

Aplikacija je namijenjena za Android mobile uređaje, te smo se potrudili da omogućimo kompatibilnost sa dalekom verzijom sustava zvanom Kitkat, odnosno verzijom 4.4.

### Općenito o korištenim tehnologijama i bibliotekama

MVP = model-view-presenter

- Derivat MVC (Model-View-Controller) architecture patterna
- Dijeli aplikaciju na tri glavna dijela – model, prikaz i prezenter
- Presenter se ponaša kao middle-man između modela i prikaza
- **Prikaz** (view) bi trebao biti prilično glup – samo uzima podatke (najčešće na temelju raznih događaja), šalje ih prenteru na obradu i od prezentera dobiva ono što bi trebao prikazati
- **Model** (model...) definira strukturu podataka koji se obrađuju u prenteru i prikazuju na prikazu
- **Prezenter** = business logic, u njemu se obrađuju podatci
- Prikaz i prezenter imaju svaki svoje interfaceove, odnosno apstraktne klase preko kojih komuniciraju, tj. komuniciraju preko funkcija deklariranih u tim klasama
- Na primjer - za prikaz logina, njemu pripada LoginView interface i pripadajuća LoginActivity konkretna implementacija LoginView-a
- Za logiku logina, koristimo interface LoginPresenter, i njegovu implementaciju LoginPresenterImpl koja implementira te funkcije
- Generalno, imamo ovakvu strukturu: NazivPresenter i NazivPresenterImpl, i ui/NazivView, NazivActivity

### MainActivity

- Za razliku od LoginActivity i SignupActivity, ovaj je uz pomoć PageAdapter-a i TabLayouta podijeljen na dva dijela – ustvari se fragmenti dodaju na MainActivity (svaki od njih je definiran zasebno kao uobičajeni activity, svaki sa svojim View interfaceom i Prezenterom)

### Google Maps API

- Lokacije, onLocationChanged, markeri, all that jazz
- Implementirano u main/lokacije/LokacijeFragment i svim pripadajućim ui/ostalim klasama
- Uz pomoć **LocationServices.getFusedLocationProviderClient** i **Looper**-a zahtjevamo u određenim intervalima lokaciju. „Pretplaćujemo“ se na onLocationResult() koji zove našu funkciju onLocationChanged(), uspoređujemo ga sa starom lokacijom i po potrebi u bazi osvježavamo lokaciju
- Sa mLocationRequest.setPriority() namješatamo prioritet preciznosti, tj. balansiramo količinu potrošene energije i preciznost

### Firestore / Firebase Auth / Firestore

- realtime baza podataka
- nudi eventove onDataChange što omogućuje lako prikazivanje drugih lokacija u stvarnom vremenu
- FB auth nudi razne mogućnosti autentifikacije (od kojih smo iskoristili email i anonimnu prijavu)
- Firestore nudi usluge spremanja datoteka i većih količina podataka (u našem slučaju, samo slika)
- Uobičajeno je FB brz, no ponekad zna biti neobično spor pri prijavi

## Mockito i JUnit (dio Espresso frameworka)

- framework za unit testing
- jos ovo trebamo konkretno implementirati (odnosno popraviti, jer smo počeli s tim, ali s obzirom da nije nužno.. well, it fell apart )

## Dagger2

- Googleova biblioteka za DI (dependency injection – injekciju zavisnosti)
- Omogućava lakši unit testing, jer se klasa odvajaju od zavisnosti na koje se oslanja – što bi reklo da se zavisnosti mogu mijenjati, bez da se mijenja klasa koja ih koristi
- Odnosno, moduli visoke razine, koji pružaju kompleksnu logiku, trebali bi biti lako ponovo iskoristitvi i na njih ne bi trebale utjecati promjene u modulima niže razine, koje samo pružaju pomoćne funkcionalnosti
- Na prvi pogled izgleda bespotrebno komplicirano, i jest ispočetka kompliciranije za složiti, ali *in the long run* olakšava cijelu stvar
- Umjesto da se preko konstruktora dodaju kod instanciranja klase, npr.

```
private LoginPresenter _presenter;  
public LoginActivity(LoginPresenter presenter) {  
    this._presenter = presenter;  
}
```

- Prezenter se umjesto toga, u View-ove injektiraju uz pomoć **@Inject** pribilješki, npr.

```
@Inject LoginPresenter<LoginView> presenter;
```

- I ovaj bi se dio mogao unaprijediti (odnosno proširiti i na Modele koje koristimo) jer baza za to je već implementirana

