App\Http\Controllers\DijeteController@edit                      | web,auth.basic |
|        | POST      | dip                         | dip.store        | App\Http\Controllers\DijetePoremecajController@store            | web,auth.basic |
|        | GET|HEAD  | dip                         | dip.index        | App\Http\Controllers\DijetePoremecajController@index            | web,auth.basic |
|        | GET|HEAD  | dip/create                  | dip.create       | App\Http\Controllers\DijetePoremecajController@create           | web,auth.basic |
|        | PUT|PATCH | dip/{dip}                   | dip.update       | App\Http\Controllers\DijetePoremecajController@update           | web,auth.basic |
|        | GET|HEAD  | dip/{dip}                   | dip.show         | App\Http\Controllers\DijetePoremecajController@show             | web,auth.basic |
|        | DELETE    | dip/{dip}                   | dip.destroy      | App\Http\Controllers\DijetePoremecajController@destroy          | web,auth.basic |
|        | GET|HEAD  | dip/{dip}/edit              | dip.edit         | App\Http\Controllers\DijetePoremecajController@edit             | web,auth.basic |
|        | GET|HEAD  | info                        | info.index       | App\Http\Controllers\ApiController@index                        | web,auth.basic |
|        | GET|HEAD  | info/{info}                 | info.show        | App\Http\Controllers\ApiController@show                         | web,auth.basic |
|        | GET|HEAD  | nalaz                       | nalaz.index      | App\Http\Controllers\NalazController@index                      | web,auth.basic |
|        | POST      | nalaz                       | nalaz.store      | App\Http\Controllers\NalazController@store                      | web,auth.basic |
|        | GET|HEAD  | nalaz/create                | nalaz.create     | App\Http\Controllers\NalazController@create                     | web,auth.basic |
|        | PUT|PATCH | nalaz/{nalaz}               | nalaz.update     | App\Http\Controllers\NalazController@update                     | web,auth.basic |
|        | GET|HEAD  | nalaz/{nalaz}               | nalaz.show       | App\Http\Controllers\NalazController@show                       | web,auth.basic |
|        | DELETE    | nalaz/{nalaz}               | nalaz.destroy    | App\Http\Controllers\NalazController@destroy                    | web,auth.basic |
|        | GET|HEAD  | nalaz/{nalaz}/edit          | nalaz.edit       | App\Http\Controllers\NalazController@edit                       | web,auth.basic |
|        | POST      | poremecaj                   | poremecaj.store  | App\Http\Controllers\PoremecajController@store                  | web,auth.basic |
|        | GET|HEAD  | poremecaj                   | poremecaj.index  | App\Http\Controllers\PoremecajController@index                  | web,auth.basic |
|        | GET|HEAD  | poremecaj/create            | poremecaj.create | App\Http\Controllers\PoremecajController@create                 | web,auth.basic |
|        | PUT|PATCH | poremecaj/{poremecaj}       | poremecaj.update | App\Http\Controllers\PoremecajController@update                 | web,auth.basic |
|        | DELETE    | poremecaj/{poremecaj}       | poremecaj.destroy| App\Http\Controllers\PoremecajController@destroy                | web,auth.basic |
|        | GET|HEAD  | poremecaj/{poremecaj}       | poremecaj.show   | App\Http\Controllers\PoremecajController@show                   | web,auth.basic |
|        | GET|HEAD  | poremecaj/{poremecaj}/edit  | poremecaj.edit   | App\Http\Controllers\PoremecajController@edit                   | web,auth.basic |
|        | GET|HEAD  | validation                  | validation.index | App\Http\Controllers\ValidationController@index                 | web,auth.basic |
+--------+-----------+-----------------------------+------------------+-----------------------------------------------------------------+---------------+
```

Slika 3.4. *Rute, zahtjevi i metode API aplikacije*

Vrlo je lako uočiti da svaka ruta ima posrednika (*engl. Middleware*) – "*auth.basic*" što označava da svaka ruta zahtjeva korisničko ime i lozinku kako bi se obavile tražene funkcije (stvaranje, brisanje, obnavljanje ili prikaz podataka). Na slici 3.5. može se vidjeti metoda za prikaz podataka koja prima datum te vraća sve podatke koji su nakon toga datuma uneseni ili mijenjani. To je primjer funkcije koja biva pozvana kada korisnik opet pokrene aplikaciju (nakon što je već prošao autentikaciju i prijavu) ili kada želi osvježiti podatke.

```php
public function show($date)
{

    $djeca = dijete::where('updated_at', '>', $date)->get();
    $dijete_poremecaj = dijete_poremecaj::where('updated_at', '>', $date)->get();
    $poremecaji = poremecaj::where('updated_at', '>', $date)->get();
    $nalazi = nalaz::where('updated_at', '>' , $date)->get();
    $odgojiteljice = odgojiteljica::where('updated_at', '>' , $date)->get();
    $skupine = skupina::where('updated_at', '>' , $date)->get();
    $vrtic = vrtic::where('updated_at', '>' , $date)->get();

    return response()->json(['djeca' => $djeca,
                        'poremecaji' => $poremecaji,
                        'dijete_poremecaj' => $dijete_poremecaj,
                        'nalazi' => $nalazi,
                        'odgojiteljice' => $odgojiteljice,
                        'skupine' => $skupine ,
                        'vrtici' => $vrtic]);
}
```

Slika 3.5. Primjer funkcije za prikaz podataka nakon određenog datuma u kontroleru

U primjeru koda vidljiv je i Laravelov ORM (engl. *Object Relational Mapping* – olakšani pristup podacima u bazi) imenom *Eloquent*. ORM je način kojim se modele unutar aplikacije direktno mapira na tablice u bazi i tako se dobiva lakši pristup entitetima i pojedinim varijablama. Dovoljno je pozvati klasu koja odgovara tablici u bazi (npr. “dijete” ili “nalaz“) da nam Laravel ponudi određene ugrađene funkcije (npr. “*where*('uvjet')“ ili “all()”) što je znatno jednostavnije od pisanja SQL upita nad bazom.

Na vrlo sličan način implementirane su sve putanje i funkcije koje se pozivaju kada poslužitelj dobije zahtjev na odgovarajućoj adresi. Princip rada za svaku rutu je:

1. Autentikacija korisnika
2. Pozivanje odgovarajuće metode
3. Upit nad bazom
4. Vraćanje podataka u JSON formatu

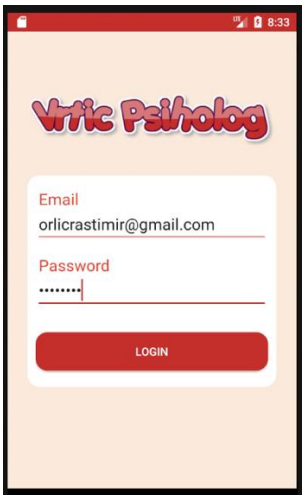
3.2. Dizajn Android aplikacije

Android aplikacija za razliku od API aplikacije ima i korisničko sučelje koje je od izrazite važnosti za svakog korisnika. Dobro dizajnirana aplikacija znači bolje korisničko iskustvo i samim time kvalitetniju aplikaciju. Dizajn je rađen minimalistički s ciljem što bolje prezentacije podataka i što lakšeg snalaženja korisnika. *RecyclerView* [19] je glavni element Android dizajna

korišten za prezentiranje liste/skupine podataka. Korišten je kao vertikalna i horizontalna lista, ali također i kao rešetka (engl. *Grid*). Aplikacija se sastoji većinom od prezentacije podataka u *RecyclerView* elementima te formi koja se popuni kako bi se unio novi nalaz za pacijenta. *RecyclerView* je u stvari poboljšani, optimiziraniji i unaprjeđeniji *ListView* [20] koji unosi neke novine u razvoj Android aplikacija. Prema [21] kao prvo i najbitnije svojstvo, *RecyclerView* uvodi obvezan *ViewHolder*. Dok se prije također mogao implementirati *ListView* s *ViewHolder*-om, sada je to postalo obvezno jer je uočeno kako je taj način populiranja liste s podacima najefektivniji. Drugo jako bitno unaprjeđenje je *LayoutManager* – klasa koja kroz par linija omogućuje potpuno mijenjanje smjera učitavanja podataka u *RecyclerView* kao i samu prezentaciju (vertikalna i horizontalna lista, *grid*). *Item Animator* i *Item Decorator* su također dvije nove značajke koje *RecyclerView* čine vizualno efektivnijim i moćnijim. S animatorom je omogućen različit način animiranja samih stavki u listi, a *decorator* je pružio jasnije i lakše dodavanje razmaka između stavki. Korištenje *RecyclerView*-a je postao industrijski standard u razvoju Android aplikacija.
