



ВТУ "Св. Св. Кирил и Методий"  
Факултет "Математика и Информатика"

КУРСОВА РАБОТА  
ПО ДИСЦИПЛИНАТА  
МОБИЛНИ ПРИЛОЖЕНИЯ ЗА ANDROID  
На тема:  
COMMUNITY BLOG

Изготвил:  
Пламенна Викторова Петрова  
Специалност:  
Софтуерно инженерство  
Факултетен № 1909011132

Проверил:  
доц. д-р.  
Златко Георгиев  
Върбанов

Велико Търново  
2021

**Изисквания и примерни теми**  
За изпит по „Мобилни приложения за Android“

**Спец. „софтуерно инженерство“, II курс**

**Изпитна сесия м.юни 2020 г.**

**Морски шах**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase)
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите.
- Различни нива на трудност.

**Бесеница**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase).
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.).
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите (ако се използва база данни).
- Различни нива на трудност (които може да се определят от дължините на думите) или избор между думи на български и думи на английски.

## **Органайзер (To-Do List)**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase) – опционално.
- Възможност за добавяне, редактиране и изтриване на задачи.
- Възможност за задаване срок на изпълнение на задачите.
- Възможност за задаване на приоритет и/или категория на задачите.
- Използване на Speech-to-Text (по преценка и Text-to-Speech).

## **Чат приложение (Messenger)**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase).
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самото приложение, да се използват функционалностите на Google Sign-in и т.н.)
- Възможност за изпращане на съобщения към група.
- Възможност за изпращане на индивидуални съобщения.
- Преглед/изтриване на история на изпратените съобщения (може да се изтриват автоматично след определен период от време)

## **Аудио/видео плейър**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase).
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самото приложение, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на най-слушаните песни/най-гледаните видеа (може и без база данни).
- Възможност за преглеждане статистиката на другите потребители (ако има база данни).
- Възможност да се избира от коя директория да се използват файловете.

## **Судоку**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase)
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите.
- Различни нива на трудност.

## **Bricks**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase)
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите.
- Различни нива на трудност.

## **Тетрис**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase)
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите.
- Различни нива на трудност.

## **Приложение за готварство**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase).
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.) – опционално. Ако тази функционалност се имплементира, може да се добави възможност за разглеждане на рецептите на други потребители.
- Възможност за добавяне, редактиране и изтриване на рецепти.
- Възможност за задаване на категория на дадена рецепта, добавяне в любими.
- Възможност за добавяне на списък с продукти за пазаруване.

## **Бикове и крави**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase)
- Логин: sign up, sign in, sign out (може да се регистрира потребител за самата игра, да се използват функционалностите на Google Sign-in и т.н.)
- Статистика на спечелени/загубени игри (да се запазват точки и т.н.; тази част би могла да се реализира и без използването на база данни).
- Класация, показваща мястото на текущия потребител спрямо останалите.
- Различни нива на трудност.

## **Речник (тълковен)**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да се използва база данни (локална или Firebase) – опционално.
- Възможност за добавяне, редактиране и изтриване на думи и техните значения.
- Възможност за определяне на категория или област на дадена дума, ако тя може да се използва като термин - в специфичната терминология в различни сфери.

- Приложението да разполага с първоначално множество от думи (речника да не е празен когато първоначално се стартира приложението).
- Използване на Speech-to-Tex (по преценка и Text-to-Speech за произнасяне на самата дума).

## **Общи**

- Да има повече от един екран (activity).
- Да се поддържа ротация на устройството (landscape, portrait).
- Да има някаква активност от страна на потребителя – приложението да не показва само статичен текст/картинки.
- Да се използва база данни (локална или Firebase) - опционално.
- Потребители и логин (според приложението – например за приложения от тип „To-Do List“ може да не е съвсем уместно).
- Да се обработват адекватно евентуални грешки и изключения.
- Да се използват Services, Broadcast receivers, Content providers, Notifications, ако е подходящо (ако приложението има функционалности, които биха ги използвали).

## Съдържание:

1. Цел и изпълнение на разработката.....	8
2. Основни етапи на разработката.....	11
2.1. Обновяване на build.gradle файловете.....	11
2.2. Създаване на RegisterActivity и activity_register.xml.....	13
2.3. Създаване на LoginActivity и activity_login.xml.....	17
2.4. Създаване на Navigation Drawer.....	18
2.5. Инициализация на Popup за добавяне на постове.....	20
2.6. Поява на нотификация при успешно създаден пост.....	22
2.7. Инициализация на RecyclerView за постове.....	22
2.8. Добавяне на PostDetailsActivity.....	25
2.9. Други особености.....	29
3. Визуализация на приложението.....	31

## 1. Цел и изпълнение на разработката

Цел на разработката е да се създаде модерно мобилно приложение, поддържащо няколко динамични activity-та както и landscape и portrait ориентация на устройството. Android проектът дава възможност за регистрация на нови потребители с потребителско име, имейл, парола и потребителска снимка. Полетата на регистрационната форма активират валидация при некоректно въведени данни или празни стойности. Credential-ите за регистриран потребител се запазват в Firebase Authentication – това са id-то на потребителя, имейл за идентификатор и времето на регистрация и последен login. Изборът на потребителска снимка се извършва чрез onClickListener на ImageView посредством шаблонна потребителска снимка, съхраняваща се в drawable папката. Задейства се специално дефиниран permission в AndroidManifest.xml – READ\_EXTERNAL\_STORAGE за достъп на файлове от външно хранилище, в случая при кликуване се отваря Google Downloads и трябва да се посочи една от свалените снимки. След регистрация избраната снимка се записва в Firebase Storage. Ако потребителят реши да не избере снимка при регистрация, той получава default-ната снимка към профила си. Логинът изисква имейла на user-а и паролата му, като login формата също валидира полетата си. Всъщност LoginActivity-то е зададено като launcher activity, т.е първото което потребителят вижда при отваряне на приложението е login формата. След успешен логин или регистрация се показва HomeActivity-то. То визуализира всички досега създадени постове от потребители, зарежда Navigation Drawer, състоящ се от три фрагмента – HomeFragment и статичните Profile и Settings и накрая logout бутон като последен menu item. HomeActivity-то инициализира и роруп при натискане на бутон с иконка за написване на нов пост. Роруп-ът работи с отделен layout xml файл, който се подава на Dialog класа с цел инстанциране на диалогов прозорец. Диалоговият прозорец съдържа две TextView полета за добавяне на заглавие и описание на пост и затъмен ImageView за избор на снимка за поста. Когато всички условия са покрити и няма грешки при валидация, при кликуване на icon button се добавя обект от тип Post в Realtime Database база данни на Firebase. За снимките на постове се взема FirebaseStorage инстанция и референция към наименувана child