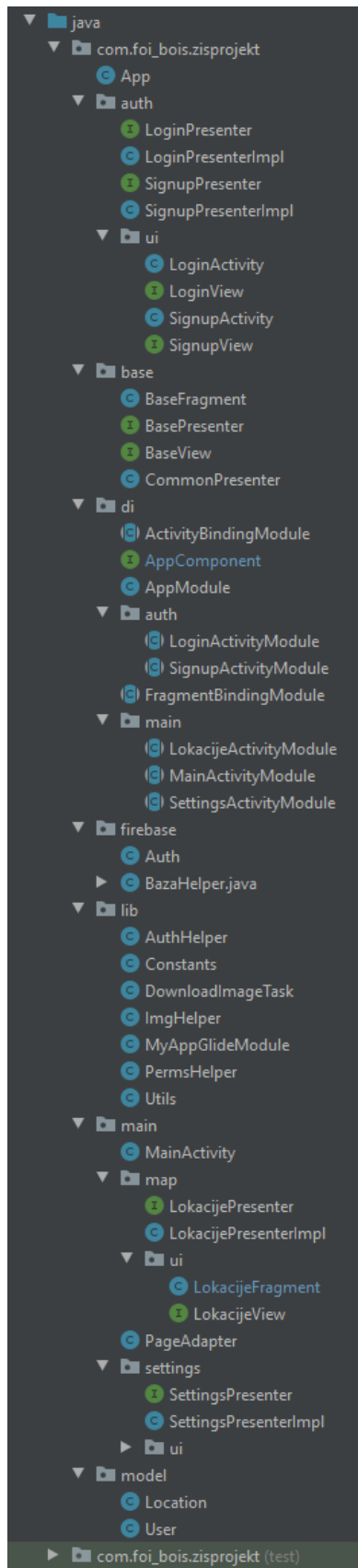
## Glide

- za prikaz i keširanje slika

## TastyToast

- Za ljepše Toastove (poruke na dnu ekrana) ☺

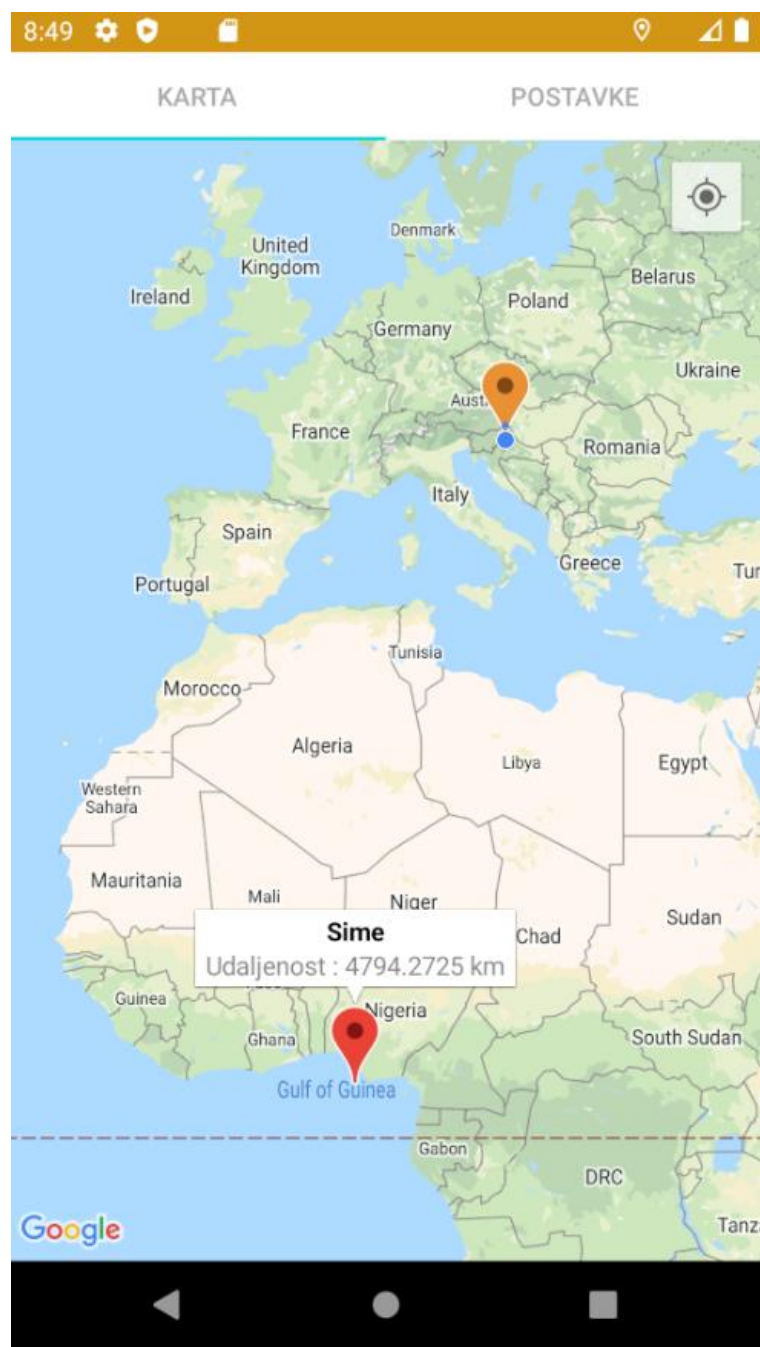
## Struktura projekta



*P.S. Koliko god da nema smisla, izgleda da volimo nenamjerno kombinirati engleski i hrvatski :)*

## OPIS FUNKCIONALNOSTI

### GLAVNI DIO – lokacije drugih instanci



Lokacija trenutnog korisnika je drugačije boje od ostalih instanci.

```

private void startLocationUpdate() {
    mLocationRequest = new LocationRequest();
    mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
    mLocationRequest.setInterval(INTERVAL);
    mLocationRequest.setFastestInterval(FATEST_INTERVAL);

    LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder();
    builder.addLocationRequest(mLocationRequest);
    LocationSettingsRequest locationSettingsRequest = builder.build();

    SettingsClient settingsClient = LocationServices.getSettingsClient(getActivity());
    settingsClient.checkLocationSettings(locationSettingsRequest);

    LocationServices.getFusedLocationProviderClient(getActivity()).requestLocationUpdates(mLocationRequest, new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            onLocationChanged(locationResult.getLastLocation(), showInfo: false);
        }
    },
    Looper.myLooper());
}