U svakom segmentu aplikacije moguće je pretraživati liste podataka. Aplikacija se sastoji od 7 dijelova:

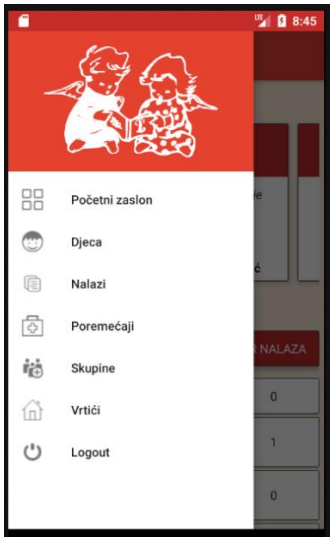
1. Početni zaslon
2. Djeca
3. Nalazi
4. Poremećaji
5. Vrtići
6. Sinkronizacija
7. Odjava

Početni zaslon prikazuje zadnjih 10 unesenih nalaza i 10 zadnje uređivanih podataka o djeci. Na zaslonu “djeca” moguće je pretraživati informacije o djeci te dodavati novo dijete u bazu. Isto je moguće i na zaslonu nalazi dok se na zaslonima poremećaji, skupine i vrtići može samo pregledavati i pretraživati podatke o pojedinim objektima. Zaslon profil pruža uvid u osnovne informacije o profilu – koliko nalaza je korisnik (psiholog) izdao te pruža mogućnost mijenjanja lozinke. Slike 3.6., 3.7. i 3.8. prikazuju zaslon za prijavu, početni zaslon i zaslon s otvorenim *NavigationView* elementom.



Slika 3.6. *Prijava u aplikaciju*

Slika 3.7. *Početni zaslon nakon prijave*



Slika 3.8. *Početni zaslon s otvorenim NavigationView elementom*

Pošto svi zasloni koriste istu *NavigationView* [22] navigaciju, konstruiran je *DrawerActivity* – Aktivnost koja sadrži *NavigationView* i *FrameLayout* u koji se ugrade elementi specifični za svaki zaslon. Svaka aktivnost nasljeđuje *DrawerActivity* te na taj način ima pristup navigaciji aplikacije, a može ugraditi svoje elemente u njega. Time je postignuta optimiziranost koda kako se ne bi za svaki zaslon morao implementirati *NavigationView* navigacija.

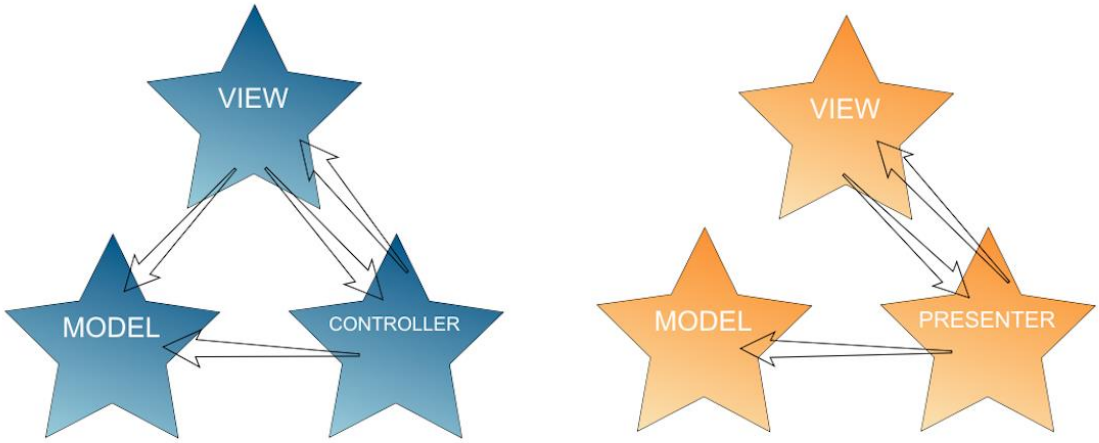
3.3. MVP u aplikaciji

Aplikacija je strukturirana na način da je svaki zaslon zaseban folder koji u sebi sadržava sučelja (model, *presenter* i *view*) te klase: *Activity* koja implementira *view* sučelje, Model koja implementira model sučelje i *Presenter* klasa koja implementira *presenter* sučelje. Korištenje sučelja dosta olakšava razvoj aplikacije. Recimo na primjer da *Activity* (zaslon) treba prikazati filtriranu listu podataka. Logički slijed tog primjera bio bi sljedeći:

1. Korisnik unosi pojam kojeg traži te pritiska gumb za traženje
2. *Activity* poziva funkciju na *Presenter* klasi
3. *Presenter* kontaktira Model klasu koja vrši upit nad bazom podataka te vraća podatke
4. Model vraća *Presenter* klasi podatke te *Presenter* poziva funkciju za prikaz podataka u *Activity* klasi
5. Korisnik dobiva prikazanu listu filtriranih podataka

Iako se možda čini kompliciran, ovakav pristup znatno olakšava proces otklanjanja grešaka i testiranja. Točno se zna koji dio aplikacije je zadužen za koju funkcionalnost. *Presenter* komunicira s *view*-om, obavlja funkcije koje su potrebne (u ovom slučaju upit nad bazom podataka – modelom) te prezentira dobivene podatke nazad u *view*. Ovo nije jedini primjer gdje sučelja olakšavaju razvoj aplikacije. Još jedan primjer gdje su znatno došla do izražaja je prilikom slanja upita na poslužitelj. Pošto se svaki *HttpRequest* nakon Android verzije 4.0 mora slati asinkrono (na drugoj niti), potrebno je znati kada je zahtjev završen. To se može ostvariti koristeći klasu *AsyncTask* koja je dio Android SDK-a, a koja je korištena prilikom početnog dohvaćanja svih podataka iz baze, no također moguće je koristiti i sučelja. Kada se poziv koristeći *Retrofit* biblioteku postavi u red (engl. *Enqueue*) nastavlja se rad aplikacije, no kada se poziv izvrši pozivanjem metode sučelja koje implementira neka klasa rezultira izvršavanjem one funkcije koju je potrebno izvršiti nakon uspješnog ili neuspješnog poziva. Na taj način se programer ne mora bojati da će izgubiti tok programa te preskočiti funkciju koja mu je potrebna. Funkcija koja je implementirana kroz sučelje može se pozvati bilo kada (u ovom slučaju nakon

što je poziv prema API aplikaciji izvršen). Na taj način implementacija sučelja postaje svojevrsni osluškivač (engl. *Listener*). Zato je MVP arhitektura implementirana kroz sučelja. Na taj način se omogućuje potpuno objektno orijentiran kod bez brige o preskakanju dijelova koda. Bitno je također razlikovati MVP i MVC arhitekturu. Na slici 3.9. je vrlo jednostavno predstavljena njihova razlika.



