Академија струковних
студија Шумадија
Одсек Крагујевац

Студијски програм: Информатика
Предмет: Развој мобилних апликација

СТАМБЕНИ ОГЛАСИ

-пројекат-

Предметни наставник:

Проф. Хрвоје Пушкарић

Студент:

Магдалена Јакшић 23/2020

Крагујевац 2023.

САДРЖАЈ

1.УВОД

Андроид студио је развојно окружење за апликације са Андроид оперативним системом. Пружа комплетно развојно окружење, мноштво алата, едитор кода и темплејтове и омогућава лакши и бржи развој апликација. Садржи виртуелне уређаје на којима апликације могу да се тестирају.

Развој једне апликације у овом развојном окружењу биће приказан на примеру апликације за стамбене огласе. Апликација је написана у Јава програмском језику, минималан потребан АПИ је 5.0 Android Lollipop. Функционише тако што корисник може да приступи као гост или улогован корисник. Ако се улогује добија приступ функционалностима за додавање, измену и брисање огласа. Сви огласи су излистани на главном прозору и могу да се сортирају или филтрирају по одређеним критеријумима.

У првом поглављу описана је структура базе података и табеле које су потребне за функционисање апликације, након тога су приказани сви кориснички прозори појединачно, са својим дизајном, кодом и методама које су коришћене.

Први приказ је логовање описано у поглављу три, из кога може да се приступи главној страни приказаној у поглављу шест или регистрацији приказаној у поглављу два. Након регистрације корисник се преусмерава на прозор за логовање одакле приступа главној страни на којој може видети све огласе чија је имплементација приказана у поглављу пет, шест и седам. Такође може приступити прозору за додавање огласа које је приказано у поглављу осам или свом налогу, у коме су приказани сви огласи које је тај корисник избацио, чија је имплементација приказана у поглављу девет. Из свог налога може да брише или измени оглас и да се одјави што је приказано у задњем поглављу.

2. DATABASEHELPER.java

За пројекат стамбених огласа потребне су две табеле. Једна у којој ће се чувати сви огласи и једна у којој ће се чувати регистровани налози.

Табела Налози се састоји од четири колоне, за ид, корисничко име, имејл и шифру корисника. У њу ће се приликом активности регистрације убацивати унети кориснички подаци. **Табела Огласи** се састоји од девет колона, за ид, за информације и за слику огласа. У њу ће се убацивати информације о огласу приликом активности за додавање огласа.

Да би се то остварило прави се класа **DatabaseHelper** која проширује SQLiteOpenHelper за управљање базом података унутар апликације. На почетку се дефинише име базе података и верзија која се мења када се промени шема базе података. Затим се дефинишу потребне табеле и колоне које те табеле садрже и праве се упити за стварање тих табела.

```
public class DatabaseHelper extends SQLiteOpenHelper {

   private static final String DATABASE_NAME = "OglasiDB";
   private static final int DATABASE_VERSION = 1;

   //TABELA OGLASI
   public static final String TABLE_OGLASI = "Oglasi";
   public static final String COLUMN_ID = "id";
   public static final String COLUMN_LOKACIJA = "lokacija";
   public static final String COLUMN_NASELJE = "naselje";
...
   private static final String CREATE_TABLE_OGLASI =
         "CREATE TABLE " + TABLE_OGLASI + " (" +
               COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
               COLUMN_LOKACIJA + " TEXT," +
...
               COLUMN_SLIKA + " BLOB" +
               ");";

   // REGISTRACIJA-NALOZI
   public static final String TABLE_NALOZI = "nalozi";
   public static final String COLUMN_ID_NALOZI = "id_naloga";
   public static final String COLUMN_USERNAME = "username";
...
   private static final String CREATE_TABLE_NALOZI = "CREATE TABLE " + TABLE_NALOZI +
         "(" +
         COLUMN_ID_NALOZI + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
```

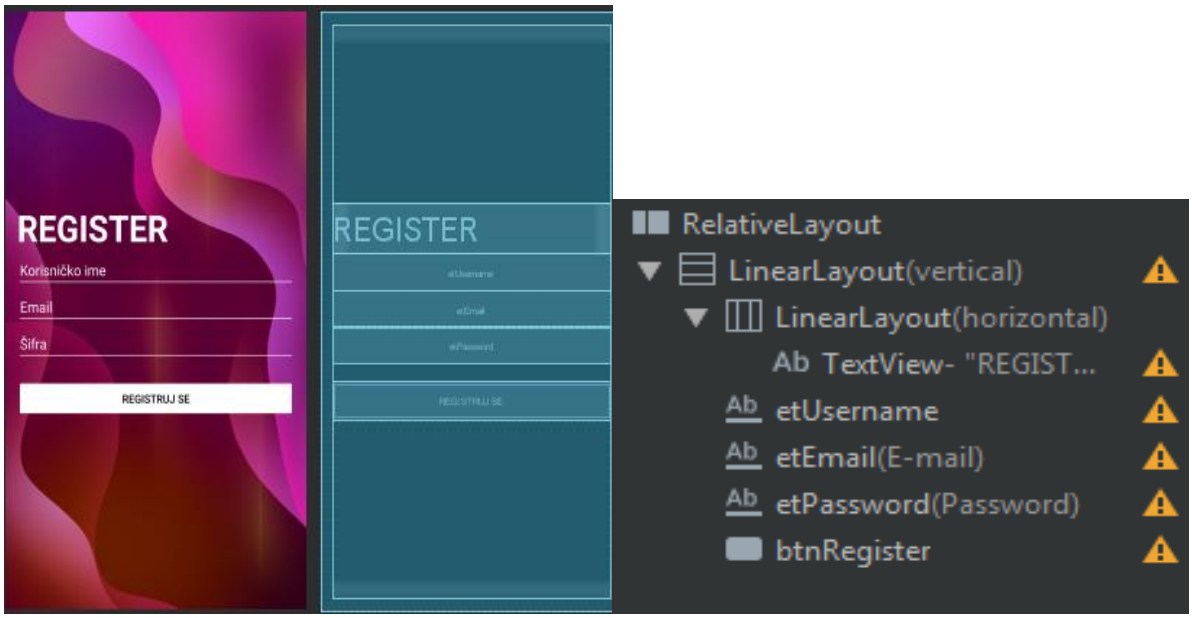
У овом коду је приказан конструктор и метода **onCreate** која се позива приликом креирања базе података и у њој се извршава упит за креирање табела. Методе **updateDatabase()** и **onUpgrade()** служе за ажурирање базе података приликом промене структуре података.

```
       public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_TABLE_OGLASI);
    db.execSQL(CREATE_TABLE_NALOZI);
}
public void updateDatabase(SQLiteDatabase db) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_OGLASI);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NALOZI);
    onCreate(db);
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    updateDatabase(db);
}
```