```

Osvježavanje lokacije za trenutnu instancu radimo uz pomoć **LocationRequest** API-ja i pripadajućih metoda. API izlaže onLocationResult callback te na temelju njega asinkrono dobivamo lokaciju i šaljemo u onLocationChanged.

```

private void onLocationChanged(android.location.Location newLocation, boolean showInfo) {
    if(oldLocation.getLatitude() == newLocation.getLatitude() && oldLocation.getLongitude() == newLocation.getLongitude())
        return;

    String msg = "Nova trenutna lokacija" + " : " +
        newLocation.getLatitude() + " , " + newLocation.getLongitude();
    if(showInfo)
        Toast.makeText(getActivity(), msg, Toast.LENGTH_SHORT).show();

    LatLng newPos = new LatLng(newLocation.getLatitude(), newLocation.getLongitude());
    if(myPosMarker != null) { //JIC
        myPosMarker.setPosition(newPos);

        BazaHelper.getInstance().refreshMyLocation(new Location(newPos), new BazaHelper.FirebaseUserCallback() {
            @Override
            public void onCallback(boolean isSuccessful) {
                if(isSuccessful){
                    Log.d(TAG, getResources().getString(R.string.Log_LocationSavingSuccess));
                }else{
                    Log.d(TAG, getResources().getString(R.string.Log_LocationSavingFail));
                }
            }
        });
    }

    oldLocation = newPos;
}

```

Ako se trenutna lokacija promijenila od one koje smo zadnji put zabilježili, mijenjamo je na karti (sa Marker.setPosition) i šaljemo te podatke u BazaHelper klasu (koja je Singleton, pa koristimo getInstance() ) te od nje čekamo asinkroni callback i bilježimo uspješnost promjene.

U klasi BazaHelper nalaze se ove dvije funkcije koje koristimo za osvježavanje naše lokacije.

```
public void refreshMyLocation(Location location, final FirebaseAuthCallback callback){
    refreshLocationForUser(FirebaseAuth.getInstance().getCurrentUser(), location, callback);
}

public void refreshLocationForUser(final FirebaseAuth user, Location location, final FirebaseAuthCallback callback){
    dbUserRef.child(user.getUid()).child("location").setValue(location)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                dbUserRef.child(user.getUid()).child("locationUpdated").setValue(ServerValue.TIMESTAMP).addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        callback.onCallback(task.isSuccessful());
                    }
                });
            }
        });
}
```

Prvo osvježimo lokaciju za korisnika s našim UUID-om, te kada taj proces završi, još promijenimo vrijeme kada je lokacija osvježena (za to koristimo ServerValue.TIMESTAMP koji vraća trenutno UNIX vrijeme)

Dopuštenja

```
private boolean checkLocationPerms() {
    return PermsHelper.checkPerms(getActivity(), ACCESS_FINE_LOCATION);
}

@Override
public void onMapReady(GoogleMap googleMap) {
    map = googleMap;

    if(checkLocationPerms()) {
        googleMap.setMyLocationEnabled(true);
    }
}
```

```
public class PermsHelper {
    public static boolean checkPerms(Activity activity, String perm) {
        if (ContextCompat.checkSelfPermission(activity, perm) ==
            PackageManager.PERMISSION_GRANTED) {
            return true;
        } else {
            requestPerms(activity, perm);
            return false;
        }
    }

    public static void requestPerms(Activity activity, String perm){
        ActivityCompat.requestPermissions(activity, new String[]{ perm }, requestCode: 1);
    }
}
```

Na početku provjeravamo imamo li dopuštenja za lokaciju uz pomoć klase PermsHelper i metode **checkPerms**, te ih po potrebi zahtjevamo uz pomoć **requestPerms**. Možemo birati između ACCESS\_COARSE\_LOCATION ako želimo što manje trošiti bateriju ili ACCESS\_FINE\_LOCATION ako želimo maksimalnu preciznost.

## OSVJEŽAVANJE DRUGIH LOKACIJA

```
@Override
public void loadOtherLocations() {
    final BazaHelper db = BazaHelper.getInstance();
    final long maxDiff = 60 * 1000;

    db.readAllUserData(new BazaHelper.FirebaseAllUserCallback() {
        @Override
        public void onCallback(HashMap<String, User> userDataList) {
            getView().clearMap();

            String mojId = FirebaseAuth.getInstance().getCurrentUser().getUid();

            long unixTime = System.currentTimeMillis() / 1000;

            for(String id : userDataList.keySet()){
                User userData = userDataList.get(id);

                if(id.equals(mojId))
                    getView().addMyLocation(userData.getLocation().getLatLng());
                else if (unixTime - userData.getLocationUpdated() <= maxDiff )
                    getView().addPositionMarker(userData.getLocation().getLatLng(), userData.getUsername());
            }
        }
    });
}
```

Za osvježavanje lokacija drugih korisnika, koristimo funkciju `loadOtherLocations()`, koja uz pomoć klase `BazaHelper` i pripadajućih `Callback` metoda, vadi sve ostale lokacije kao parove `String` (`id`) i `User`, pri čemu `User` sadrži lokaciju te vrijeme kada je ta lokacija zabilježena (u obliku UNIX vremena).

U slučaju da se `id` poklapa sa trenutnim korisnikom, dodajemo ga na kartu uz pomoć `addMyLocation`, inače provjeravamo je li razlika između trenutnog UNIX vremena i onog zapisanog u poljima unutar `User`-a, te ako je ta razlika manja od `maxDiff`, dodajemo ga na kartu.