папка, където се съхраняват с url-ите си. Същият подход се използва и при запис на потребителските снимки. При създаване на нов обект на конструктора се подават следните аргументи – заглавие на поста, подходящо описание, url-ът на изтеглената снимка, id-то на потребителя, който иска да събмитне поста, url-ът на потребителската му снимка и потребителското му име. За намирането на url-ите, id-то и потребителското име се ползват методи наготово от абстрактния клас `FirebaseUser` след изпълнение на метода `getCurrentUser()` от абстрактния клас `FirebaseAuth`. Останалите аргументи в конструктора са от тип `String`. За тях е нужно попълненият текст да бъде преведен към `String` с `toString()`. После се execute-ва `addPost(post)` Helper метод, където `post` е новосъздаден `Post` обект. Методът работи с инстанция на базата данни. Референцията може да се определи и добави на момента, ако базата не открива таблица, адресираща постове. За всеки прибавен пост се генерира уникален ключ чрез референцията, който в последствие се подава на setter от `Post` модела. Референцията set-ва като стойност `Post` обекта и при успех на екрана се появява `Toast` съобщение за успешно добавен пост. Тогава се build-ва и нотификация за конкретния контекст, използваща id на `notification channel`, който се регистрира от `NotificationManager`-а в `onCreate()` метода на `activity`-то чрез извикване на `private` метода `createNotificationChannel()`. Нотификацията показва малка `drawable` иконка, заглавието “New Post Added” и `content` - името на поста и `username`-а на потребителя, който е негов автор. За всяка нотификация се генерира уникално `Id`, така че при `n` на брой новодобавени публикации, потребителят получава толкова на брой нотификации. Когато един пост не може да се създаде успешно, се обработва изключение и се извежда `exception message`. Списъкът от постове се retrieve-ва от `Realtime Database`. За целта са използвани `PostAdapter` и `RecyclerView` в `home_fragment.xml` файла. В `PostAdapter` се inflate-а друг `xml` файл `row_post_item` в `onCreateViewHolder()` метода, който служи като `template` за всеки един `item` на `RecyclerView`-а. В метода `onBindViewHolder()` се обхождат и bind-ват всички `Post` обекти от `List`-а и в повтарящия се `template` на `RecyclerView` се задават коректни данни за постове чрез setter-и от класа на модела. Информацията се извлича от базата данни и се set-ва от `holder`-а. За снимките на поста и на потребителя се използва `Glide` библиотека, която