3.РЕГИСТРАЦИЈА

3.1.activity_register.xml

Прозор за регистрацију се састоји од вертикалног layout-a са EditText пољима која служе за унос корисничких података при регистрацији у које спадају корисничко име, имејл и шифра. Такође садржи и дугме за регистрацију и TextView који служи као наслов. Layout је централно поравнат и компонентама је прилагођен дизајн у смислу величине и боје фонта и боје позадине. Главни layout садржи слику позадине екрана и padding.



3.2. RegisterActivity.java

У **RegisterActivity** се **пре onCreate()** методе дефинишу поменуте компоненте из xml фајла и DatabaseHelper за рад са базом података. Затим се **у onCreate()** методи иницијализују помоћу **findViewById()** где се прослеђује идентификатор одговарајућег елемента који је дефинисан у xml-у.

Свака класа проширује **AppCompatActivity** што означава да се користи као активност и има приступ додатним функционалостима.

```
public class RegisterActivity extends AppCompatActivity {

private EditText etUsername;
private EditText etEmail;
private EditText etPassword;
private Button btnRegister;
DatabaseHelper databaseHelper;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);

    etUsername = findViewById(R.id.etUsername);
    etEmail = findViewById(R.id.etEmail);
    etPassword = findViewById(R.id.etPassword);
    btnRegister = findViewById(R.id.btnRegister);
    databaseHelper= new DatabaseHelper(this);
```

Дугмету за регистрацију се додаје **onClickListener()** што значи када корисник кликне на дугме десиће се акције дефинисане у овом блоку. За ово конкретно дугме, на клик узимају се вредности које је корисник унео а затим се помоћу методе **checkUsernameExist()** која се налази у **DatabaseHelper** класи проверава да ли унесено корисничко име већ постоји у бази тако што се пролеђује тој методи на проверу.

Ако постоји појављује се порука и прекида регистрација, у супротном се вредности додају у базу података помоћу методе **addNalog()** која је такође дефинисана у DatabaseHelper класи.

Ако је регистрација успешна помоћу Intent-а се корисник преусмерава на прозор за логовање уз поруку да се успешно регистровао у супротном му излази порука да регистрација није успела. **Intent** је објекат који служи за комуникацију између компоненти и покретање сервиса или активности, док је **Toast** елемент корисничког интерфејса који обично служи за приказ порука кориснику.

```
       btnRegister.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String username = etUsername.getText().toString();
        String email = etEmail.getText().toString();
        String password = etPassword.getText().toString();

        if (databaseHelper.checkUsernameExists(username)) {
            Toast.makeText(RegisterActivity.this, "Korisničko ime već
postoji. Izaberite drugo.", Toast.LENGTH_SHORT).show();
            return;
        }

        long result = databaseHelper.addNalog(username, email, password);

        if (result != -1) {
            Toast.makeText(RegisterActivity.this, "Registracija uspešna!",
Toast.LENGTH_SHORT).show();
            Intent intent=new Intent(RegisterActivity.this,
LoginActivity.class);
            startActivity(intent);
            finish();
        } else {
            Toast.makeText(RegisterActivity.this, "Registracija nije uspela.
Pokušajte ponovo.", Toast.LENGTH_SHORT).show();
        }
    }
});
```

2.3. методе из DatabaseHelpera

Методи **addNalog()** која се користи у RegisterActivity се прослеђује корисничко име, имејл и шифра. У њој се отвара база у режиму за писање података и онда се објекту **ContentValues** прослеђују унесени подаци и припадајуће колоне из табеле Налози. Помоћу методе **insert()** се затим тај ред додаје у табелу и затвара се база података. Метода враћа резултат то јест ид додатог реда. Ако је ид -1 онда регистрација није успела. **ContentValues** је објекат који служи за мапирање имена колона на вредности, а **insert()** је метода за додавање новог реда у табелу.

```
public long addNalog(String username, String email, String password) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_USERNAME, username);
    values.put(COLUMN_EMAIL, email);
    values.put(COLUMN_PASSWORD, password);
    long result = db.insert(TABLE_NALOZI, null, values);
    db.close();
    return result;
}
```

Методи **checkUsernameExists()** се прослеђује корисничко име и отвара се база података у режиму читања. Дефинише се **Cursor** за итерацију кроз резултате упита и **упит** који садржи име табеле над којом се извршава, низ колона који желимо, у овом случају је ид, затим услов где се проверава корисничко име и затим аргумент за услов то јест корисничко име. Након тога се добија број редова и затвара cursor. Ако је број редова већи од нула значи да корисничко име постоји.