## TESTNE LOKACIJE

```
@Override
public void createTestMarkers(boolean addOthers) {
    LatLng yourPos = new LatLng(45.8150, 15.9819); //Zagreb

    ArrayList<LatLng> testLocations = new ArrayList<>();
    testLocations.add(new LatLng(48.2082, 16.3738)); //Beč
    testLocations.add(new LatLng(48.8566, 2.3522)); //Paris

    getView().addMyLocation(yourPos);

    if(addOthers) {
        for(LatLng testPos : testLocations){
            getView().addPositionMarker(testPos, title: "Testna pozicija");
        }
    }
}
```

Za testiranje, koristili smo neke poznate lokacije, npr. Zagreb kao korisnikovu lokaciju, i listu s drugima koja se može promijeniti po potrebi.



## DAGGER

```
@Module
abstract class ActivityBindingModule {

    @ContributesAndroidInjector(modules = LoginActivityModule.class) //sve aktivnosti
    abstract LoginActivity loginActivity();

    @ContributesAndroidInjector(modules = SignupActivityModule.class)
    abstract SignupActivity signupActivity();

    @ContributesAndroidInjector(modules = MainActivityModule.class)
    abstract MainActivity mainActivity();
}
```

Sve se aktivnosti povezuju uz pomoć ActivityBindingModule i AppComponent.

```
@Singleton
@Component(modules = {
    AndroidSupportInjectionModule.class,
    ActivityBindingModule.class,
    FragmentBindingModule.class,
    AppModule.class})

public interface AppComponent extends AndroidInjector<App> {
    void inject(App application);

    @Component.Builder
    abstract class Builder extends AndroidInjector.Builder<App> { }
}
```

Kako bi se povezali prezenter i pogledi koriste se moduli poput LoginActivityModule ili SettingsActivityModule.

```
@Module
public abstract class LoginActivityModule {

    @Binds
    abstract LoginPresenter<LoginView> providePresenter(LoginPresenterImpl<LoginView> presenter);
}
```

```
@Module
public abstract class SettingsActivityModule {

    @Binds
    abstract SettingsPresenter<SettingsView> providePresenter(SettingsPresenterImpl<SettingsView> presenter);
}
```

Pogled se konkretno *injektira* u prezentera uz pomoć @Inject oznake, na primjer u slučaju prezentera za postavke:

```
public class SettingsPresenterImpl<V> extends SettingsView extends CommonPresenter<V> implements SettingsPresenter<V> {

    @Inject
    SettingsPresenterImpl(){}
}
```

## MODELI

```
public class User {
    //private String id; sifra, i sve ostalo je u FirebaseAuth!
    private String avatar; //trebala base64 bit, ipak je sad link na firestore
    private String username;
    private Location location;
    private long locationUpdated; //ServerValue.TIMESTAMP daje u ms razliku od UNIX vremena

    public User(){ //za JSON to POJO
    }

    public User( String username, String avatar, Location location, long locationUpdated){
        this.username = username;
        this.avatar = avatar;
        this.location = location;
        this.locationUpdated = locationUpdated;
    }

    public void setAvatar(String img) { this.avatar = img; }

    public String getAvatar() { return avatar; }

    public Location getLocation() { return location; }

    public void setLocation(Location location) { this.location = location; }

    public long getLocationUpdated() { return locationUpdated; }

    public void setLocationUpdated(long locationUpdated) { this.locationUpdated = locationUpdated; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    // public Date getLocationUpdated(){ return locationUpdated; }

    // public void setLocationUpdated(Date date) { this.locationUpdated = date; }
}
```

Klasa User sadrži attribute avatar (koji pokazuje na link na sliku spremljenu na Firestore-u), korisničko ime, tj. username, pripadnu instancu klase Location te vremensku oznaku kada je lokacija promijenjena. Polja poput id ili password ne držimo u ovoj klasi jer su ona već sadržana kao dio FirebaseAuth klase, odnosno singletona i njezinog getCurrentUser.