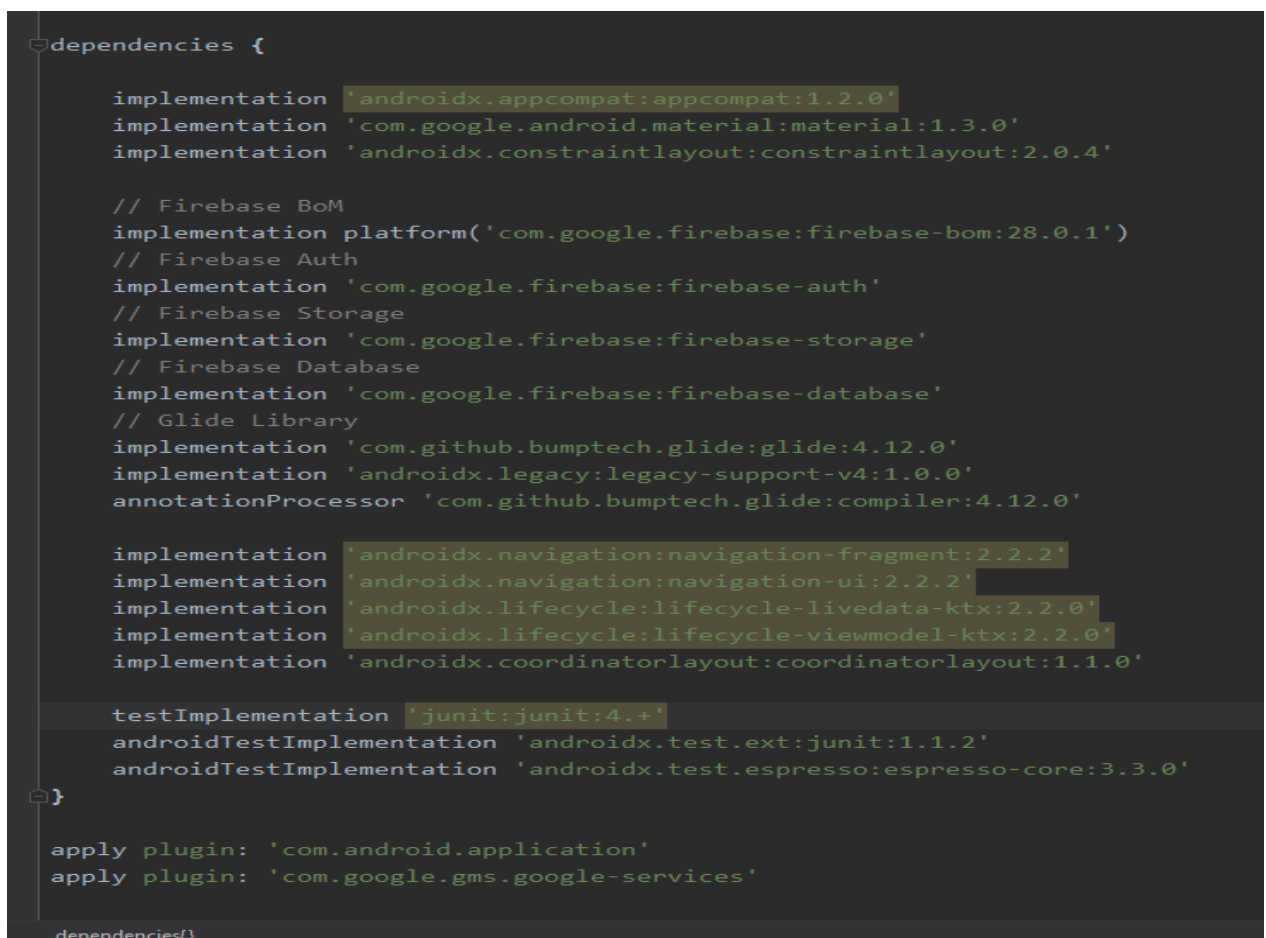
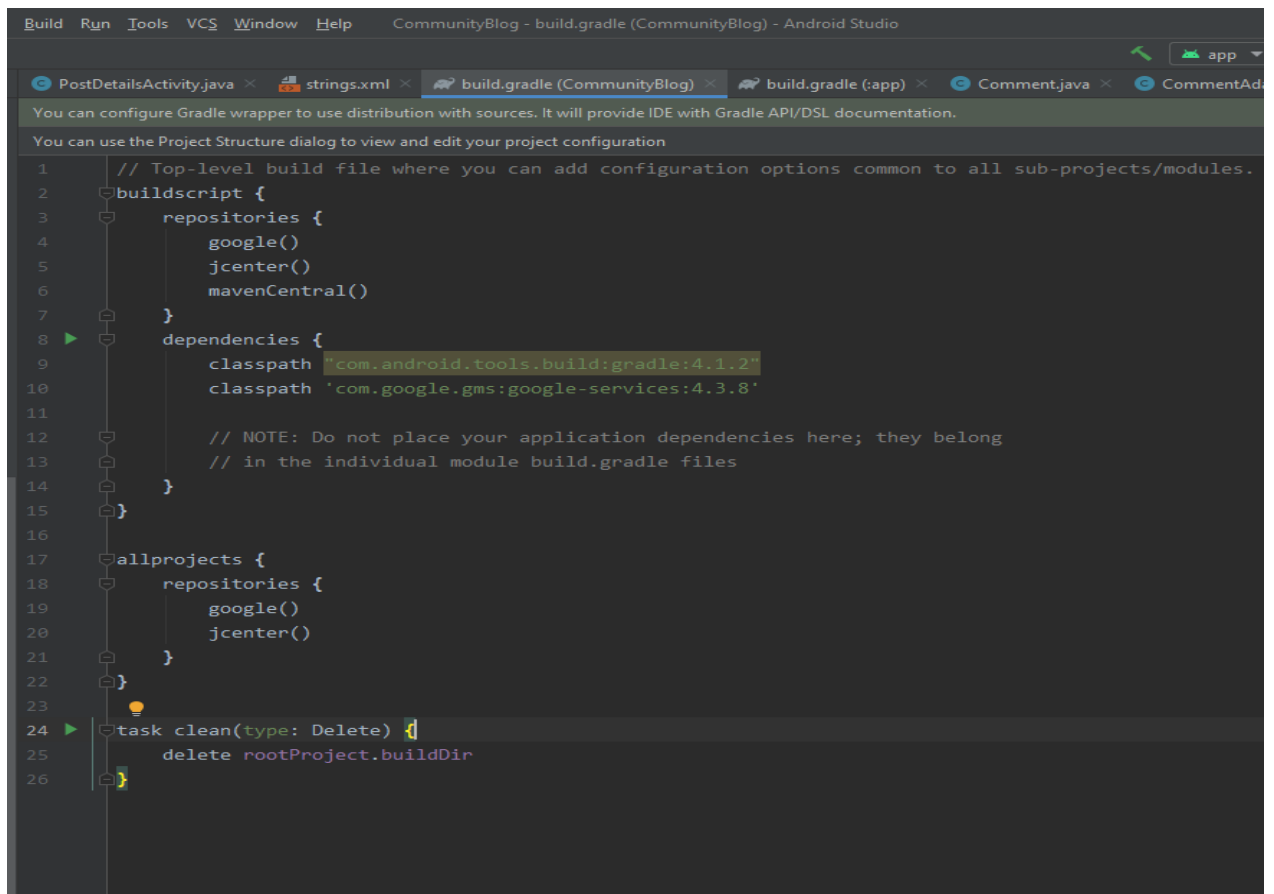
load-ва снимката чрез нейния url на местата на ImageView-тата. Извършва се проверка за избрана потребителска снимка. Ако потребителят е с default-ната, се зарежда drawable ресурса. В onCreateView() се активира RecyclerView-a като layout. Взема се и референция от Posts таблицата от базата. В onStart() метода става и самото добавяне на всеки item от RecyclerView-a. В addValueEventListener-a, закачен към database референцията се инстанцира ArrayList от Post обекти. Обхождат се DataSnapshot обектите и се вмъкват в списъка. Инстанцира се и PostAdapter-ът, като аргументи на конструктора му са конкретното activity и списъка от Post обекти. Следващата стъпка е да се set-не PostAdapter-a за RecyclerView. Визуализацията включва подредени снимките на постове една под друга. В долната част на всяка могат да се видят заглавието на публикацията, потребителското име на автора и потребителската му снимка. При кликването върху кой да е пост се активира onClickListener от PostAdapter-a. В listener-a се задава нов intent за PostDetailsActivity класа. PostDetailsActivity-то връща layout, insert-нат в NestedScrollView, със снимката на поста, неговото заглавие, датата на създаване, потребителското име на автора и потребителската му снимка. Освен това има и EditText за добавяне на нов коментар. До него се визуализира и снимката на current user-a. С бутона Add се добавя и нов коментар. Списъкът от коментари отново е реализиран чрез RecyclerView и адаптер за коментари. Още в PostAdapter-a цялата необходима информация за Post обектите се прикачва към intent-a за PostDetailsActivity-то с putExtra метода. Той работи с ключове от тип String и съответстващите им стойности, взети от gett-ерите на модела за полетата на обекти, намиращи се последователни позиции в списъка. Timestamp-овете, които не са параметри на Post конструктора се генерират автоматично, но трябва да се кастнат към long, защото са числови стойности. Следва стартиране на PostDetailsActivity-то. Там се вземат values за view-тата от xml файла по ключ, който се подава като аргумент на setContentView() метода в onCreate() метода на activity-то. За ImageView-тата пак се използват функционалностите на Glide библиотеката, на TextView стринговете се set-ват като текст, а timestamp-овете се конвертират в милисекунди посредством инстанция на абстрактния клас Calendar и се формират, така че да се показват деня, месеца и годината на създаване на

публикацията. Execute-ва се и метод за инициализация на RecyclerView за коментарите. В метода се задава Layout-а на RecyclerView-а. Взема се инстанция и референция към базата данни и се добавя нов запис с child element на таблицата Comments с ключа на post-а, за който е написан коментара. Ключът се достъпва от `getIntent().getExtras().getString("postKey")`, стойността на който е прибавен към intent-а за PostDetailsActivity-то още в PostAdapter-а. При натискане на AddComment бутона се активира `onClickListener`. Също както при постовете има изграден модел за коментари с конструктор с параметри съдържанието на коментара, id-то на коментиращия потребител, стринговата стойност на потребителската му снимка, а при условие че такава не е избрана още при регистрация, се работи с null, потребителското му име и автоматично генериран timestamp. Когато `onSuccessListener`-ът отчете, че е създаден нов обект от тип `Comment`, референцията към базата данни set-ва запис с информация за този обект. В противен случай на екрана се изписва Toast съобщение с грешката. `CommentAdapter`-ът извършва същата функция като `PostAdapter`-а. `Inflate`-ва се layout xml файл, който изпълнява ролята на template за единичен коментар. На всички view-та се подават прилежащите им стойности от базата. `Comment` обектите се вмъкват в празен `ArrayList` чрез `DataSnapshot`, инстанцира се `CommentAdapter`-ът спрямо контекста и списъка и на RecyclerView-а се set-ва адаптерът. Така целият List от коментари се визуализира успешно. В резултат може да се види снимката на коментиращия заедно с потребителското му име и съдържанието на коментара му и също в кой ден и в колко часа е изпратил коментара.

## **2. Основни етапи на разработката**

### **2.1. Обновяване на build.gradle файловете**

Приложението ползва услугите на Firebase и за да се осъществи връзка между двете страни, трябва да се добави `google-service.json`, който се изтегля при самата регистрация на проекта. В `build.gradle` за Project се закача `google()` в `repositories`, а за `module dependencies` за Firebase Bom, Firebase Auth, Firebase Storage, Firebase Database и Glide.