Cursor је интерфејс за приступ резултатима упита. Обично се користи приликом извршавања упита где треба вршити итерацију кроз резултате.

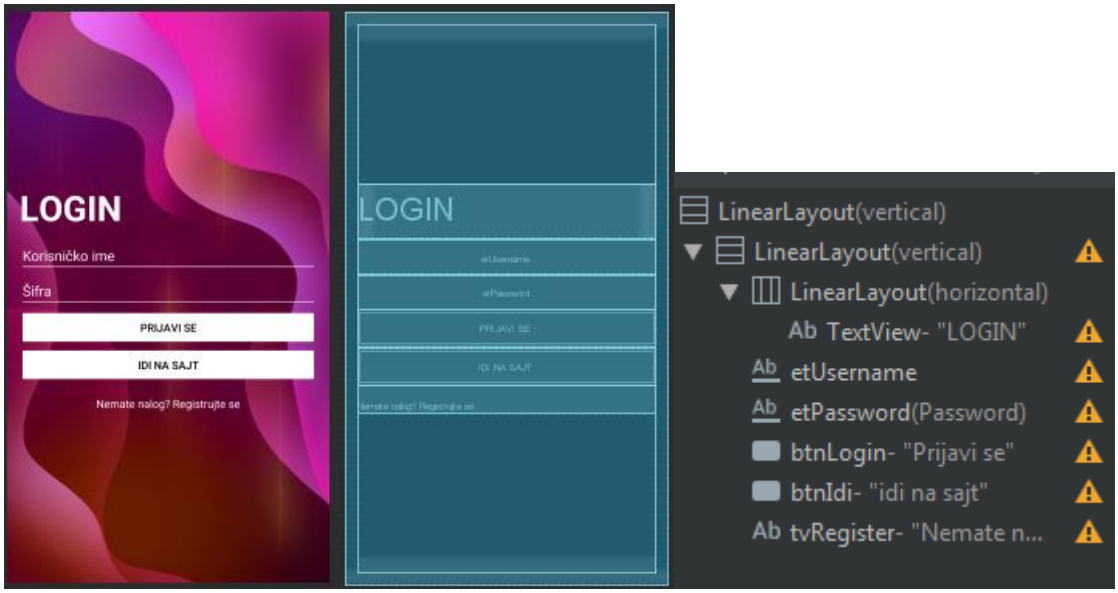
Query() је метода koja служи за извођење SQL упита.

```
   public boolean checkUsernameExists(String username) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.query(
            TABLE_NALOZI,
            new String[]{COLUMN_ID_NALOZI},
            COLUMN_USERNAME + "=?",
            new String[]{username},
            null,
            null,
            null,
            null
    );

    int count = cursor.getCount();
    cursor.close();
    return count > 0;
}
```

4. ЛОГОВАЊЕ

4.1.activity_login.xml

Прозор за логовање садржи вертикални layout, централно позициониран са две EditText компоненте које служе за унос корисничког имена и шифре, два дугмета од којих је прво за пријаву а друго за приступ огласима без логовања. Садржи и два TextView-а, један за наслов а други који је кликабилан и води ка прозору за регистрацију. Позадина је дефинисана у главном layout-у.



4.2. LoginActivity.java

Компоненте из xml-а су дефинисане и иницијализоване на исти начин као што је описано у RegisterActivity.

```
       public class LoginActivity extends AppCompatActivity {

    private EditText etUsername, etPassword;
        ...
    DatabaseHelper databaseHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        ...
        databaseHelper= new DatabaseHelper(this);
```

Кликом на дугме **btnLogin** које представља дугме за пријаву узимају се унесене вредности из EditText поља и помоћу методе **checkLogin()** из DatabaseHelper-а се проверава да ли су информације тачне. Ако јесу помоћу Intent-а се прелази на **MainActivity** који представља главни део апликације и прослеђује се унесено корисничко име помоћу **putExtra()** да би се пратило који корисник је улогован. Ако су подаци нетачни излази одговарајућа порука.

```
       btnLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String username = etUsername.getText().toString();
        String password = etPassword.getText().toString();
        boolean loginSuccessful = databaseHelper.checkLogin(username,
password);
        if (loginSuccessful) {
            Intent intent = new Intent(LoginActivity.this,
MainActivity.class);
            intent.putExtra("USERNAME", username);
            startActivity(intent);
            finish();
        } else {
            Toast.makeText(LoginActivity.this, "Pogresno korisnicko ime ili
lozinka!", Toast.LENGTH_SHORT).show();
```

Кликом на **tvRegister** који представља кликабилни **TextView** за регистрацију помоћу Intent-a се корисник преусмерава на прозор за регистрацију.

```
       tvRegister.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        Intent registerIntent = new Intent(LoginActivity.this,
RegisterActivity.class);
        startActivity(registerIntent);
    }
});
```