Nadalje imamo jedan prazni konstruktor (kako bi mogli koristiti JSON to POJO pretvorbu, umjesto da ručno pretvaramo sadržaj baze koja je u obliku JSON datoteke), te uobičajeni konstruktor sa svim atributima za inicijalizaciju klase.

```

public class Location {
    private double lat;
    private double lng;

    public Location() {
    } //treba i jedan prazan konstruktor za firebase

    public Location(double lat, double lng) {
        this.lat = lat;
        this.lng = lng;
    }

    public Location(LatLng latLng) {
        this.lat = latLng.latitude;
        this.lng = latLng.longitude;
    }

    public double getLng() { return lng; }

    public void setLng(double lng) { this.lng = lng; }

    public double getLat() { return lat; }

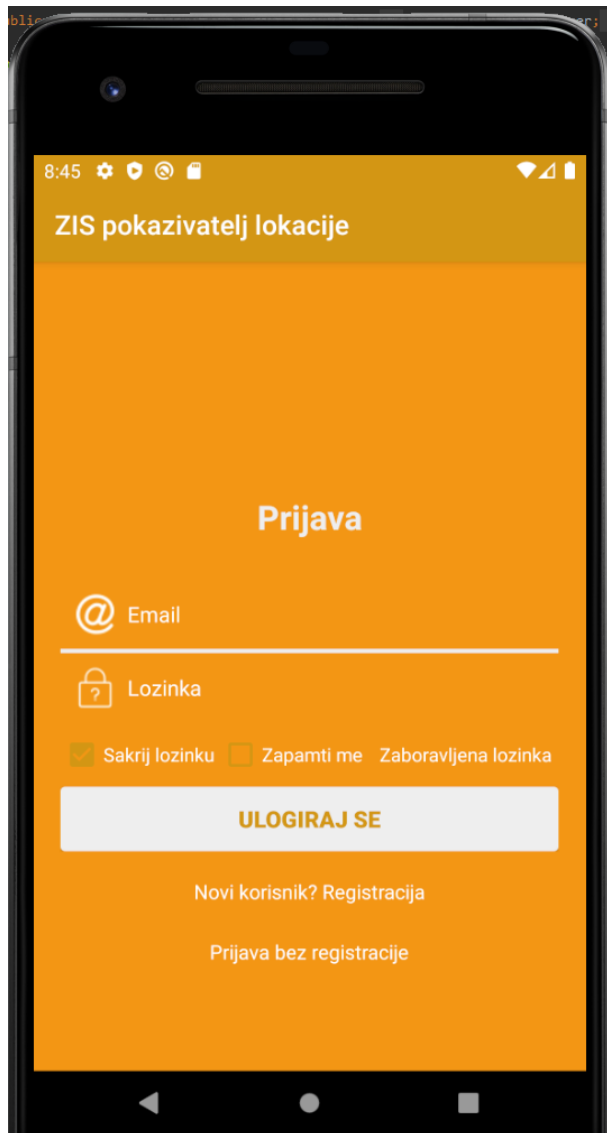
    public void setLat(double lat) { this.lat = lat; }

    @Exclude
    public LatLng getLatLng(){ return new LatLng(lat, lng); }
}

```

Prilično jednostavan model u kojeg spremamo lokaciju korisnika, odnosno par **širina-duljina**. Ponovo zbog JSON to POJO imamo jedan prazni i jedan uobičajeni konstruktor.

## AUTENTIFIKACIJA LOGIN



8:45

ZIS pokazivatelj lokacije

### Prijava

@ Email

Lozinka

☒ Sakrij lozinku ☐ Zapamti me [Zaboravljena lozinka](#)

**ULOGIRAJ SE**

[Novi korisnik? Registracija](#)

[Prijava bez registracije](#)

```

@Override
public void loginWithEmailPass(String email, String password) {
    if(email.length() == 0 || password.length() == 0)
        return;

    FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
        .addOnCompleteListener((task) -> {
            if (task.isSuccessful() && task.getResult() != null) {
                getView().onLoginResult( isSuccess: true, task.getResult().getUser());
            } else {
                getView().onLoginResult( isSuccess: false, user: null);
            }
        });
}

@Override
public void checkLogin(){ //ako je FB user null, onda nije ulogiran.. wowsies!
    getView().onCheckLogin( isLoggedIn: user == null);
}

@Override
public void loginAnon(){
    FirebaseAuth.getInstance().signInAnonymously()
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful() && task.getResult() != null) {
                    getView().onAnonLoginResult( isSuccess: true, task.getResult().getUser());
                } else {
                    getView().onAnonLoginResult( isSuccess: false, user: null);
                }
            }
        });
}

```

Korisnici se mogu prijaviti uz svoje postojeće korisničke podatke ili anonimno, a to nam omogućuje FirebaseAuth, sa svojim metodama `signInWithEmailAndPassword()` i `signInAnonymously()`. Na svaku od tih metoda može se dodati slušatelj, koji će asinkrono obavijestiti o uspješnosti prijave te pozvati metode `onLoginResult` odnosno `onAnonLoginResult` iz pripadajućeg pogleda.

## LOGIN POGLED

```
@Override
protected void onDestroy(){
    presenter.detach();
    super.onDestroy();
}

public void onSignupClick(View v) { startActivity(new Intent( packageContext: this, SignupActivity.class)); }

public void onLoginClick(View v){
    String email = tbUsername.getText().toString();
    String pass = tbPass.getText().toString();
    if (TextUtils.isEmpty(email) || TextUtils.isEmpty(pass)) {
        TastyToast.makeText( context: this, "Nesto fali!", TastyToast.LENGTH_SHORT, TastyToast.ERROR).show();
        return;
    }

    presenter.loginWithEmailPass(email, pass);
}

public void onAnonLoginClick(View v) { presenter.loginAnon(); }

@Override
public void onCheckLogin(boolean isLoggedIn){
    if(isLoggedIn)
        TastyToast.makeText( context: this, "Vec ste prijavljeni!", Toast.LENGTH_SHORT, TastyToast.INFO).show();
}

@Override
public void onLoginResult(boolean isSuccess, FirebaseUser user) {
    if(isSuccess){
        TastyToast.makeText( context: this, "Dobro dosli: " + user.getEmail(), TastyToast.LENGTH_SHORT, TastyToast.SUCCESS).show();
        startActivity(new Intent( packageContext: this, MainActivity.class));
    }
    else
        TastyToast.makeText( context: this, "Pepehands", TastyToast.LENGTH_SHORT, TastyToast.ERROR).show();
}

@Override
public void onAnonLoginResult(boolean isSuccess, FirebaseUser user) {
    if(isSuccess) {
        TastyToast.makeText( context: this, "Dobro dosli tko god da jeste", Toast.LENGTH_SHORT, TastyToast.INFO).show();
        startActivity(new Intent( packageContext: this, MainActivity.class));
    }
    else
        TastyToast.makeText( context: this, "Neuspjesna anonimna prijava", Toast.LENGTH_SHORT, TastyToast.ERROR).show();
}

private void onCheckShowPassClicked(View v){
    if(checkShowHidePass.isChecked())
        tbPass.setTransformationMethod(new PasswordTransformationMethod());
    else
        tbPass.setTransformationMethod(null);
}

private void onCheckRememberMeClicked(){
    if(checkRememberMe.isChecked())
        AuthHelper.saveLoginCredsToPerfs(getApplicationContext(), tbUsername.getText().toString(), tbPass.getText().toString());
    else
}
```