## 2.2. Създаване на RegisterActivity и activity\_register.xml

Използван е готов метод, предоставен от FirebaseAuth за регистрация на потребител. Той се изпълнява при попълване на валидни данни.

```
private void CreateUserAccount(String email, String name, String password)
{
    // this method creates user account with specific email and password

    mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: this, task -> {
            if (task.isSuccessful()) {
                // user account created successfully
                showMessage("Account created");
                // after we created user account we need to update the profile picture and name

                // now we check if the picked image is null or not
                if (pickedImgUri != null)
                {
                    updateUserInfo(name, pickedImgUri, mAuth.getCurrentUser());
                }
                else
                {
                    updateUserInfoWithoutPhoto(name, mAuth.getCurrentUser());
                }
            }
            else
            {
                // failed to create an account
                showMessage("Failed to create an account" + Objects.requireNonNull(task.getException()).getMessage());
                regBtn.setVisibility(View.VISIBLE);
                loadingProgress.setVisibility(View.INVISIBLE);
            }
        });
}
```

```
regBtn.setOnClickListener(view -> {
    regBtn.setVisibility(View.INVISIBLE);
    loadingProgress.setVisibility(View.VISIBLE);
    final String email = userEmail.getText().toString();
    final String password = userPassword.getText().toString();
    final String passwordToConfirm = userPasswordConfirmation.getText().toString();
    final String name = userName.getText().toString();

    if ( email.isEmpty() || name.isEmpty() || password.isEmpty() || !password.equals(passwordToConfirm) ) {
        // something goes wrong : all fields must be filled
        // we need to display an error message
        showMessage("Please Verify all fields");
        regBtn.setVisibility(View.VISIBLE);
        loadingProgress.setVisibility(View.INVISIBLE);
    }
    else
    {
        // everything is ok and all fields are filled now we can start creating user account
        // CreateUserAccount method will try to create the user if the email is valid
        CreateUserAccount(email, name, password);
    }
});
```

При избор на потребителска снимка и SDK версия  $\geq 22$  се задейства методът `checkAndRequestForPermission()`. В `Manifest.xml` файла четенето на файлове от външен storage вече е позволено. Заделят се специални request кодове за потребителските снимки и при кликване на default-ната снимка, user-ът трябва да се съгласи да даде достъп на изтеглените снимки до приложението. При по-малка SDK версия или липса на permission се отваря алтернативно галерията чрез gallery intent.

```
private void openGallery() {
    // TODO: open gallery intent and wait for user to pick an image !
    Intent galleryIntent = new Intent(Intent.ACTION_GET_CONTENT);
    galleryIntent.setType("image/*");
    startActivityForResult(galleryIntent, REQUESTCODE);
}

private void checkAndRequestForPermission() {
    if (ContextCompat.checkSelfPermission(context, RegisterActivity.this, Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
        if (ActivityCompat.shouldShowRequestPermissionRationale(activity, RegisterActivity.this, Manifest.permission.READ_EXTERNAL_STORAGE)) {
            Toast.makeText(context, RegisterActivity.this, "Please accept for required permission", Toast.LENGTH_SHORT).show();
        } else {
            ActivityCompat.requestPermissions(activity, RegisterActivity.this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, PReqCode);
        }
    }
    else
    {
        openGallery();
    }
}
```

```
ImgUserPhoto = findViewById(R.id.regUserPhoto);

ImgUserPhoto.setOnClickListener(view -> {
    if (Build.VERSION.SDK_INT >= 22) {
        checkAndRequestForPermission();
    }
    else
    {
        openGallery();
    }
});
}
```

След създаване на акаунт в Firebase информацията за потребителя се update-ва спрямо това дали той е избрал потребителска снимка или не. При първия случай снимката се съхранява в Firebase Storage, папка user-photos. Във втория снимката приема стойност null. Когато няма грешки в onSuccessListener на метода от FirebaseUser класа – updateProfile се слага onCompleteListener и се извиква метода updateUI, което означава че user-ът се redirect-ва към HomeActivity-то чрез intent.

```
// update user photo and name
private void updateUserInfo(final String name, Uri pickedImgUri, final FirebaseUser currentUser)
{
    // first we need to upload the user photo to the Firebase storage and get the url
    StorageReference mStorage = FirebaseStorage.getInstance().getReference().child("users_photos");
    StorageReference imagePath = mStorage.child(pickedImgUri.getLastPathSegment());

    imagePath.putFile(pickedImgUri).addOnSuccessListener(taskSnapshot -> {
        // image uploaded successfully
        // now we can get our image url

        imagePath.getDownloadUrl().addOnSuccessListener(uri -> {
            // uri contains the user image url
            UserProfileChangeRequest profileUpdate = new UserProfileChangeRequest.Builder()
                .setDisplayName(name)
                .setPhotoUri(uri)
                .build();

            currentUser.updateProfile(profileUpdate)
                .addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        // the user info is updated successfully
                        showMessage("The registration is completed");
                        updateUI();
                    }
                });
        });
    });
}
```

```
// update user photo and name
private void updateUserInfoWithoutPhoto(final String name, final FirebaseUser currentUser)
{
    UserProfileChangeRequest profileUpdate = new UserProfileChangeRequest.Builder()
        .setDisplayName(name)
        .build();

    currentUser.updateProfile(profileUpdate)
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                // the user info is updated successfully
                showMessage("The registration is completed");
                updateUI();
            }
        });
}
```

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode == RESULT_OK && requestCode == REQUESTCODE && data != null) {
        // the user has successfully picked an image
        // we need to save it's reference to an Uri variable
        pickedImgUri = data.getData();
        ImgUserPhoto.setImageURI(pickedImgUri);
    }
}

private void updateUI() {
    Intent homeActivity = new Intent(getApplicationContext(), HomeActivity.class);
    startActivity(homeActivity);
    finish();
}

```

Реализацията на activity\_register.xml файл става възможна благодарение на ConstraintLayout, т.е всеки компонент е ограничен спрямо други или вертикален или хоризонтален guideline. Този тип layout е интегриран и за останалите layout xml файлове.

Пример :

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Activities.LoginActivity"
    android:background="#FFFFFF">

    <ImageView
        android:id="@+id/login_photo"
        android:layout_width="90dp"
        android:layout_height="90dp"
        android:layout_marginTop="36dp"
        android:scaleType="fitXY"
        android:contentDescription="Sample User Photo"
        app:layout_constraintBottom_toTopOf="@+id/login_mail"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.048"
        app:srcCompat="@drawable/userphoto" />

    <EditText
        android:id="@+id/login_mail"
        android:layout_width="265dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="32dp"
        android:autoFillHints="Mail"
        android:background="@drawable/reg_edittext_style"
        android:ems="10"
        android:hint="Mail"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/guideline3"
        app:layout_constraintEnd_toEndOf="parent"

```



## 2.3. Създаване на LoginActivity и activity\_login.xml

Предпочетен е готов метод от Firebase за логин с имейл и парола. При кликване на dummy снимката потребителят се redirect-ва към регистрационната форма, т.е активира се Intent към RegisterActivity-то. Когато има успешен логин, user interface-ът на приложението се абонира за HomeActivity intent.