Кликом на дугме **btnIdi(** „иди на сајт“) се помоћу Intent-a корисник преусмерава на главни прозор и шаље се информација помоћу **putExtra()** да корисник није улогован што значи да ће му функције за додавање огласа и приступ налогу бити сакривене. **PutExtra()** је метода за слање додатних информација(кључ-вредност) путем Intent-a.

```
       btnIdi.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent mainintent = new Intent(LoginActivity.this, MainActivity.class);
        mainintent.putExtra("IS_NOT_LOGGED_IN", true);
        startActivity(mainintent);
    }
});
```

4.3. методе из DatabaseHelpera

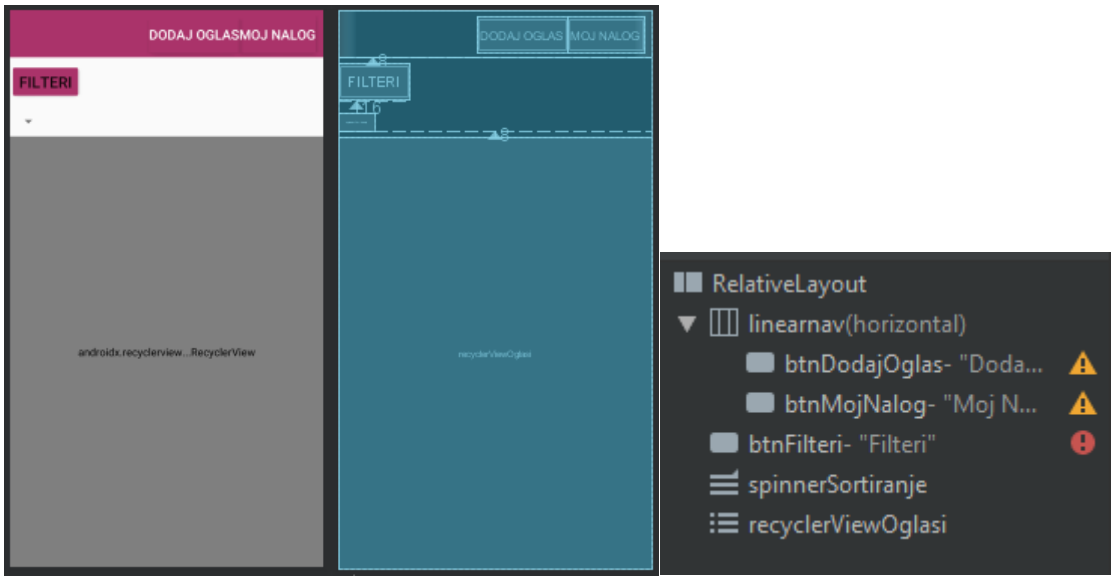
Методи **checkLogin()** која се користи у LoginActivity се прослеђују корисничко име и шифра. Отвара се база података у режиму читања и прави се Cursor објекат. Принцип рада је исти као и у методи за регистрацију само што упит садржи већи услов и више аргумената. Проверава се да ли постоји ред у табели налози чији је садржај исти као информације које је корисник унео. Ако постоји логовање је успешно и метода враћа true .

```
        public boolean checkLogin(String username, String password) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.query(
            TABLE_NALOZI,
            new String[]{COLUMN_ID_NALOZI},
            COLUMN_USERNAME + "=? AND " + COLUMN_PASSWORD + "=?",
            new String[]{username, password},
            null,
            null,
            null,
            null
    );

    int count = cursor.getCount();
    cursor.close();
    return count > 0;
}
```

5. ПРИКАЗ ОГЛАСА

5.1. activity_main.xml

XML за главну страну садржи навигацију са два дугмета „ДОДАЈ ОГЛАС“ и „МОЈ НАЛОГ“, дугме са опцијама за филтрирање огласа, спинер са опцијама за сортирање огласа и recyclerView који служи да прикаже огласе. **RecyclerView** је компонента корисничког интерфејса која служи за приказивање листа података. Користи **Адаптер** да би повезао податке са приказом, **LayoutManager** за дефинисање распореда елемената у листи, и **ViewHolder** који чува референцу на сваки елемент.



5.2. MainActivity.java

Компоненте из xml-а су дефинисане на стандардни начин. Поред њих дефинисан је код који се користи као индентификатор приликом позива **startActivityForResult(), oglasiList** листу која ће садржати огласе за recyclerView, **oglasiAdapter** за повезивање листе са recyclerView-ом и управљање сваким елементом у листи, **oglasChangeListener** интерфејс који ће да служи за комуникацију са другим активностима.

```
      public class MainActivity extends AppCompatActivity{
    private static final int MOJ_NALOG_REQUEST_CODE = 1001;
    private List<Oglas> oglasiList;
    private RecyclerView recyclerViewOglasi;
    private OglasiAdapter oglasiAdapter;
    private OglasiAdapter2 oglasiAdapter2;
..
    private Spinner spinnerSortiranje;
    DatabaseHelper databaseHelper;
    public static OnOglasChangeListener oglasChangeListener;
```

Даље је приказан код за лансирање активности мој налог **ActivityResultLauncher<Intent> mojNalogLauncher** пошто је startActivityForRseult() deprecated(застарело). **Result->** означава шта ће се десити када се активност заврши. У овом случају приказ свих огласа се освежава тако што се прави листа свих огласа из базе која представља освежену листу свих огласа и прослеђује се Адаптеру, а затим се он обавештава да је дошло до промена јер се у активности Мој Налог огласи бришу и мењају.

```
    private ActivityResultLauncher<Intent> mojNalogLauncher =
registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == RESULT_OK) {
                    List<Oglas> refreshedOglasi =
databaseHelper.getAllOglasi();
                    oglasiAdapter.setOglasiList(refreshedOglasi);
                    oglasiAdapter.notifyDataSetChanged();
                }
            }
    );
```