Slika 3.9. *MVC i MVP*

Prema [23] MVC arhitektura kao i MVP dijeli aplikaciju na tri dijela koja imaju različite odgovornosti. Kod MVC arhitekture, model sadrži podatke, stanja i poslovnu logiku aplikacije.

- Model je mozak cijele aplikacije. Nije direktno vezan za određeni *view* ili *controller* i zbog toga može biti više puta iskorišten i na više mjesta.
- *View* ima zadaću prezentacije modela. *View* mora prikazivati UI, ali isto tako i komunicirati s *controller*-om kada korisnik ima interakciju s aplikacijom. U MVC arhitekturi, *view* dijelovi su vrlo jednostavni i mogli bismo reći glupi. Ne sadrže skoro nikakvu logiku niti znanja o modelima, a ne mogu razumjeti ni što treba učiniti kada korisnik klikne određeni gumb, niti u kojem stanju se nalazi aplikacija. Ideja je da budu takvi kako bi što manje bili vezani za određeni model i na taj način biti podložniji promjenama.
- *Controller* je ono što povezuje sve dijelove aplikacije zajedno. On je glavni nadležni za sve što se događa u aplikaciji. Kada *view* kaže *controller*-u da je korisnik pritisnuo gumb, *controller* odlučuje kako će ostvariti interakciju s modelom i to predočiti korisniku. *Controller* tu može odlučiti različite stvari, npr. Ukoliko je prije unešena lozinka,

provjeriti njenu ispravnost prije nego što se pošalje novi zahtjev na poslužitelju. Kod MVC arhitekture, *controller* je gotovo uvijek *Activity* ili *Fragment*.

MVP arhitektura razlaže *controller* tako da se na *Activity* može gledati kao *view* što je nekako i prirodno u Androidu.

- Model je isti kao i kod MVC arhitekture.
- View je malo drugačiji. U MVP arhitekturi, *Activity* i *Fragment* su smatrani kao dio *view*-a. Najbolja praksa je implementirati sučelje kako bi se *Presenter* mogao preko sučelja povezivati na *view*, a isto tako to omogućava lakše testiranje tako da se *view* može oponašati i spojiti s *Presenter*-om.
- *Presenter* je zapravo *controller* iz MVC arhitekture, ali nije nimalo povezan s *view*-om, samo preko sučelja. To također doprinosi ispitljivosti (testiranju) aplikacije, a povećava i modularnost i fleksibilnost što je kod MVC arhitekture znalo predstavljati značajan problem.

4. *BACKEND* APLIKACIJE

Backend ili dio u pozadini, skriveni dio, predstavlja sve ono što korisnik ne vidi, a događa se u pozadini kako bi korisniku bili prezentirani oni sadržaji koje vidi i s kojima je u interakciji. Do sad je već opisana arhitektura aplikacije (MVP), no u ovom poglavlju bit će više govora o poslovnoj logici i bazama podataka. Kako je već objašnjeno, Laravel ima svoj vlastiti ORM mehanizam kojim se olakšava pristup podacima u bazi, ali Android nema. Moguće je koristiti neke vanjske biblioteke za ORM u Androidu, no u samom Android SDK dohvaćanje podataka je ostvareno kroz pokazivače. To nisu isti pokazivači kao u programskom jeziku C. Pokazivači (*engl. Cursor*) u Javi pokazuju na retke u tablici, odnosno oni su rezultat upita nad bazom.

Nakon što se pošalje upit nad bazom, funkcija *query* vraća pokazivač na sve dohvaćene elemente. Naredbama *moveToFirst* i *moveToNext* moguće je iterirati kroz sve elemente te ih onda mapirati u željene objekte/modele. Slika 4.1. prikazuje pretraživanje baze s poremećajima, njihov dohvat te mapiranje u objekte modela.

```
@Override
public ArrayList<Disorder> getDisordersData(Context context, MyDbHelper myDbHelper, SQLiteDatabase db) {
    Cursor disordersCursor = null;
    try{
        disordersCursor = db.query(DatabaseEntries.DISORDER_TABLE_NAME,
                DatabaseEntries.DISORDER_COLUMNS,
                null,
                null,
                null,
                null,
                DatabaseEntries.DISORDER_NAME + " ASC ",
                null);
        if(disordersCursor.moveToFirst()){
            do{
                Disorder disorder = new Disorder(disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.DISORDER_ID)),
                        disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.DISORDER_NAME)),
                        disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.DISORDER_DESCRIPTION)),
                        disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.DISORDER_THERAPY)),
                        disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.CREATED_AT)),
                        disordersCursor.getString(disordersCursor.getColumnIndex(DatabaseEntries.UPDATED_AT)));

                allDisorders.add(disorder);
            }while(disordersCursor.moveToNext());
        }

    }finally {
        disordersCursor.close();
    }

    return allDisorders;
}
```

Slika 4.1. Prikaz korištenja pokazivača u čitanju podataka iz baze

Kako se ne bi moralo pamtiti svaku varijablu svakog entiteta u tablici, implementirana je klasa *DatabaseEntries* koja u sebi sadrži imena svih varijabli, svih entiteta, grupirane popise stupaca itd. To je također vidljivo u slici 12 kada u *query* funkciju za parametar bude predan *DatabaseEntries.DISORDER_COLUMNS*. U varijabli *DISORDER_COLUMNS* sadržani su svi stupci (varijable) entiteta *disorder*. Ova funkcija vraća popis svi poremećaja sa svim pripadajućim varijablama, točnije, tablica poremećaj sa svim njezinim stupcima mapirana je u

objekt poremećaj koji sadrži sve stupce u varijablama. Ova funkcija se nalazi u modelu, a pozvana je od strane *presenter*-a nakon što je *view* zatražio podatke da može prezentirati ono što mu je potrebno. Tok naredbi.

1. Instanciranje *view*-a
2. *View* traži od *presenter*-a podatke koji mu trebaju
3. *Presenter* dohvaća podatke iz modela
4. Model vrši upit nad bazom i vraća podatke
5. *Presenter* šalje podatke u *view*
6. *View* prezentira sve podatke

Da se dobije dojam kolika je razlika u razvoju između mobilnih aplikacija i web aplikacija prikazan je i tok naredbi za dohvat svih poremećaja u *Laravel framework*-u. Cijela funkcija za dohvat poremećaja u Android bazi, zamijenjena je jednom linijom koda prikazanoj na slici 4.2.