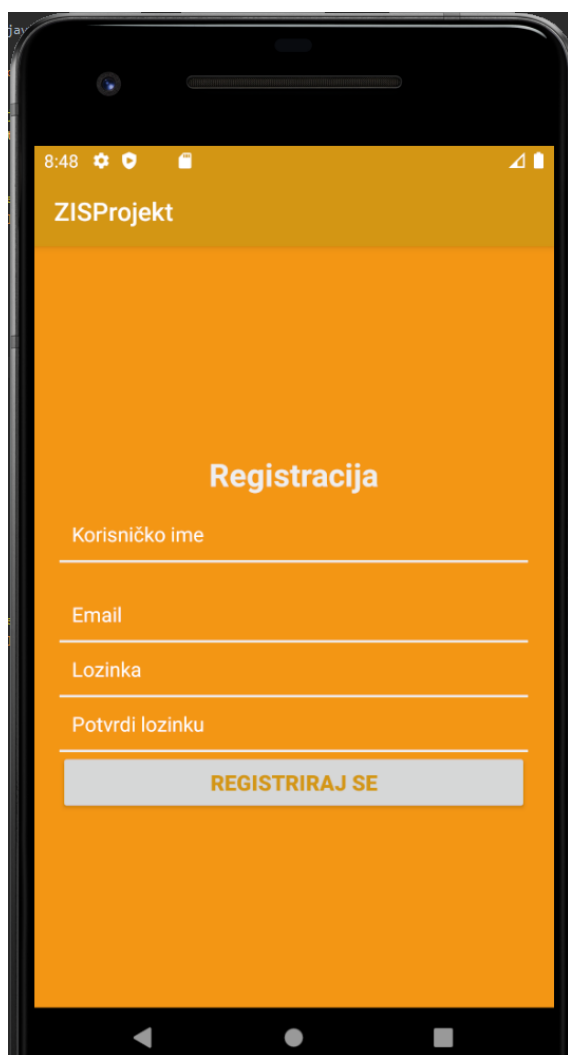
Kao što smo već rekli na početku, pogledi ne bi trebali provoditi nikakvo procesuiranje podataka, već samo prikazivati ono što mu prezenter šalje, odnosno slati prezenteru podatke na obradu. Tako i ovaj pogled, na primjer, kada se gumb za prijavu pritisne, šalje sa `onLoginClick()` ono što je korisnik upisao prema prezenteru, a kada je prezenter gotov s obradom, zove `onLoginResult` kako bi se prikazali rezultati prijave, odnosno korisnika odvelo na glavni dio aplikacije.

Aplikacija može i zapamtiti podatke s kojima se korisnik prijavio, te smo to implementirali u klasi AuthHelper.

```
public class AuthHelper {  
    public static boolean isValidEmail(String email) {  
        Pattern pattern = Patterns.EMAIL_ADDRESS;  
        return pattern.matcher(email).matches();  
    }  
  
    public static boolean isRememberMeChecked(Context context){  
        SharedPreferences prefs = context.getSharedPreferences(  
            "ZISProjekt_perfs", mode: 0);  
  
        return prefs.getBoolean( key: "rememberMeChecked", defValue: false);  
    }  
  
    public static void saveLoginCredsToPerfs(Context context, String email, String pass){  
        SharedPreferences prefs = context.getSharedPreferences(  
            "ZISProjekt_perfs", mode: 0);  
  
        SharedPreferences.Editor prefsEditor = prefs.edit();  
        prefsEditor.putBoolean("rememberMeChecked", true);  
        prefsEditor.putString("email", email);  
        prefsEditor.putString("pass", pass);  
        prefsEditor.apply();  
    }  
  
    public static void removeSavedLoginCreds(Context context){  
        SharedPreferences prefs = context.getSharedPreferences(  
            "ZISProjekt_perfs", mode: 0);  
  
        SharedPreferences.Editor prefsEditor = prefs.edit();  
        prefsEditor.putBoolean("rememberMeChecked", false);  
        prefsEditor.remove("email");  
        prefsEditor.remove("pass");  
        prefsEditor.apply();  
    }  
  
    public static Pair<String, String> getSavedLoginCreds(Context context){  
        SharedPreferences prefs = context.getSharedPreferences(  
            "ZISProjekt_perfs", mode: 0);  
  
        return new Pair<>(prefs.getString( key: "email", defValue: null), prefs.getString( key: "pass", defValue: null));  
    }  
}
```

SharedPreferences omogućava da na laki način spremamo koje god podatke želimo u memoriju telefona. Funkcija saveLoginCredsToPerfs sprema te podatke, dok ih removeSavedLoginCreds čita kako bi se automatski mogao korisnik prijaviti.