У **onCreate()** методи путем **getIntent()** се узима корисничко име које је прослеђено путем Intent-а из активности логовања и из активности мој налог. Корисничко име је више пута прослеђено приликом преласка и враћања из различитих активности да би се спречило губљење информације. **RecyclerView** се повезује се **Адаптером** који садржи листу свих огласа која је добијена помоћу методе **getAllOglasi()** из DatabaseHelper-а и поставља му се **линеарни layout**.

```
...
databaseHelper = new DatabaseHelper(this);

Intent intent = getIntent();
String username = intent.getStringExtra("USERNAME");
String username2=intent.getStringExtra("IME_VLASNIKA");

recyclerViewOglasi = findViewById(R.id.recyclerViewOglasi);
recyclerViewOglasi.setLayoutManager(new LinearLayoutManager(this));
oglasiList = databaseHelper.getAllOglasi();
oglasiAdapter = new OglasiAdapter(this, oglasiList);
recyclerViewOglasi.setAdapter(oglasiAdapter);
```

Ако се корисник није улоговао послат је Intent **IS_NOT_LOGGED_IN** и онда се сакривају функционалности за додавање огласа и приступ налогу.

```
if (intent.hasExtra("IS_NOT_LOGGED_IN")) {

        btnDodajOglas.setVisibility(View.GONE);
        btnMojNalog.setVisibility(View.GONE);

}
```

Кликом на дугме **btnDodajOglas** за додавање огласа корисник се преусмерава на активност за додавање огласа којој се прослеђује корисничко име улогованог корисника.

```
    btnDodajOglas.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {

    Intent intent = new Intent(MainActivity.this, DodajOglasActivity.class);
    intent.putExtra("KORISNICKO_IME", username);
    intent.putExtra("KORISNICKO_IME2",username2);
    startActivity(intent);
```

Кликом на дугме **btnMojNalog** за приступ налогу корисник се преусмерава на активност Мој Налог којој се такође шаље корисничко име. Затим се имплементира метода **onOglasChanged()** из интерфејса **onOglasChangeListener** и кад се она позове приликом измене огласа освежава се листа огласа и обавештава се адаптер о променама да би се освежио recyclerView. Покреће се мој налог активност преко дефинисаног **ActivityResultLauncher<Intent> mojNalogLauncher.**

```
   btnMojNalog.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    Intent intentNalog=new Intent(MainActivity.this,MojNalog.class);
    intentNalog.putExtra("KORISNICKO_IME", username);
    intentNalog.putExtra("KORISNICKO_IME2",username2);
    MainActivity.oglasChangeListener = new OnOglasChangeListener() {
        @Override
        public void onOglasChanged() {
            List<Oglas> refreshedOglasi = databaseHelper.getAllOglasi();
            oglasiAdapter.setOglasiList(refreshedOglasi);
            oglasiAdapter.notifyDataSetChanged();
        }
    };
    mojNalogLauncher.launch(intentNalog);
```

Када се кликне дугме **btnFilteri** за филтрирање огласа по критеријумима позива се **showFiltersDialog()** метода за приказ прозора са опцијама за филтрирање огласа.

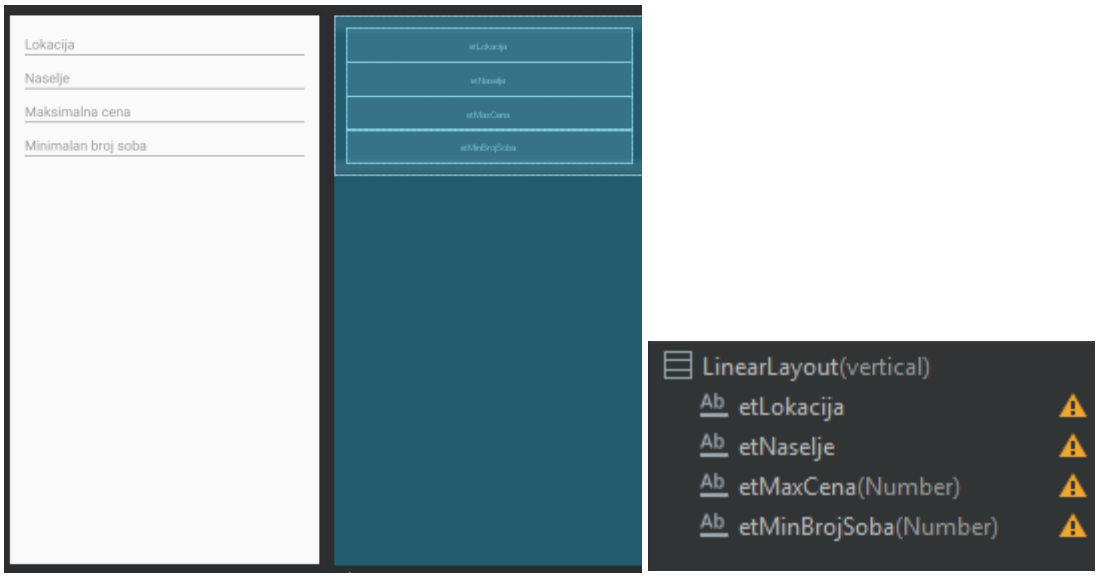
```
   btnFilteri.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    showFiltersDialog();
}
```