```
$poremecaji = poremecaj::all();
```

Slika 4.2. *Dohvat poremećaja pomoću Laravel eloquent-a*

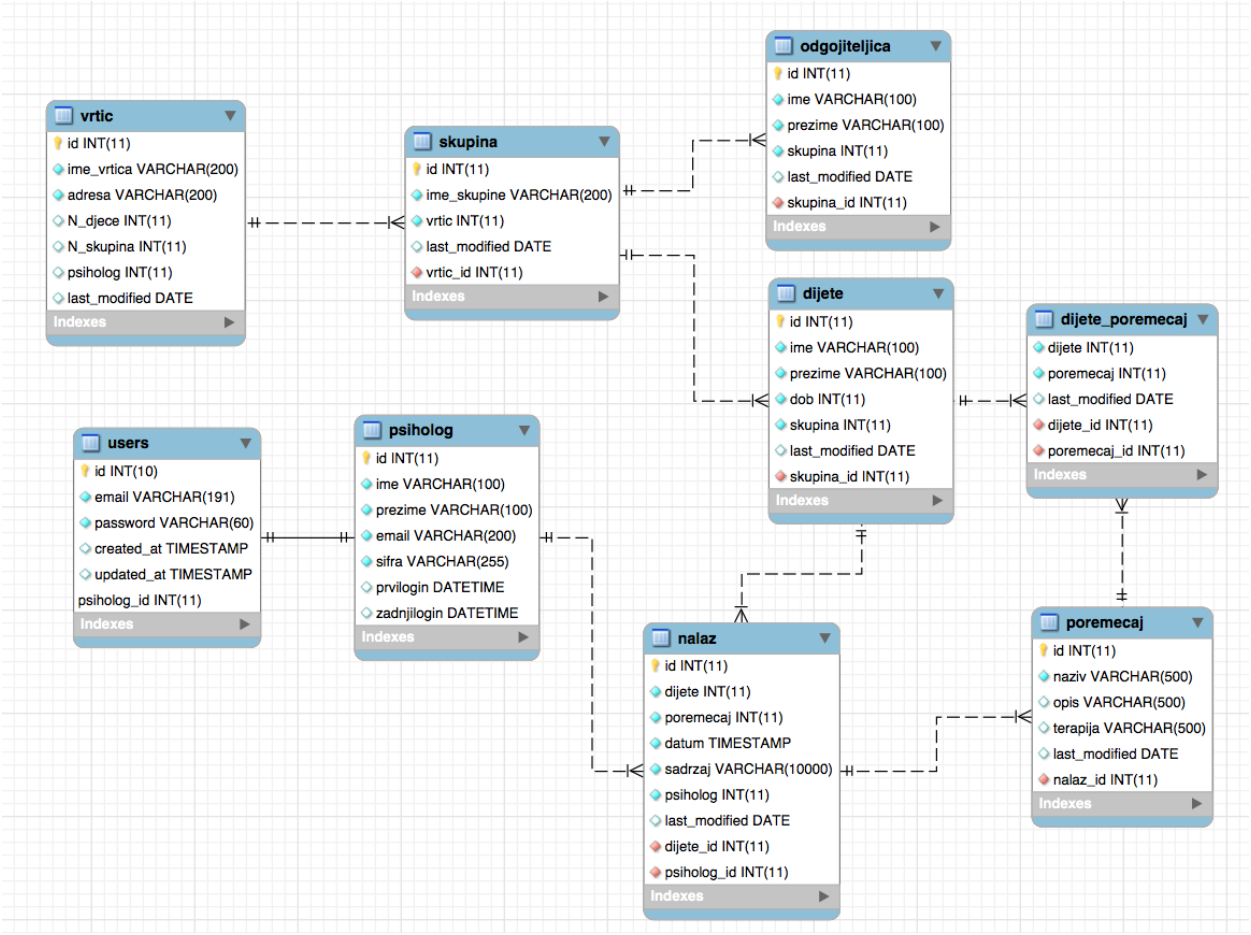
4.1. Dizajn baze podataka

Na slici 4.3. vizualno je prikazana shema baze podataka na strani poslužitelja kojoj mobilna aplikacija pristupa preko API aplikacije. Za shematski prikaz baze korištena je aplikacija MySQLWorkbench verzije 6.3. Aplikacija se spaja na poslužitelj te korisnik odabire bazu za koju želi napraviti shematski prikaz, tako na vrlo lak način omogućuje pregledan i učinkovit prikaz same baze kako bi se korisnik mogao bolje snaći u vezama. To posebice ima velik značaj kod velikih baza podataka gdje veze između entiteta nisu uvijek najjasnije iz imena ili samo poslužiteljskog prikaza.

Arhitektura baze je vrlo jednostavna i lako čitljiva iz slike. Popis entiteta proizvod je *brainstorming* metode u kojoj se pišu svi pojmovi kojih se osoba može sjetiti, a da su vezani uz određeni pojam, u ovom slučaju za vrtićkog psihologa. Nakon što se eliminiraju sve one nebitne stvari ostaju one najbitnije usko vezane za aplikaciju. Tada se vrši selekcija nad njima i dolazi se do popisa entiteta koji su najnužniji za bazu podataka i aplikaciju. Tada je potrebno utvrditi veze i odnose između entiteta te na samom kraju napisati *SQL* kod za stvaranje baze. U bazu su podaci za djecu i odgojiteljice uneseni na način da su u različitim datotekama bila zapisana

imena i prezimena sa stranica koje sadrže popis različitih Hrvatskih imena i prezimena. Napisana je PHP skripta koja nasumično povezuje imena i prezimena te su ona spremana u bazu. Za vrtiće korišteni su stvarni podaci o imenima i adresama vrtića u Osijeku, dok su skupine potpuno nasumično ubačene u vrtiće. Poremećaji su dodavani također nasumično sa različitih stranica koje sadrže popise poremećaja.

Što se same sheme baze tiče, objekt vrtić može imati više skupina (npr. mlađa, srednja, predškolska…) te je stoga veza između ta dva objekta 1 na više. Isto tako skupina može imati više odgojiteljica i više djece. Kod djeteta može biti zabilježeno više poremećaja, ali isto tako jedan poremećaj može bit zabilježen kod više djece. Ovdje je veza više na više te je stoga implementiran objekt između njih**,** tzv. pivot tablica u kojem se događa spajanje objekta dijete i poremećaj. Za svako dijete je moguće napisati više nalaza i isto tako za svaki poremećaj može biti napisano više nalaza. Stoga je dijete u vezi jedan na više s nalazom, a isto tako i poremećaj. Psiholog može napisati više nalaza, a psiholog i korisnik su u vezi 1 na 1 jer je svaki psiholog i korisnik. Svaki objekt u bazi na poslužitelju preslikan je i u lokalnu bazu podataka na mobitelu, samo što je kod Android operativnog sustava korišten SQLite naspram MySQL baze koju koristi poslužitelj.



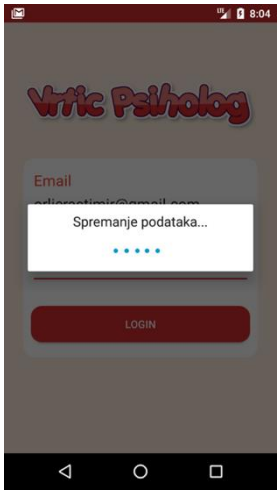
Slika 4.3. *Baza podataka na poslužitelju*

4.2. Skladištenje, slanje i osvježavanje podataka

Pošto je u ovoj aplikaciji implementirana lokalna baza podataka, podatke je potrebno dohvaćati, slati i osvježavati. Dohvaćanje podataka vrši se pri samom pokretanju aplikacije. Ukoliko je korisnik već bio prijavljen, ostat će zapamćeni njegovi podaci i direktno će se ažurirati baza, a ako korisnik nije bio prijavljen ili se odjavio iz aplikacije, morat će unijeti adresu svoje elektroničke pošte i lozinku kako bi se prijavio te tada dohvatio sve podatke. Dohvat podataka implementiran je na samom početku rada s aplikacijom radi što optimalnijeg korisničkog iskustva te kako se ne bi dogodilo da su u bazu uneseni novi podaci, a korisnik to ne vidi na svom mobilnom uređaju. Također, podatke je moguće ažurirati i kasnije u aplikaciji odabirući "Sinkronizacija" u izborniku. Kada se podaci sinkroniziraju odnosno vrši se ažuriranje podataka, dohvaćaju se svi unosi iz baze na poslužitelju koji u "*updated_at*" atributu imaju datum poslije datuma zadnjeg dohvata podataka. Datum zadnjeg dohvata sprema se u pričuvnu memoriju

Android aplikacije odnosno *Shared Preferences*. Obje ove funkcionalnosti (dohvat i sinkronizacija podataka) koriste vrlo bitnu komponentu Android SDK-a – *AsyncTask*.