## SIGNUP



The image shows a smartphone screen with an orange background. At the top, the status bar shows the time 8:48, a gear icon, a shield icon, a battery icon, and a signal strength icon. Below the status bar, the app title "ZISProjekt" is displayed in white. The main heading "Registracija" is centered in white. Below the heading, there are four input fields with labels: "Korisničko ime", "Email", "Lozinka", and "Potvrdi lozinku". Each field has a white underline. At the bottom, there is a grey button with the text "REGISTRIRAJ SE" in orange. The bottom of the screen shows the Android navigation bar with a back arrow, a home circle, and a recent apps square.

8:48

ZISProjekt

### Registracija

Korisničko ime

Email

Lozinka

Potvrdi lozinku

REGISTRIRAJ SE



```

@Override
public void signUpWithEmail(final String username, String email, String password, String confirmPassword) {
    if(password.length() <= 5){
        getView().onSignUpResult( signUpResult: 2, user: null);
        return;
    }

    if(password.length() == 0){
        getView().onSignUpResult( signUpResult: 3, user: null);
        return;
    }

    if(!password.equals(confirmPassword)){
        getView().onSignUpResult( signUpResult: 4, user: null);
        return;
    }

    if(email.length() == 0){
        getView().onSignUpResult( signUpResult: 5, user: null);
        return;
    }

    if(!AuthHelper.isEmailValid(email)){
        getView().onSignUpResult( signUpResult: 6, user: null);
        return;
    }

    FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            public void onComplete(final @NonNull Task<AuthResult> signUpTask) {
                if (signUpTask.isSuccessful()){
                    UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                        .setDisplayName(username).build();

                    user = FirebaseAuth.getInstance().getCurrentUser();
                    user.updateProfile(profileUpdates).addOnCompleteListener(new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            if(task.isSuccessful()) {
                                getView().onSignUpResult( signUpResult: 0, signUpTask.getResult().getUser());
                                BazaHelper.getInstance().addDefaultFieldsForUser(user, username); //zbog praktičnih razloga, spremamo Username na obje lokacije
                            }
                            else {
                                getView().onSignUpResult( signUpResult: 1, user: null);
                            }
                        }
                    });
                }
                // else
                //     getView().onSignUpResult(1, null);
            }
        });
}

```

Nakon što korisnik na UI-ju pritisne gumb za prijavu, provjerava se validnost svih polja, te u slučaju da ništa nije pošlo po zlu, zove se funkcija createUserWithEmailAndPassword iz FirebaseAuth sa tim parametrima.

Na nju se dodaje onComplete slušatelj, u slučaju da je korektno izvršena prijava u FirebaseAuth, postavljamo username za korisnika uz pomoć UserProfileChangeRequest i setDisplayName funkcije.

Taj zahtjev povezujemo uz trenutnog korisnika tako da pozovemo updateProfile funkciju na koju dodajemo još jednog slušatelja kako bismo znali je li taj zahtjev prošao.

U slučaju da je uspješno proveden, dodajemo početne vrijednosti tako da pozovemo addDefaultFieldsForUser() iz BazaHelper klase.

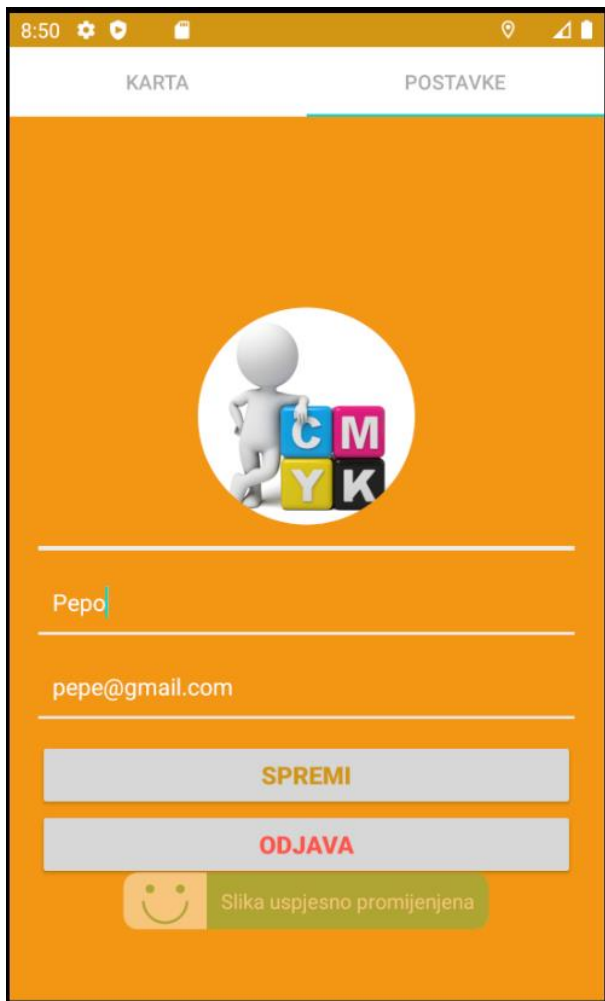
```

public void addDefaultFieldsForUser(final FirebaseAuth user, @Nullable String username){
    User customFields = new User(username, Constants.DEFAULT_AVATAR_URL, new Location( lat: 0.0, lng: 0.0), locationUpdated: 0); //locationUpdated 0 znači da bude postavljeno na 1.1.1970 ili tina sigurno prije "izlaska" aplikacije (
    dbUserRef.child(user.getUid()).setValue(customFields).addOnCompleteListener((task) -> {
        if(task.isSuccessful()){
            Log.d( tag: "BAZA", "Successfully saved in DB");
        }else{
            Log.d( tag: "BAZA", "Successfully saved in DB" + " for UID: " + user.getUid());
        }
    });
}

```

Ta funkcija dodaje defaultni avatar URL, postavlja lokaciju na (0.0, 0.0), te vrijeme kada je osvježena na 0 milisekundi.