Сортирање у **spinner-y** се врши тако што се прави **ArrayAdapter** који користи sortiranje_niz.xml фајл у ком се налазе опције за сортирање, по цени опадајуће и растуће, и поставља layout за падајући мени са опцијама. Тај адаптер се поставља на спинер који приказује те опције корисницима. Када се одабере опција позива се **onItemSelected()** метода у којој се позива метода **sortirajOglase().** . **Position** садржи позицију одабране опције и позива се метода **sortirajOglase().**

```
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
            this,
            R.array.sortiranje_niz,
            android.R.layout.simple_spinner_item
    );

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
    spinnerSortiranje.setAdapter(adapter);
    spinnerSortiranje.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parentView, View
selectedItemView, int position, long id) {
            sortirajOglase(position);
        }

        @Override
        public void onNothingSelected(AdapterView<?> parentView) {
        }
    });

}
```

Метода **showFiltersDialog()** служи за приказ прозора са опијама за филтрирање које се састоји од поља за унос локације, насеља, максималне цене и минималног броја соба. Иницијализује се **builder** за дијалог и поставља му се одговарајућ xml који представља изглед те компоненте. Проналазе се елементи из xml-a, поставља се дугме филтрирај и кликом на то дугме узимају се унете вредности. Ако су поља за максималну цену и минималан број соба празне онда се вредности постављају на нулу. Прави се копија листе свих огласа да би се филтери применили на ту листу и листа са филтрираним огласима помоћу методе **getFilteredOglasi()** која се затим ставља у Адаптер који се обавештава да су подаци промењени како би ажурирао recyclerView.

```
       private void showFiltersDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Filteri");
    View dialogView = getLayoutInflater().inflate(R.layout.dialog_filters,
null);
    builder.setView(dialogView);


    final EditText etLokacija = dialogView.findViewById(R.id.etLokacija);
    ...
    builder.setPositiveButton("Filtriraj", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

            String lokacija = etLokacija.getText().toString();
            ...
            int maxCena = strMaxCena.isEmpty() ? 0 :
Integer.parseInt(strMaxCena);
            int minBrojSoba=strMinBrojSoba.isEmpty() ? 0 :
Integer.parseInt(strMinBrojSoba);

            List<Oglas> kopijaOgLasiListe = new ArrayList<>(oglasiList);
            List<Oglas> filtriraniOglasi = getFilteredOglasi(lokacija,
naselje,maxCena, minBrojSoba, kopijaOgLasiListe);
            oglasiAdapter.setOglasiList(filtriraniOglasi);
            oglasiAdapter.notifyDataSetChanged();
        }
    });

    builder.setNegativeButton("Otkaži", new DialogInterface.OnClickListener()
{
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        } ...   builder.show();
```



Приказан је xml за дизајн прозора дијалога.

Методи **getFilteredOglasi()** се прослеђују локација, насеље, минималан број соба, максимална цена и листа свих огласа. Иницијализује се листа филтрираних огласа. Итерира се кроз сваки оглас и постављају се услови, ако је један од услова задовољен оглас се додаје у листу. Ако нису постављени враћа се листа са свим огласима.

```
       private List<Oglas> getFilteredOglasi(String lokacija, String naselje,
int maxCena, int minBrojSoba, List<Oglas> listaoglasa) {
    List<Oglas> filtriraniOglasi = new ArrayList<>();

    for (Oglas oglas : listaoglasa) {
        boolean uslovLokacija = lokacija.isEmpty() ||
oglas.getLokacija().equalsIgnoreCase(lokacija);
        boolean uslovNaselje = naselje.isEmpty() ||
oglas.getNaselje().equalsIgnoreCase(naselje);
        boolean uslovMaxCena = maxCena <= 0 || oglas.getCena() <= maxCena;
        boolean uslovMinBrojSoba = minBrojSoba <= 0 || oglas.getBrojSoba() >=
minBrojSoba;
        if (uslovLokacija && uslovNaselje && uslovMaxCena &&
uslovMinBrojSoba) {
            filtriraniOglasi.add(oglas);
        }
    }

    if (lokacija.isEmpty() && naselje.isEmpty() && maxCena <= 0 &&
minBrojSoba <= 0) {
        return listaoglasa;
    }

    return filtriraniOglasi;
}
```