AsyncTask [25] je pomoćna klasa dizajnirana oko *Thread* i *Handler* klase, a služi za pravilno i lako iskorištavanje UI niti. Ona omogućava izvođenje operacija u pozadini te objavljivanja rezultata na UI niti bez da programer mora manipulirati nitima i/ili rukovateljima. Google u svojim Android smjernicama navodi činjenicu da *AsyncTask* ne bi trebao biti korišten za operacije koje traju duže od par sekundi pošto on blokira korištenje korisničkog sučelja dok se ne izvrše sve operacije u pozadini. Pošto se u ovoj aplikaciji ne radi o prevelikoj količini podataka, zahtjev za dohvat podataka i njihovo spremanje u bazu ne bi trebalo nikad trajati dulje od par sekundi. Na slici 4.4. vidljiv je dijalog koji se pokaže prilikom dohvata podataka.



Slika 4.4. *Dohvat podataka kod prijave*

Dok traje proces dohvata i spremanja podataka, korisnik je prisiljen čekati završetak procesa u pozadini, a kako bi znao zašto čeka i zašto ništa ne može kliknuti, predstavljen mu je *Spots Dialog*, biblioteka koja animira Android *Dialog* točkicama koje dolaze s lijeve strane i "izlaze" na desnu stranu.

AsyncTask se definira koristeći tri generička tipa:

1. *Params* – tip podataka koji se šalju na *AsyncTask*-u i bitni su za samu izvedbu
2. *Progress* – tip podataka koje *AsyncTask* može objavljivati tokom trajanja pozadinskog procesa
3. *Result* – tip podataka koji predstavlja rezultat pozadinskog procesa

Osim što je za *AsyncTask* definirano tri tipa podataka, on je podijeljen u četiri bitna koraka koje valja razumijeti kako bi se ispravno implementirao:

1. *onPreExecute* – kako samo ime kaže, ovo je metoda koja se poziva neposredno prije izvršavanja pozadinskih zadataka, a u ovom slučaju korištena je za postavljenje *Spots Dialog view*-a.
2. *doInBackground* – u ovoj metodi implementirani su svi zadaci koji se obavljaju u pozadini
3. *onProgressUpdate* – metoda zaslužna za prikazivanje progresa, npr. u slučajevima kada je prikazivan postotak izvršenosti pozadinskih zadataka
4. *onPostExecute* – metoda pozvana nakon izvršavanja pozadinskih podataka

Slika 4.5. prikazuje implementaciju potrebnih metoda *AsyncTask*-a s izuzetkom prikaza cijele *doInBackground* metode zbog njezine opširnosti (preko 500 linija koda) – populira sve tablice s podacima tako što deserijalizira JSON u objekte definiranih modela te tada koristeći *Cursor*-e sprema podatke u bazu.

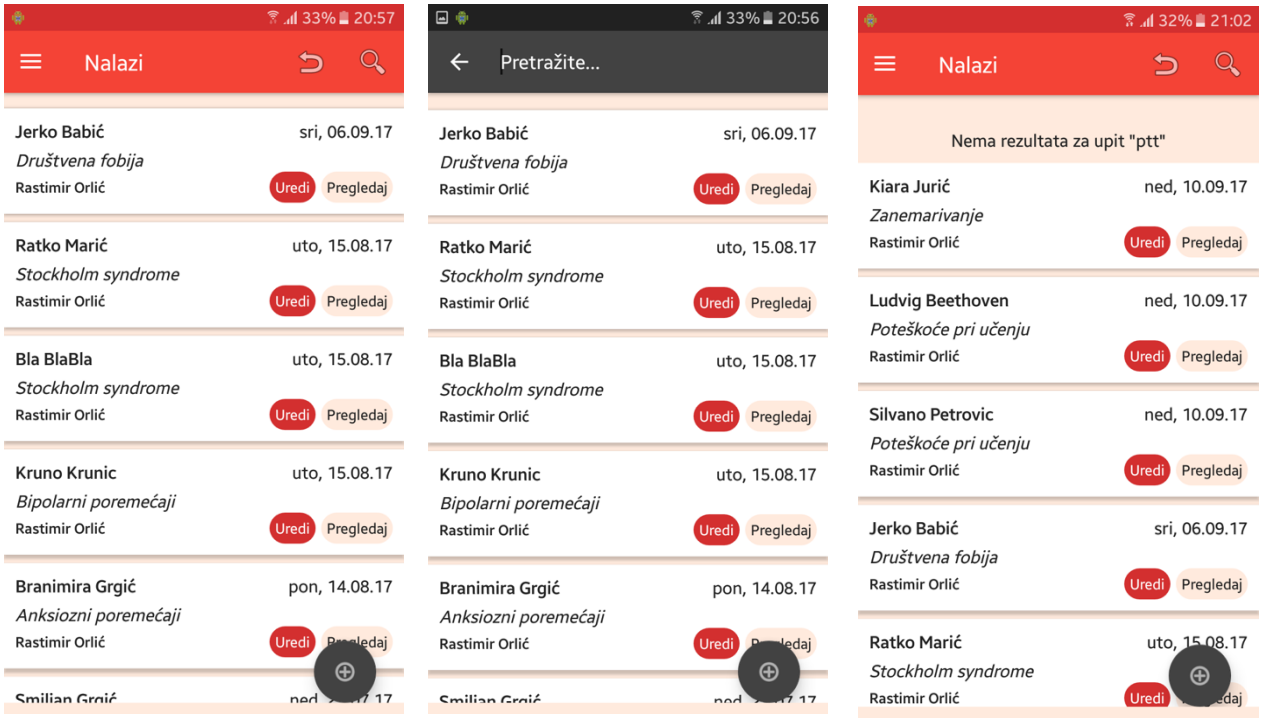
```
@Override
protected void onPreExecute() {
    dialog = new SpotsDialog(mContext, "Spremanje podataka...");
    dialog.show();
}

@Override
protected void onPostExecute(Void aVoid) {
    if(dialog.isShowing()){
        dialog.dismiss();
    }

    listener.onDownloadFinished();
}

@Override
protected Void doInBackground(String... params) {

    MyDbHelper myDbHelper = new MyDbHelper(mContext);
    SQLiteDatabase db = myDbHelper.getWritableDatabase();
```

Slika 4.5. *Metode AsyncTask-a*

5. UPUTE ZA KORIŠTENJE APLIKACIJE I TESTIRANJE

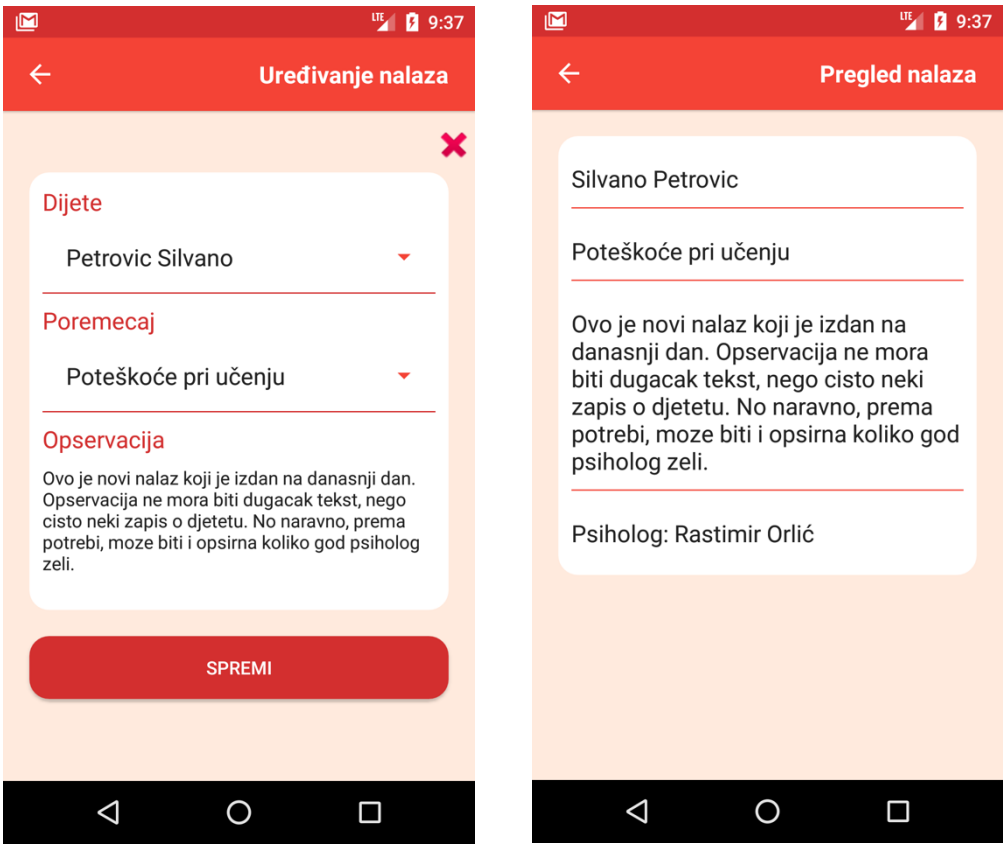
5.1. Upute za upotrebu

Nakon što je korisnik prijavljen u aplikaciju dobiva popis najnovije unesenih nalaza i podataka o djeci u bazu. Dodirom na dijete ili nalaz otvaraju se zasloni s prikazom detaljnijih podataka. Prikaz svih nalaza te njihovo pretraživanje moguće je kroz odabir sekcije “Nalazi“ u izborniku. Tada korisniku budu prikazani svi nalazi (slika 5.1.) poredani redom po datumu izdavanja/ažuriranja.