## POSTAVKE



U postavkama korisnik može promijeniti svoje korisničko ime, te email.

```
@Override
public void saveCurrentUserSettings(final String email, final String username) {
    final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
    user.updateEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            UserProfileChangeRequest profileUpdates = new UserProfileChangeRequest.Builder()
                .setDisplayName(username).build();

            user.updateProfile(profileUpdates).addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if(!task.isSuccessful())
                        return;

                    BazaHelper.getInstance().setUsernameForUser(user, username, new BazaHelper.FirebaseUserCallback(){
                        @Override
                        public void onCallback(boolean isSuccessful) {
                            getView().onSettingsChanged(isSuccessful);
                        }
                    });
                }
            });
        }
    });
}
```

Na sličan način sa UserProfileChangeRequest kao što to implementirali za prijavu. (ponovo s puno ugniježđenih slušatelja ☹)

Također smo omogućili da korisnik doda sam svoj avatar. (iako smo imali nešto veće planove za to, trenutno nema nekakve praktične korisnosti, osim toga što si korisnik može malo personalizirati iskustvo)

Kada korisnik klikne na sliku, tj. avatar, provjeravaju se dopuštenja ( `checkFileParams()` ), te se korisnika upituje da na svom mobitelu pronađe sliku ( `loadAvatarImage()` ), uz pomoć Glide biblioteke prikazujemo tu sliku, te zovemo `uploadAvatarToFirestore()` iz prezentera da uploada sliku na Firestore.

```
private void onAvatarClick(View v){
    checkFilePerms();
    loadAvatarImage();
}

private void loadAvatarImage(){
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("image/*");

    startActivityForResult(intent, requestCode: 200);
}

public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 200) {
        Uri currFileURI = data.getData();

        if(currFileURI != null){
            String path = currFileURI.getPath();
            presenter.uploadAvatarToFirestore(currFileURI);

            Glide.with(getActivity()).load(currFileURI).placeholder(R.drawable.default_avatar).circleCrop().into(ivAvatar);
            // ivAvatar.setImageURI(currFileURI);
        }
    }
}

@Override
public void onAvatarUpload(boolean isSuccessful){
    if(isSuccessful)
        TastyToast.makeText(getActivity(), "Slika uspješno promijenjena", Toast.LENGTH_SHORT, TastyToast.SUCCESS).show();
    else
        TastyToast.makeText(getActivity(), "Neuspješna promjena slike", Toast.LENGTH_SHORT, TastyToast.ERROR).show();
}
```

```

@Override
public void uploadAvatarToFirestore(final Uri path) {
    BazaHelper baza = BazaHelper.getInstance();
    final FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();

    if(currentUser == null)
        return;

    baza.uploadAvatarToFirestore(currentUser, path, (isSuccessful, avatarUrl) -> {
        if (isSuccessful){
            BazaHelper.getInstance().setAvatarForUser(currentUser, avatarUrl, (isSuccessful) -> {
                getView().onAvatarUpload(isSuccessful);
            });
        }
    });
}

```

Ta metoda provjerava postoji li uopće trenutni korisnik, te zove uploadAvatarToFirestore iz BazaHelper kako bi se konačno avatar uploadao na Firestore pohranu.

```

public void uploadAvatarToFirestore(FirebaseUser user, Uri uri, final FirebaseUploadCallback callback){
    final StorageReference avatarRef = storageRef.getReference( location: "avatars").child(user.getUid());

    avatarRef.putFile(uri).addOnCompleteListener((task) -> { //malo uzasa jer je i getDownloadUrl Task :(
        if(task.isSuccessful()){
            avatarRef.getDownloadUrl().addOnSuccessListener((OnSuccessListener) (uri) -> {
                callback.onCallback( isSuccessful: true, uri.toString());
            });
        }
        else {
            callback.onCallback( isSuccessful: false, avatarUrl: "");
        }
    });
}

```

Uzimamo StorageReference instancu za mapu „avatars“, te referencu u njoj za našeg korisnika (prema njegovom UUID-u).

Uz pomoć putFile uploadamo na taj URL sliku i čekamo rezultate.

Konačno, kada učitavamo sliku za našeg korisnika, ne smijemo to raditi na glavnoj dretvi (ili ćemo izblokirati aplikaciju dok se ne završi skidanje), pa koristimo asinkroni zadatak zvan `DownloadImageTask`.

```
public DownloadImageTask(final Listener listener) { this.listener = listener; }

@Override
protected Bitmap doInBackground(String... urls) {
    final String url = urls[0];
    Bitmap bitmap = null;

    try {
        final InputStream inputStream = new URL(url).openStream();
        bitmap = BitmapFactory.decodeStream(inputStream);
    } catch (final MalformedURLException malformedUrlException) { }
    catch (final IOException ioException) { }
    return bitmap;
}

public interface Listener {
    void onImageDownloaded(final Bitmap bitmap);
    void onImageDownloadError();
}

@Override
protected void onPostExecute(Bitmap downloadedBitmap) {
    if (null != downloadedBitmap) {
        listener.onImageDownloaded(downloadedBitmap);
    } else {
        listener.onImageDownloadError();
    }
}
```