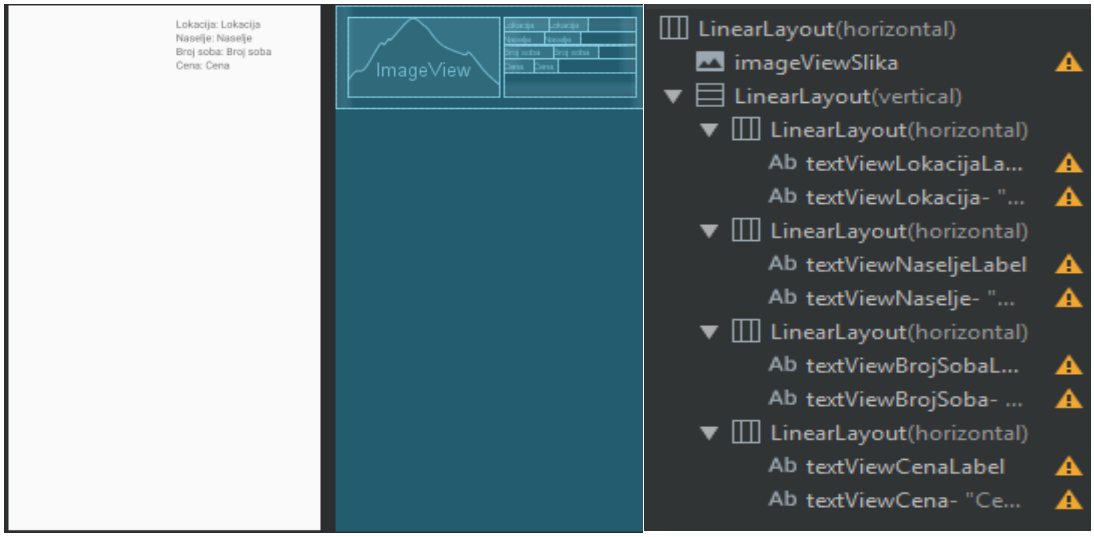
Метода **sortirajOglase()** служи за сортирање огласа по цени. Добија се листа огласа из адаптера, помоћу case знамо која је опција одабрана и помоћу Collections и Comparator се огласи сортирају по цени у растућем или опадајућем поретку. Затим се поставља листа у адаптер и адаптер се обавештава о променама што омогућује да се огласи прикажу по одговарајућем сортирању.

```
       private void sortirajOglase (int selectedItemPosition){
    List<Oglas> oglasi = oglasiAdapter.getOglasiList();

    switch (selectedItemPosition) {
        case 0:
            //RASTUCE
            Collections.sort(oglasi, new Comparator<Oglas>() {
                @Override
                public int compare(Oglas o1, Oglas o2) {
                    return Integer.compare(o1.getCena(), o2.getCena());
                }
            });
            break;
        case 1:
            //OPADAJUCE
            Collections.sort(oglasi, new Comparator<Oglas>() {
                @Override
                public int compare(Oglas o1, Oglas o2) {
                    return Integer.compare(o2.getCena(), o1.getCena());
                }
            });
            break;
    }
    oglasiAdapter.setOglasiList(oglasi);
    oglasiAdapter.notifyDataSetChanged();
}
```

6.ОГЛАСИ

6.1. item_oglas.xml

Приказан XML служи за дизајн једног огласа. Састоји се од слике и вертикалног layout-а са хоризонталним у којима се налазе информације за локацију, насеље, цену и број соба. Представља начин на који ће кориснику да се прикажу огласи из базе у MainActivity.



6.2.Oglas.java

Класа **Oglas** представља модел огласа по табели из базе, са одговарајућим гетерима и сетерима за сваку колону и конструктором.

```
        public class Oglas {
    private long id;
    private String lokacija;
    private String naselje;
...
    private byte[] slika;

        //гетери и сетери
        public Oglas(long id, String lokacija, String naselje, int cena, int brojSoba,
String opis, String kontakt,String vlasnik, byte[] slika) {
    this.id = id;
    this.lokacija = lokacija;
    this.naselje = naselje;
...
    this.vlasnik = vlasnik;
    this.slika = slika;
}
```

6.3.OglasAdapter.java

Адаптер се користи за повезивање огласа са RecyclerView-ом и прима контекст и листу огласа која ће се приказивати као параметре за конструктор. **Context** је класа која пружа информације о тренутом стању апликације и даје приступ различитим ресурсима и операцијама, а Адаптер је компонента која се користи за повезивање листе или генерално неког извора података са корисничким интерфејсом.

```
       public class OglasiAdapter extends
RecyclerView.Adapter<OglasiAdapter.OglasViewHolder> {

    private Context context;
    private List<Oglas> oglasiList;

    public OglasiAdapter(Context context, List<Oglas> oglasiList) {
        this.context = context;
        this.oglasiList = oglasiList;
    }
```

Ова метода се користи за креирање новог **ViewHolder** објекта који чува референцу на приказ сваког елемента, item_oglas.xml се претвара у поглед и прослеђује се том објекту.

```
       @Override
public OglasViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(context).inflate(R.layout.item_oglas,
parent, false);
    return new OglasViewHolder(view);
}
```

Метода **onBindViewHolder()** служи за повезивање података са ViewHolder-ом. Узима се објекат типа Оглас на одређеној позицији који садржи податке за приказ и они се прослеђују одговарајућим пољима у ViewHolder-у.

```
       @Override
public void onBindViewHolder(OglasViewHolder holder, int position) {
    Oglas oglas = oglasiList.get(position);

    holder.textViewLokacija.setText(oglas.getLokacija());
    ...
```

Узима се и слика која се помоћу **Bitmap** методе **decodeByteArray()** декодира из бајт низа у Bitmap објекат како би се могла приказати у ImageViewSlika из ViewHolder-a. **Bitmap** је класа која представља мапу битова и служи за манипулацију сликама.

```
    if (oglas.getSlika() != null) {
        Bitmap bitmap = BitmapFactory.decodeByteArray(oglas.getSlika(), 0,
oglas.getSlika().length);
        holder.imageViewSlika.setImageBitmap(bitmap);
    }
```

Затим се поставља **onClickListener** на сваку ставку. Када корисник кликне на неки оглас преусмерава се на **активност додавања огласа** којој се шаљу додатни подаци о огласу на који је корисник кликнуо преко Intent-a.

```
    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
           Intent intent = new Intent(context, DetaljiOglasaActivity.class);
            intent.putExtra("ID",oglas.getId());
            intent.putExtra("LOKACIJA", oglas.getLokacija());
            ...
            intent.putExtra("OPIS", oglas.getOpis());
            context.startActivity(intent);
        }
    });
}
```