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login);

userMailLogin = findViewById(R.id.Login_mail);
userPasswordLogin = findViewById(R.id.Login_password);
btnLogin = findViewById(R.id.Login_btn);
loginProgress = findViewById(R.id.Login_progress);
mAuth = FirebaseAuth.getInstance();
HomeActivity = new Intent( packageContext, this, com.example.communityblog.Activities.HomeActivity.class);
loginPhoto = findViewById(R.id.Login_photo);

loginPhoto.setOnClickListener(view -> {
    Intent registerActivity = new Intent(getApplicationContext(), RegisterActivity.class);
    startActivity(registerActivity);
    finish();
});

loginProgress.setVisibility(View.INVISIBLE);

btnLogin.setOnClickListener(view -> {
    loginProgress.setVisibility(View.VISIBLE);
    btnLogin.setVisibility(View.INVISIBLE);

    final String loginMail = userMailLogin.getText().toString();
    final String loginPassword = userPasswordLogin.getText().toString();

    if (loginMail.isEmpty() || loginPassword.isEmpty()) {
        showLoginMessage( text: "Please Verify All Fields");
        btnLogin.setVisibility(View.VISIBLE);
        loginProgress.setVisibility(View.INVISIBLE);
    }
    else
    {
        signIn(loginMail, loginPassword);
    }
});
}
```

```
private void signIn(String mail, String password)
{
    mAuth.signInWithEmailAndPassword(mail, password)
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                loginProgress.setVisibility(View.INVISIBLE);
                btnLogin.setVisibility(View.VISIBLE);
                updateUI();
            }
            else
            {
                showLoginMessage(Objects.requireNonNull(task.getException()).getMessage());
                btnLogin.setVisibility(View.VISIBLE);
                loginProgress.setVisibility(View.INVISIBLE);
            }
        });
}
```

## 2.4. Създаване на NavigationDrawer

Инициализиран е default-ният NavigationDrawer. Той представлява sidebar, който може да се разпъва и съдържа снимката на потребителя, username, имейла на потребителя и меню елементи – отделни фрагменти и не на последно място logout бутон.

```
@Override
public void onBackPressed()
{
    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.home, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the HomeActivity/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.

    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

```

/StatementWithEmptyBody, RedundantSuppression/
@Override
public boolean onNavigationItemSelected(MenuItem item)
{
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_home) {
        Objects.requireNonNull(getSupportActionBar()).setTitle("Home");
        getSupportFragmentManager().beginTransaction().replace(R.id.container, new HomeFragment()).commit();
    }
    else if (id == R.id.nav_profile) {
        Objects.requireNonNull(getSupportActionBar()).setTitle("Profile");
        getSupportFragmentManager().beginTransaction().replace(R.id.container, new ProfileFragment()).commit();
    }
    else if (id == R.id.nav_settings) {
        Objects.requireNonNull(getSupportActionBar()).setTitle("Settings");
        getSupportFragmentManager().beginTransaction().replace(R.id.container, new SettingsFragment()).commit();
    }
    else if (id == R.id.nav_signout) {
        FirebaseAuth.getInstance().signOut();
        Intent loginActivity = new Intent(getApplicationContext(), LoginActivity.class);
        startActivity(loginActivity);
        finish();
    }

    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}

```

## Update-ване на sidebar-a

```

public void updateNavHeader()
{
    NavigationView navigationView = findViewById(R.id.nav_view);
    View headerView = navigationView.getHeaderView(0);
    TextView navUsername = headerView.findViewById(R.id.nav_username);
    TextView navUserMail = headerView.findViewById(R.id.nav_user_mail);
    ImageView navUserPhoto = headerView.findViewById(R.id.nav_user_photo);

    navUserMail.setText(currentUser.getEmail());
    navUsername.setText(currentUser.getDisplayName());

    // now we will use Glide to load user image
    // first we need to import the library

    if (currentUser.getPhotoUrl() != null)
    {
        Glide.with( activity, this).load(currentUser.getPhotoUrl()).into(navUserPhoto);
    }
    else
    {
        Glide.with( activity, this).load(R.drawable.userphoto).into(navUserPhoto);
    }
}

```

## 2.5. Инициализация на Роруп за добавяне на постове

```
CommunityBlog - HomeActivity.java [CommunityBlog.app] - Android Studio
CommunityBlog > Activities > HomeActivity
HomeActivity.java x LoginActivity.java x HomeActivity.java x activity_home.xml x activity_home_drawer.xml x fragment_home.xml x fragment_profile.xml x

private void iniPopup()
{
    popupAddPost = new Dialog( context: this);
    popupAddPost.setContentView(R.layout.popup_add_post);
    popupAddPost.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
    popupAddPost.getWindow().setLayout(Toolbar.LayoutParams.MATCH_PARENT, Toolbar.LayoutParams.WRAP_CONTENT);
    popupAddPost.getWindow().getAttributes().gravity = Gravity.TOP;

    // ini popup widgets
    popupUserImage = popupAddPost.findViewById(R.id.popup_user_image);
    popupPostImage = popupAddPost.findViewById(R.id.popup_img);
    popupTitle = popupAddPost.findViewById(R.id.popup_title);
    popupDescription = popupAddPost.findViewById(R.id.popup_description);
    popupAddBtn = popupAddPost.findViewById(R.id.popup_add);
    popupClickProgress = popupAddPost.findViewById(R.id.popup_progressBar);

    // load current user profile photo
    if (currentUser.getPhotoUrl() != null)
    {
        Glide.with( activity: HomeActivity.this).load(currentUser.getPhotoUrl()).into(popupUserImage);
    }
    else
    {
        Glide.with( activity: HomeActivity.this).load(R.drawable.userphoto).into(popupUserImage);
    }
}
```

```
if (!popupTitle.getText().toString().isEmpty() && !popupDescription.getText().toString().isEmpty() && pickedImgUri != null) {
    //everything is okay, no empty or null values
    // Creating Post Object and adding it to Firebase Realtime Database
    // first we need to upload the post image
    // access firebase storage
    StorageReference storageReference = FirebaseStorage.getInstance().getReference().child("community_blog_images");
    StorageReference imageFilePath = storageReference.child(pickedImgUri.getLastPathSegment());

    imageFilePath.putFile(pickedImgUri).addOnSuccessListener(taskSnapshot -> imageFilePath.getDownloadUrl().addOnSuccessListener(uri -> {
        String imageDownloadLink = uri.toString();
        // create post Object

        if (currentUser.getPhotoUrl() != null)
        {
            Post post = new Post(popupTitle.getText().toString(), popupDescription.getText().toString(), imageDownloadLink, currentUser.getUid(), currentUser.getPhotoUrl());
            // Add post to Firebase Database
            addPost(post);
            // Add a notification that a new post has been created
            setUpPostCreatedNotification(post);
        }
        else
        {
            Post post = new Post(popupTitle.getText().toString(), popupDescription.getText().toString(), imageDownloadLink, currentUser.getUid(), userPhoto);
            // Add post to Firebase Database
            addPost(post);
            // Add a notification that a new post has been created
            setUpPostCreatedNotification(post);
        }
    })).addOnFailureListener(exception -> {
        // something goes wrong when uploading the picture
        showMessage(exception.getMessage());
        popupClickProgress.setVisibility(View.INVISIBLE);
        popupAddBtn.setVisibility(View.VISIBLE);
    }));
}
```