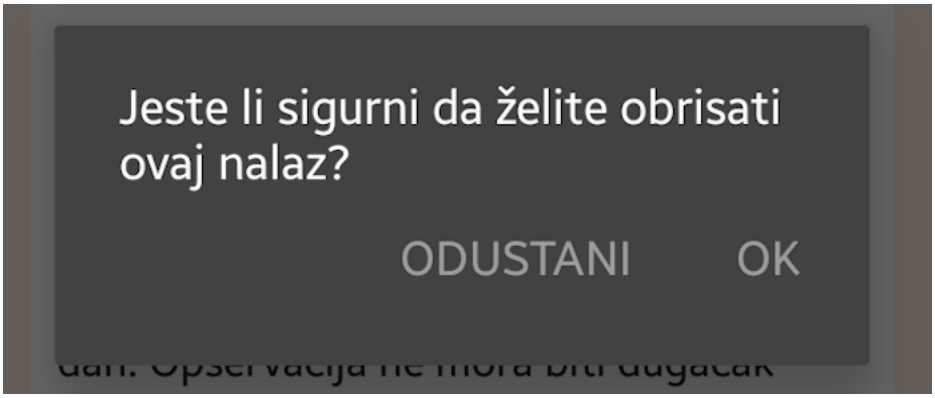
Slika 5.1. Zaslon nalaza

Ako ne postoji rezultat za traženi pojam pojavljuje se obavijest o tome iznad popisa nalaza, a ukoliko ima rezultata prikazana je lista samo s rezultatima te se vrlo lako može vratiti popis svih nalaza pritiskom na ikonicu pored pretraživanja. Pretraživanje je implementirano po imenu i prezimenu djeteta, imenu poremećaja te imenu i prezimenu psihologa. Svaki nalaz moguće je pregledati ili urediti. Pregled nalaza dopušta korisniku da na brzinu vidi detalje o samom nalazu dok uređivanje dopušta i mijenjanje samog nalaza te brisanje. Ta dva zaslona prikazana su na slici 5.2.



Slika 5.2. Uređivanje i pregled nalaza

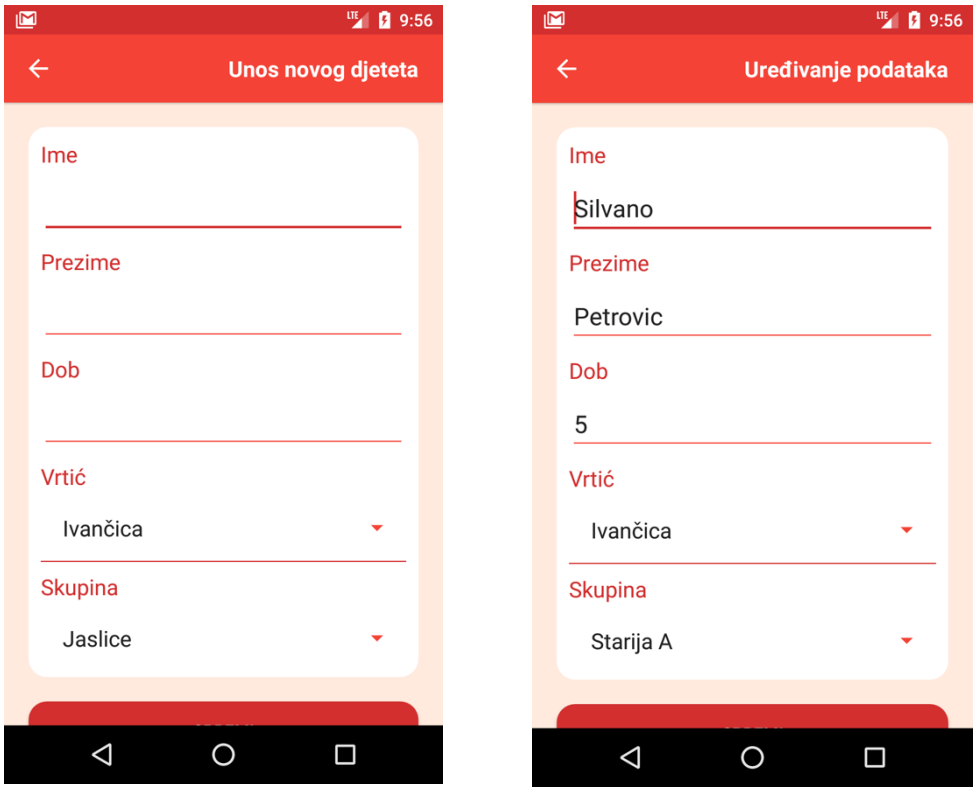
Brisanje nalaza prikazano je korisniku kroz dijalog na slici 5.3. Tako je izbjegnuta situacija da korisnik slučajno stisne na gumb za brisanje te nehotice obriše nalaz iz baze. Također, kod uređivanja nalaza odabir djeteta i poremećaja je onemogućeno, a omogućeno je jedino mijenjanje opservacije jer bi mijenjanjem ta dva podatka nastajao zapravo posve novi nalaz.