## Добавяне на пост към базата

```
CommunityBlog - Post.java [CommunityBlog.app] - Android Studio
CommunityBlog > Models > Post
PostAdapter.java x HomeActivity.java x activity_home_drawer.xml x fragment_home.xml x fragment_profile.xml x app_bar_home.xml x PostAdapter.java x
package com.example.communityblog.Models;

import com.google.firebase.database.ServerValue;

public class Post {

    private String title;
    private String description;
    private String picture;
    private String userId;
    private String userPhoto;
    private String userName;
    private Object timeStamp;
    private String postKey;

    public Post(String title, String description, String picture, String userId, String userPhoto, String userName)
    {
        this.title = title;
        this.description = description;
        this.picture = picture;
        this.userId = userId;
        this.userPhoto = userPhoto;
        this.userName = userName;
        this.timeStamp = ServerValue.TIMESTAMP;
    }

    public Post()
    {
    }

    public String getTitle() { return title; }

    public String getDescription() { return description; }

    public String getPicture() { return picture; }
```

```
private void addPost(Post post)
{
    FirebaseDatabase database = FirebaseDatabase.getInstance("https://communityblog-e2892-default-rtdb.europe-west1.firebaseio.com/");
    DatabaseReference myRef = database.getReference("posts").push();

    // get post unique ID and update post key
    String key = myRef.getKey();
    post.setPostKey(key);

    // add post data to firebase database
    myRef.setValue(post).addOnSuccessListener(aVoid -> {
        showMessage("Post Added successfully");
        popupClickProgress.setVisibility(View.INVISIBLE);
        popupAddBtn.setVisibility(View.VISIBLE);
        popupAddPost.dismiss();
    });
}
```

## 2.6. Поява на нотификация при успешно създаден пост

```
private void createNotificationChannel()
{
    // Create the NotificationChannel, but only on API 26+ because
    // the NotificationChannel class is new and not in the support library
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = "Test Notification";
        String description = "Test Notification Description";
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel( id: "community-blog-test", name, importance);
        channel.setDescription(description);
        NotificationManager notificationManager = getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }
}

private void setUpPostCreatedNotification(Post post)
{
    NotificationCompat.Builder builder = new NotificationCompat.Builder( context: this, channelId: "community-blog-test")
        .setSmallIcon(R.drawable.ic_notification)
        .setContentTitle("New Post Added")
        .setContentText(post.getTitle() + " by " + post.getUserName())
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
    // Generating a random id
    int randomId = (int) ((new Date().getTime() / 1000L) % Integer.MAX_VALUE);
    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
    notificationManager.notify(randomId, builder.build());
}
```

## 2.7. Инициализация на RecyclerView за постове

В PostAdapter-a :

```

tools  VCS  Window  Help  CommunityBlog - PostAdapter.java [CommunityBlog.app] - Android Studio
CommunityBlog > Adapters > PostAdapter
PostAdapter.java  HomeActivity.java  activity_home_drawer.xml  fragment_home.xml  fragment_profile.xml  app_bar_home.xml  PostAdapter.java

public class PostAdapter extends RecyclerView.Adapter<PostAdapter.MyViewHolder>

{
    Context mContext;
    List<Post> mData;

    public PostAdapter(Context mContext, List<Post> mData)
    {
        this.mContext = mContext;
        this.mData = mData;
    }

    @NonNull
    @Override
    public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
    {
        View row = LayoutInflater.from(mContext).inflate(R.layout.row_post_item, parent, attachToRoot false);
        return new MyViewHolder(row);
    }

    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int position)
    {
        holder.postTitle.setText(mData.get(position).getTitle());
        Glide.with(mContext).load(mData.get(position).getPicture()).into(holder.imgPost);

        String userImg = mData.get(position).getUserPhoto();
        if (userImg != null)
        {
            Glide.with(mContext).load(userImg).into(holder.imgPostProfile);
        }
        else
        {
            Glide.with(mContext).load(R.drawable.userphoto).into(holder.imgPostProfile);
        }

        holder.postUsername.setText(mData.get(position).getUserName());
    }
}

```

```

@Override
public int getItemCount() { return mData.size(); }

public class MyViewHolder extends RecyclerView.ViewHolder
{
    TextView postTitle;
    ImageView imgPost;
    ImageView imgPostProfile;
    TextView postUsername;

    public MyViewHolder(View itemView)
    {
        super(itemView);

        postTitle = itemView.findViewById(R.id.row_post_title);
        imgPost = itemView.findViewById(R.id.row_post_image);
        imgPostProfile = itemView.findViewById(R.id.row_post_profile_img);
        postUsername = itemView.findViewById(R.id.row_post_username);

        itemView.setOnClickListener(view -> {
            Intent postDetailsActivityIntent = new Intent(mContext, PostDetailsActivity.class);
            int position = getAdapterPosition();
            postDetailsActivityIntent.putExtra( name: "title", mData.get(position).getTitle());
            postDetailsActivityIntent.putExtra( name: "postImage", mData.get(position).getPicture());
            postDetailsActivityIntent.putExtra( name: "description", mData.get(position).getDescription());
            postDetailsActivityIntent.putExtra( name: "postKey", mData.get(position).getPostKey());
            postDetailsActivityIntent.putExtra( name: "userPhoto", mData.get(position).getUserPhoto());
            postDetailsActivityIntent.putExtra( name: "userName", mData.get(position).getUserName());
            // user name not added to post object
            postDetailsActivityIntent.putExtra("userName", mData.get(position).getUserName());
            long timestamp = (long) mData.get(position).getTimeStamp();
            postDetailsActivityIntent.putExtra( name: "postDetailsDate", timestamp);
            mContext.startActivity(postDetailsActivityIntent);
        });
    }
}

```

## B HomeFragment-a :

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState)
{
    // Inflate the layout for this fragment
    View fragmentView = inflater.inflate(R.layout.fragment_home, container, attachToRoot: false);
    postRecyclerView = fragmentView.findViewById(R.id.postRecyclerView);
    postRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
    postRecyclerView.setHasFixedSize(true);
    firebaseDatabase = FirebaseDatabase.getInstance("https://communityblog-e2892-default-rtdb.europe-west1.firebaseio.com/");
    databaseReference = firebaseDatabase.getReference("Posts");
    return fragmentView;
}