Slika 5.3. Dijalog za brisanje nalaza

Na sličan način implementirani su i zasloni za djecu, samo što kod djece izostaje zaslon za prikaz podataka jer je sve vidljivo iz same stavke za pojedinačno dijete. Slika 5.4. prikazuje zaslone za dodavanje i uređivanje podataka o djetetu. U ovom slučaju omogućeno je mijenjanje

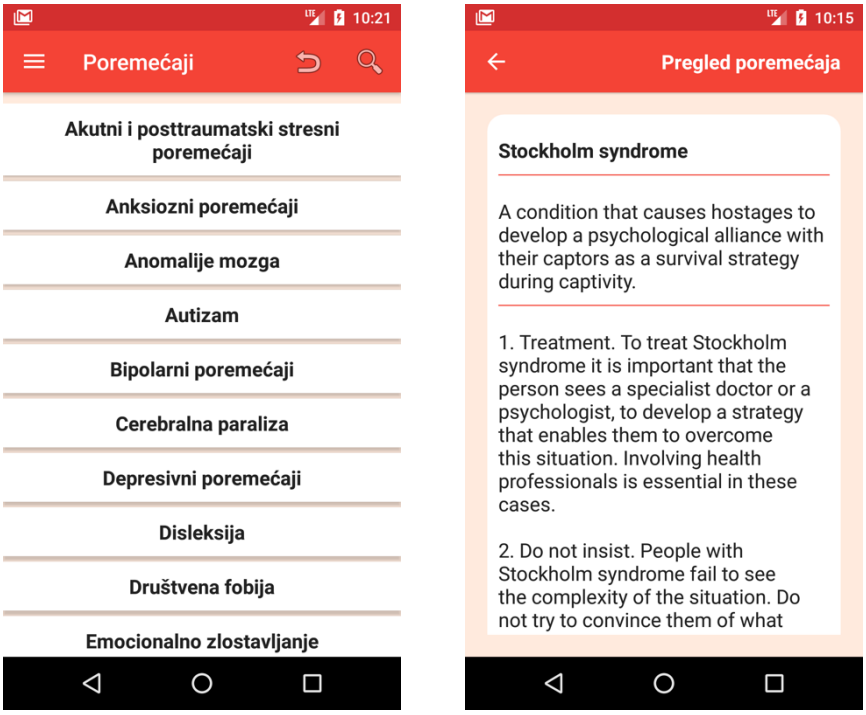
vrtića i skupina jer je vrlo vjerojatan slučaj da dijete prijeđe iz jedne skupine u drugu, a također je moguće i da se dijete premjesti u drugi vrtić.



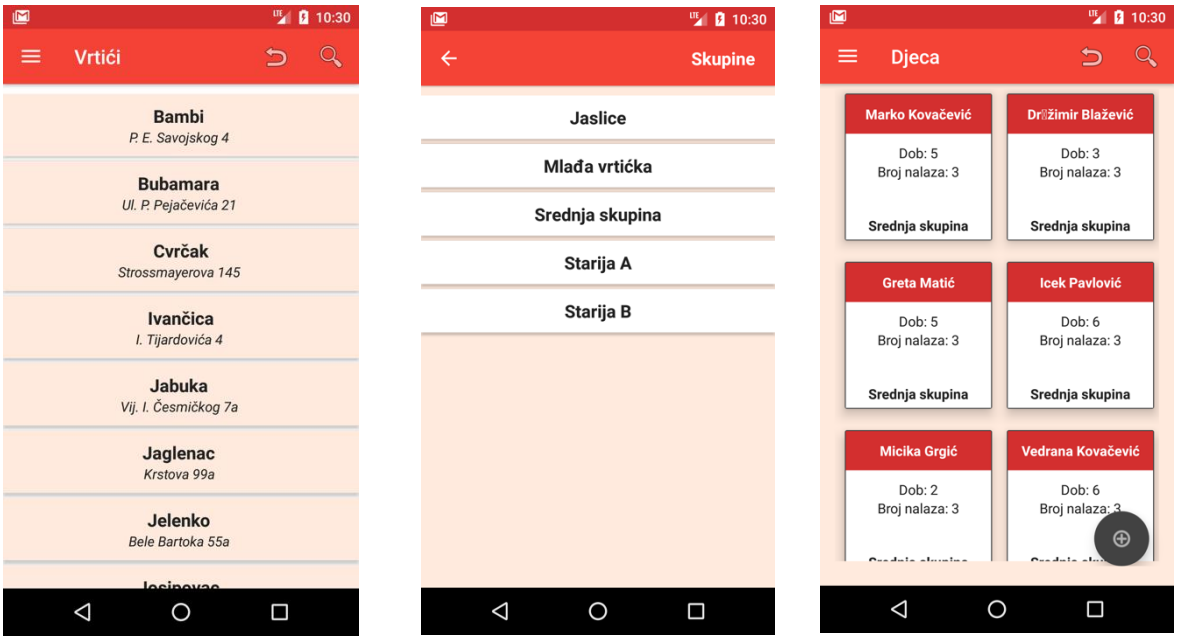
Slika 5.4. Uređivanje i dodavanje podataka o djetetu

Za prikaz skupina potrebno je raditi upit nad bazom za svaki vrtić pojedinačno u trenutku kad se vrtić odabere. Ovdje je vidljiva prednost lokalne baze podataka s kojom takvi upiti idu znatno brže, nego da je aplikacija rađena po *thin client* principu te sve podatke vukla sa servera i ne skladištila ništa veće. Tada bi se za svaki odabrani vrtić morao slati zahtjev na server, taj zahtjev obraditi, spremiti podatke i prikazati ih. Ovako je odgovor od lokalne baze gotovo trenutan.

Aplikacija je još zadužena i za prikazivanje poremećaja, osnovnih informacija o pojedinačnom poremećaju (ime, opis i terapija) te vrtića, skupina i sve djece koja pripadaju određenoj skupini. Ti zasloni prikazani su na slikama 5.5. i 5.6. Također, za poremećaje i vrtiće implementirano je pretraživanje kao i kod zaslona za djecu i nalaze.



Slika 5.5. Zasloni za pregled poremećaja



Slika 5.6. Zasloni za pregled vrtića, skupina, i djece u pojedinoj skupini

5.2. Testiranje

Testiranje mobilne aplikacije nije rađeno kroz kod, nego ručno na različitim uređajima i emulatorima kako bi se ostvario što pouzdaniji dizajn korištenjem različitih veličina ekrana, ali također i različitim verzijana Android operacijskog sustava (što znatno može promijeniti performanse i značajke aplikacije).

Emulatori korišteni u testiranju:

1. Nexus 5 – API nivo 24
2. Nexus 4 – API nivo 21

Uređaji korišteni u testiranju:

1. Samsung Galaxy J5 (2016)
2. Doogee X6

Aplikaciju je moguće koristiti u oba načina – vertikalno i horizontalno, no optimizirana je za vertikalno korištenje. Da bi se optimizirao dizajn aplikacije za horizontalno korištenje moralo bi se razmišljati i o načinu prikaza na tabletu što bi znatno povećalo napore u osmišljavanju dizajna pa je ta problematika ostavljena za neku buduću inačicu aplikacije.

6. ZAKLJUČAK

U današnje vrijeme kada je računalo i pametni telefon moguće naći na gotovo svakom radnom mjestu ukazala se potreba za modernizacijom i digitalizacijom radnog mjesta predškolskog psihologa. U te svrhe izrađen je ovaj rad. Produkt diplomskog rada su dvije potpuno funkcionalne aplikacije koje rješavaju problematiku rada psihologa u vrtićima omogućavajući im unošenja nalaza i praćenja napretka razvoja djece. Aplikacije su dizajnirane s najnovije korištenim tehnologijama u svijetu mobilnih i web aplikacija. Android aplikacija koju korisnik vidi i s kojom je stalno u doticaju nudi pregled svih vrtića, skupina, poremećaja, djece te napisanih nalaza. Također, omogućuje unošenje novih nalaza te unos nove djece u sustav. Za korištenje aplikacije je potrebna konekcija na internet, no ona nije nužna u trenutku pregledavanja nalaza, djece ili pretraživanja navedenih sadržaja. API aplikacija koja je nevidljiva korisniku omogućava Android aplikaciji komunikaciju s bazom podataka na poslužitelju te koristi jednostavnu autentikaciju putem korisničke email adrese i lozinke.