@Override
public void onStart() {
    super.onStart();

    // Get Posts List from the database
    databaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            postList = new ArrayList<>();
            for (DataSnapshot postSnapshot : dataSnapshot.getChildren()) {
                Post post = postSnapshot.getValue(Post.class);
                postList.add(post);
            }

            postAdapter = new PostAdapter(getActivity(), postList);
            postRecyclerView.setAdapter(postAdapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}
```

Изпълнение на методите и показване на резултата от RecyclerView-то в HomeActivity-то :



```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    // ini

    mAuth = FirebaseAuth.getInstance();
    currentUser = mAuth.getCurrentUser();

    // ini popup
    iniPopup();
    setupPopupImageClick();

    FloatingActionButton fab = findViewById(R.id.fab);
    fab.setOnClickListener(view ->
    {
        popupAddPost.show();
    });

    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle( activity, this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
    drawer.addDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);

    updateNavHeader();
}

// set the home fragment as the default done

getSupportFragmentManager().beginTransaction().replace(R.id.container, new HomeFragment()).commit();

createNotificationChannel();
}

```

## 2.8. Добавяне на PostDetailsActivity

Вземане на `getExtras()` стойности по ключ, дефинирани в `PostAdapter`-а и задаването им по `view`-тата:

```

String postTitle = getIntent().getExtras().getString( key: "title");
textPostTitle.setText(postTitle);

String userPostImage = getIntent().getExtras().getString( key: "userPhoto");

if (userPostImage != null)
{
    Glide.with( activity: this).load(userPostImage).into(imgUserPost);
}
else
{
    Glide.with( activity: this).load(R.drawable.userphoto).into(imgUserPost);
}

String postDescription = getIntent().getExtras().getString( key: "description");
textPostDescription.setText(postDescription);

// set comment user image
if (firebaseUser.getPhotoUrl() != null)
{
    Glide.with( activity: this).load(firebaseUser.getPhotoUrl()).into(imgCurrentUser);
}
else
{
    Glide.with( activity: this).load(R.drawable.userphoto).into(imgCurrentUser);
}

// get post id
postKey = getIntent().getExtras().getString( key: "postKey");

long date = getIntent().getExtras().getLong( key: "postDetailsDate");
textPostDateName.setText(timestampToString(date));

String postUsername = getIntent().getExtras().getString( key: "userName");
textPostUsername.setText(postUsername);

```

Записване на коментар в базата данни :

```

firebaseAuth = FirebaseAuth.getInstance();
firebaseUser = firebaseAuth.getCurrentUser();
firebaseDatabase = FirebaseDatabase.getInstance("https://communityblog-e2892-default-rtdb.europe-west1.firebaseio.com/");

// add Comment button listener
btnAddComment.setOnClickListener(v -> {

    btnAddComment.setVisibility(View.INVISIBLE);

    DatabaseReference commentReference = firebaseDatabase.getReference( path: "Comments").child(postKey).push();
    String commentContent = editTextComment.getText().toString();
    String uId = firebaseUser.getUid();
    String username = firebaseUser.getDisplayName();
    String uImg = firebaseUser.getPhotoUrl() != null ? firebaseUser.getPhotoUrl().toString() : null;
    Comment comment = new Comment(commentContent, uId, uImg, username);
    commentReference.setValue(comment).addOnSuccessListener(aVoid -> {
        showMessage("Comment added");
        editTextComment.setText("");
        btnAddComment.setVisibility(View.VISIBLE);
    }).addOnFailureListener(exception -> showMessage("Failed to add comment" +exception.getMessage()));
});

```

```

package com.example.communityblog.Models;

import com.google.firebase.database.ServerValue;

public class Comment
{
    private String content;
    private String userId;
    private String userImg;
    private String userName;

    private int defaultPhoto;

    private Object timestamp;

    public Comment()
    {
    }

    public Comment(String content, String userId, String userImg, String userName)
    {
        this.content = content;
        this.userId = userId;
        this.userImg = userImg;
        this.userName = userName;
        this.timestamp = ServerValue.TIMESTAMP;
    }

    public Comment(String content, String userId, String userImg, String userName, Object timestamp)
    {
        this.content = content;
        this.userId = userId;
        this.userImg = userImg;
        this.userName = userName;
        this.timestamp = timestamp;
    }
}

```

Инициализация на RecyclerView за коментари :

```

private void iniRecyclerViewComments()
{
    recyclerViewComments.setLayoutManager(new LinearLayoutManager( context: this));
    firebaseDatabase = FirebaseDatabase.getInstance("https://communityblog-e2892-default-rtdb.europe-west1.firebaseio.com/");
    DatabaseReference recyclerViewCommentsReference = firebaseDatabase.getReference( path: "Comments").child(postKey);
    recyclerViewCommentsReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            commentsList = new ArrayList<>();
            for (DataSnapshot userCommentSnapshot: dataSnapshot.getChildren())
            {
                Comment userComment = userCommentSnapshot.getValue(Comment.class);
                commentsList.add(userComment);
            }
            commentAdapter = new CommentAdapter(getApplicationContext(), commentsList);
            recyclerViewComments.setAdapter(commentAdapter);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}

```

## И в CommentAdapter-a

```
private Context mContext;
private List<Comment> mData;

public CommentAdapter(Context mContext, List<Comment> mData)
{
    this.mContext = mContext;
    this.mData = mData;
}

@NonNull
@Override
public CommentViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
    View row = LayoutInflater.from(mContext).inflate(R.layout.row_comment, parent, attachToRoot: false);
    return new CommentViewHolder(row);
}

@Override
public void onBindViewHolder(@NonNull CommentViewHolder holder, int position)
{
    String uImg = mData.get(position).getUserImg();
    if (uImg != null)
    {
        Glide.with(mContext).load(mData.get(position).getUserImg()).into(holder.imgUser);
    }
    else
    {
        Glide.with(mContext).load(R.drawable.userphoto).into(holder.imgUser);
    }

    holder.username.setText(mData.get(position).getUserName());
    holder.commentContent.setText(mData.get(position).getContent());
    holder.commentTime.setText(timestampToString((Long) mData.get(position).getTimestamp()));
}
```