Uočenih manjkavosti na aplikaciji nema, no moguće je unaprijediti rad i opseg aplikacije dodavanjem kalendara i rasporeda za psihologa kako bi se korisnici aplikacije mogli što bolje organizirati za rad s djecom, roditeljima, odgojiteljima itd. Također, kada bi se ovaj rad išlo komercijalizirati i stvoriti jedan kompletan proizvod za tržište, aplikacije bi se morale proširiti na način da se prilagode opsegom poslužitelja za veći broj korisnika te da se unaprijedi sigurnost API aplikacije uvođenjem *OAuth2* autentikacije koja počiva na pristupnim i osvježavajućim znakovima (engl. *token*). Tako bi se izbjeglo slanje lozinke i korisničke email adrese putem interneta. Također, bilo bi dobro u sklopu API aplikacije implementirati korisničko sučelje za dodavanje ostalih objekata u bazu (vrtića, skupina, odgojiteljica) te uposliti administratora takve stranice. Jedno vrlo zanimljivo proširenje aplikacije bilo bi mogućnost unošenja podataka dok aplikacija nije povezana na Internet. Nakon što se ostvari konekcija bilo bi potrebno pročitati sve unesene podatke te ih poslati na poslužitelj. Za to proširenje trebala bi se implementirati još jedna baza podataka na uređaju koja bi bila zadužena samo za skladištenje *offline* podataka. Punila bi se i praznila po potrebi.

Kada se uzmu u obzir zahtjevi koje aplikacija treba zadovoljiti i zadatke koje treba obaviti, implementirana aplikacija uspješno odgovara na njih sve te ima dobro postavljene temelje za daljnji razvoj, napredak i proširivanje.

LITERATURA

[1] Što radi psiholog u vrtiću,
http://www.istrazime.com/djecja-psihologija/psiholog-u-vrticu/, pristupljeno: svibanj, 2017.

[2] Standardi rada psihologa u predškolskim ustanovama,
http://www.psiholoska-komora.hr/293, pristupljeno: svibanj, 2017.

[3] Osnovno o GIT-u, https://git-scm.com/about, pristupljeno: svibanj, 2017.

[4] Osnovno o XAMP-u, https://www.apachefriends.org/about.html, pristupljeno: svibanj, 2017.

[5] Povijesni pregled Java programskog jezika,
https://www.tecmint.com/what-is-java-a-brief-history-about-java/, pristupljeno: lipanj, 2017.

[6] Osnovne značajke Java programskog jezika,
https://www.javatpoint.com/features-of-java, pristupljeno: lipanj, 2017.

[7] Osnovno o PHP-u, http://php.net/manual/en/intro-whatis.php , pristupljeno: lipanj, 2017.

[8] Povijest Android operacijskog sustava,
https://www.androidcentral.com/android-history, pristupljeno: svibanj, 2017.

[9] Povijest Android operacijskog sustava po verzijama,
https://www.android.com/history/#/marshmallow, pristupljeno: svibanj, 2017.

[10] Osnovno o razvoju za Android,
https://developer.android.com/studio/features.html, pristupljeno: svibanj, 2017.

[11] Povijest razvoja Laravel *framework*-a, https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/, pristupljeno: rujan, 2017

[12] Povijest razvoja Laravel *framework*-a, https://medium.com/vehikl-news/a-brief-history-of-laravel-5d55970885bc, pristupljeno: rujan, 2017.

[13] *Retrofit* biblioteka,
http://square.github.io/retrofit/, pristupljeno: lipanj, 2017.

[14] *Dagger* biblioteka,
https://google.github.io/dagger/, pristupljeno: lipanj, 2017.

[15] *Butterknife* biblioteka,
http://jakewharton.github.io/butterknife/, pristupljeno: lipanj, 2017.

[16] *Stetho* biblioteka,
http://facebook.github.io/stetho/, pristupljeno: lipanj, 2017.

[17] Upoznavanje s REST API-em,
https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/, pristupljeno: lipanj, 2017.

[18] Tečaj za razvoj Laravel REST API-a,
https://www.udemy.com/laravel-5-php-framework-agile-and-practical-php-restful-api/learn/v4/content, pristupljeno: lipanj, 2017.

[19] *RecyclerView* biblioteka
https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html, pristupljeno: lipanj, 2017.

[20] *ListView*,
https://developer.android.com/guide/topics/ui/layout/listview.html, pristupljeno: lipanj, 2017.

[21] Razlike između *RecyclerView*-a i *ListView*-a,
https://android.jlelse.eu/recyclerview-listview-basic-comparison-91e844a2fbc4, pristupljeno: lipanj, 2017.

[22] *NavigationView*,
https://developer.android.com/reference/android/support/design/widget/NavigationView.html, pristupljeno: lipanj, 2017.

[23] Razlika između MVC, MVP I MVVM arhitekture,
https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/, pristupljeno: lipanj 2017.

[25] *AsyncTask,* https://developer.android.com/reference/android/os/AsyncTask.html, pristupljeno: rujan, 2017.

SAŽETAK

Cilj ovoga rada je omogućiti predškolskim psiholozima bolje praćenje napretka djece u vrtićima te njihovu integraciju u vrtićke skupine. Rezultat rješavanja ovog problema je izrađena mobilna aplikacija za Android sustav korištenjem Java programskog jezika. Aplikacija psiholozima pruža mogućnost bilježenja njihovih opservacija tijekom boravka u skupinama, pregleda nalaza, djece, skupine i vrtića te pretraživanja i uređivanja postojećih podataka. Na taj način psiholozi mogu uvijek znati u kakvu skupinu dolaze, koja djeca se tamo nalaze, imaju li već nekih opservacija za određenu djecu itd. Uz mobilnu aplikaciju izrađena je i API aplikacija korištenjem Laravel razvojnog okruženja i PHP programskog jezika koja omogućava mobilnoj aplikaciji komunikaciju s udaljenim poslužiteljem; primanje, slanje i ažuriranje podataka.

Ključne riječi: predškolski psiholog, opservacija, Android, Laravel, API, komunikacija

Android application for monitoring children growth and adaptation in kindergarten

Main goal of this master thesis is to enable preschool psychologist better tracking of children growth and adaptation in kindergarten groups. To resolve this issue, a mobile application for Android was created using the Java programming language. The application provides an easy way of writting and editing opservations during psychologist stay in kindergarten groups. It also enables findings, children, groups and kindergartens data search so that the psychologist can always prepare and know in which group he comes, what are the groups main issues, etc. Alongside the mobile application, an API application was also created using the Laravel framework and PHP programming language. The API enables the mobile application to communicate with the remote server and also to fetch, send and update data.

Keywords: preeschool psychologist, opservations, Android, Laravel, API, communication

ŽIVOTOPIS

Rastimir Orlić, rođen 28.12.1993. godine u Osijeku, nedugo nakon završetka osnovne škole završava i osnovnu glazbenu školu za klavir. Nakon završetka opće gimnazije u Valpovu, 2012. Godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Preddiplomski studij uspješno završava 2015. godine te stječe zvanje prvostupnika inženjera računarstva (univ.bacc.ing.comp.). Iste godine upisuje diplomski studij Programskog inženjerstva na istoimenom fakultetu u Osijeku. Slobodno vrijeme provodi svirajući, pjevajući, volontirajući u raznim udrugama od kojih valja spomenuti udrugu IEEE u kojoj je obnašao funkciju predsjedatelja računalnog odjeljenja IEEE studentskog ogranka u Osijeku te nastoji steći što više znanja i izvan fakulteta sudjelujući u natjecanjima kao što su IEEE Xtreme. 2016. godine upisuje edukaciju za web programera u PHP programskom jeziku pri ustanovi Edunova, Osijek. Iste godine uspješno završava edukaciju. Nedugo nakon toga dobiva praksu u jednom od prvih hrvavtskih start-up firmi – Farmeronu te ga oni nakon prakse ostavljaju na poziciji junior Android programera. Iako je firma zatvorena početkom 2016. godine, 2017. prelazi u firmu nastalu iz Farmerona – Gideon Brothers na poziciju Android programera. Od 2016. godine ponosan muž supruge Kristine, a od 2017. još ponosniji otac kćerke Rite.

PRILOZI (na CD-u)

Prilog 1. Pismeni oblik rada u .doc i .pdf formatu

Prilog 2. Programski kod provedenih testova u obliku projekta alata Visual Studio