```
@Override
public int getItemCount() { return mData.size(); }

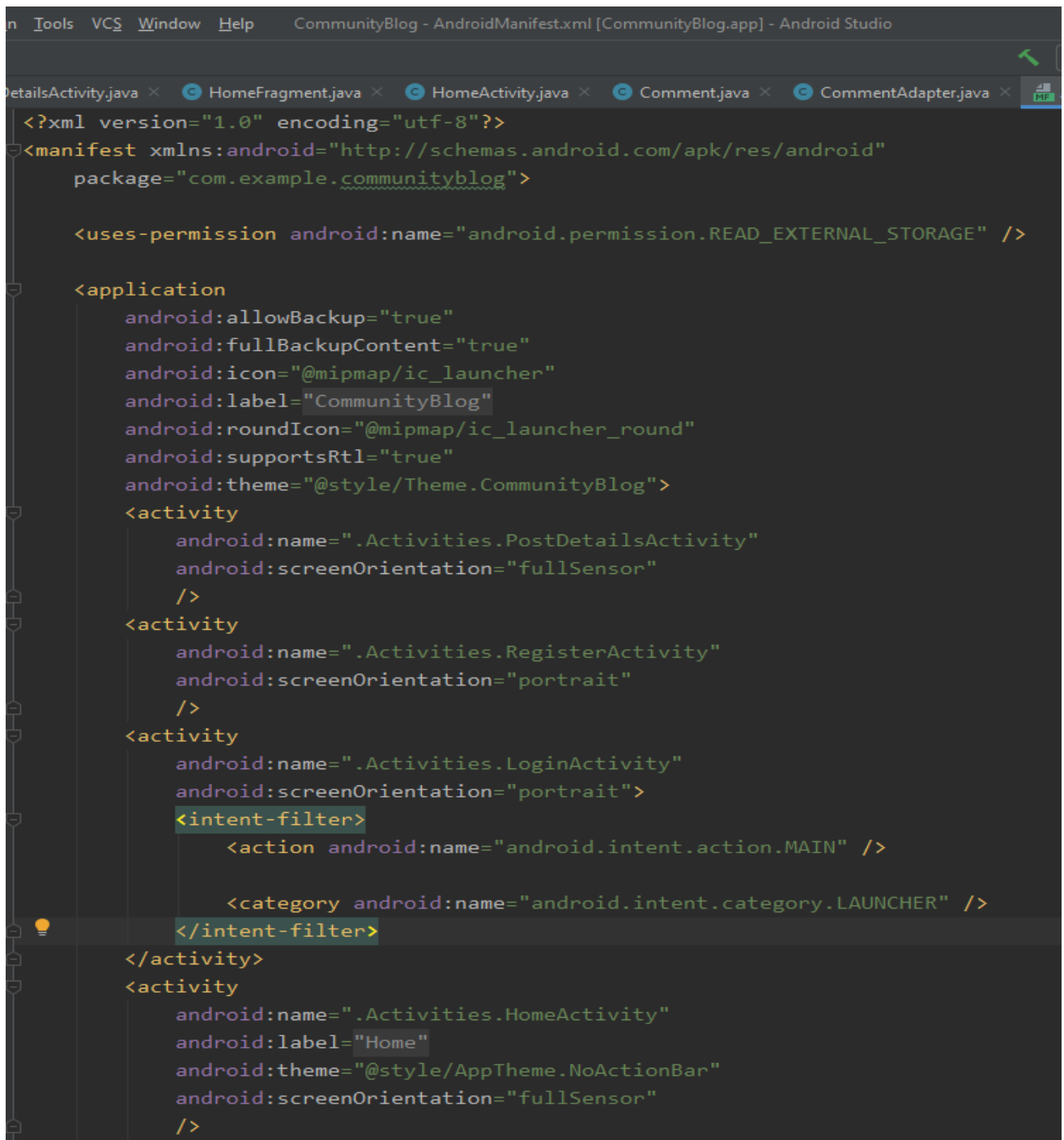
public class CommentViewHolder extends RecyclerView.ViewHolder
{
    ImageView imgUser;
    TextView username;
    TextView commentContent;
    TextView commentTime;

    public CommentViewHolder(@NonNull View itemView) {
        super(itemView);
        imgUser = itemView.findViewById(R.id.comment_user_img);
        username = itemView.findViewById(R.id.comment_username);
        commentContent = itemView.findViewById(R.id.comment_content);
        commentTime = itemView.findViewById(R.id.comment_date);
    }
}

public String timestampToString(long time)
{
    Calendar calendar = Calendar.getInstance(Locale.ENGLISH);
    calendar.setTimeInMillis(time);
    return DateFormat.format("yyyy-MM-dd hh:mm", calendar).toString();
}
```

## 2.9. Други особености

На няколко activity-та е зададен `android:screenOrientation="fullSensor"`, т.е content-ът се адаптира при ротация на устройството/емулатора. Дефинирано е разрешението `READ_EXTERNAL_STORAGE`, без което не могат да се качват снимки. LoginActivity-то играе ролята на Main и Launcher activity.



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

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

    <application
        android:allowBackup="true"
        android:fullBackupContent="true"
        android:icon="@mipmap/ic_launcher"
        android:label="CommunityBlog"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.CommunityBlog">
        <activity
            android:name=".Activities.PostDetailsActivity"
            android:screenOrientation="fullSensor"
            />
        <activity
            android:name=".Activities.RegisterActivity"
            android:screenOrientation="portrait"
            />
        <activity
            android:name=".Activities.LoginActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Activities.HomeActivity"
            android:label="Home"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="fullSensor"
            />
    </application>
</manifest>
```

В activity\_home.xml са добавени два layout файла – за информация за конкретния потребител и за navigation header menu-то :

```
1 |<?xml version="1.0" encoding="utf-8"?>
2 |<androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
3 |    xmlns:app="http://schemas.android.com/apk/res-auto"
4 |    xmlns:tools="http://schemas.android.com/tools"
5 |    android:id="@+id/drawer_layout"
6 |    android:layout_width="match_parent"
7 |    android:layout_height="match_parent"
8 |    android:fitsSystemWindows="true"
9 |    tools:openDrawer="start">
10 |
11 |    <include
12 |        layout="@layout/app_bar_home"
13 |        android:layout_width="match_parent"
14 |        android:layout_height="match_parent" />
15 |
16 |    <com.google.android.material.navigation.NavigationView
17 |        android:background="#fff"
18 |        android:id="@+id/nav_view"
19 |        android:layout_width="wrap_content"
20 |        android:layout_height="match_parent"
21 |        android:layout_gravity="start"
22 |        android:fitsSystemWindows="true"
23 |        app:headerLayout="@layout/nav_header_home"
24 |        app:menu="@menu/activity_home_drawer" />
25 |
26 |</androidx.drawerlayout.widget.DrawerLayout>
```

В процеса на работа minSdkVersion-ът е сведен до 21, за да се избегнат multidex грешките (надвишаването на 65 000 метода).

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.communityblog"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
}
```

Беше отстранен и проблем със запълването на цялостния width и height на снимките с `android:scaleType="fitXY"`. По този начин се избягва счупването на снимковия материал и при завъртане на екрана:

```
<ImageView
    android:id="@+id/login_photo"
    android:layout_width="90dp"
    android:layout_height="90dp"
    android:layout_marginTop="36dp"
    android:scaleType="fitXY"
    android:contentDescription="Sample User Photo"
    app:layout_constraintBottom_toTopOf="@+id/login_mail"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.048"
    app:srcCompat="@drawable/userphoto" />
```

### 3. Визуализация на приложението

В README.md файла към github repository-то е качено видео с тестване на проекта.