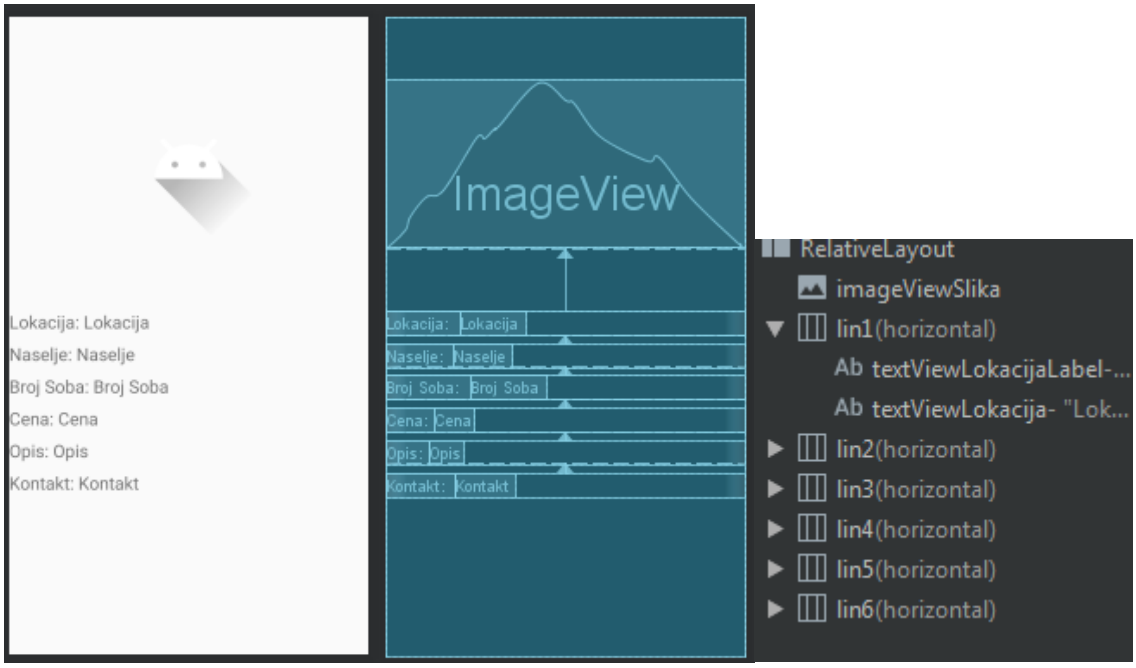
Даље су приказане методе за враћање броја огласа у листи, за постављање нове листе огласа приликом ажурирања или промена и приступ тренутној листи огласа. Даље се иницијализују референце на елементе из item_layout.xml.

```
       @Override
public int getItemCount() {
    return oglasiList.size();
}
public void setOglasiList(List<Oglas> oglasiList) {
    this.oglasiList = oglasiList;
}
public List<Oglas> getOglasiList() {
    return oglasiList;
}

public class OglasViewHolder extends RecyclerView.ViewHolder {
    TextView textViewLokacija, textViewNaselje, textViewCena,
textViewBrojSoba;
    ImageView imageViewSlika;

    public OglasViewHolder(View itemView) {
        super(itemView);
        textViewLokacija = itemView.findViewById(R.id.textViewLokacija);
        textViewNaselje = itemView.findViewById(R.id.textViewNaselje);
        textViewCena = itemView.findViewById(R.id.textViewCena);
        textViewBrojSoba = itemView.findViewById(R.id.textViewBrojSoba);
        imageViewSlika = itemView.findViewById(R.id.imageViewSlika);
    }
```

7.ДЕТАЉИ ОГЛАСА

7.1. activity_detalji_oglasa.xml

Овај Layout се састоји од слике и свих информација о огласу које прима из Intent-а које је Адаптер послао. Представља шта ће корисник видети када кликне на одређени оглас.



7.2.DetaljiOglasaActivity.java

Прво се добављају информације из Intent-а добијеног од Адаптера и постављају се у одговарајуће TextView-е из activity_detalji_oglasa.xml за приказ кориснику.

```
        Intent intent = getIntent();
        String lokacija = intent.getStringExtra("LOKACIJA");
...
        String kontakt=intent.getStringExtra("KONTAKT");

        TextView textViewLokacija = findViewById(R.id.textViewLokacija);
        TextView textViewNaselje = findViewById(R.id.textViewNaselje);
        );

 textViewLokacija.setText(lokacija);
 ...
 textViewopis.setText(opis);
```

Затим се узима ид огласа и ако је валидан, помоћу методе **getSlikabyID()** из Databasehelper-а се узима слика огласа , декодира у Bitmap и поставља у ImageView. Слика није послата преко Intent-а јер је превелика већ је директно учитана из базе помоћу поменуте методе.

```
       ImageView imageViewSlika = findViewById(R.id.imageViewSlika);
       long oglasId = intent.getLongExtra("ID", -1);

       if (oglasId != -1) {
           byte[] slikaByteArray = databaseHelper.getSlikaById(oglasId);

           if (slikaByteArray != null) {
               Bitmap bitmap = BitmapFactory.decodeByteArray(slikaByteArray,
0, slikaByteArray.length);
               imageViewSlika.setImageBitmap(bitmap);
           }
       }
```

7.3. Методе из DatabaseHelpera

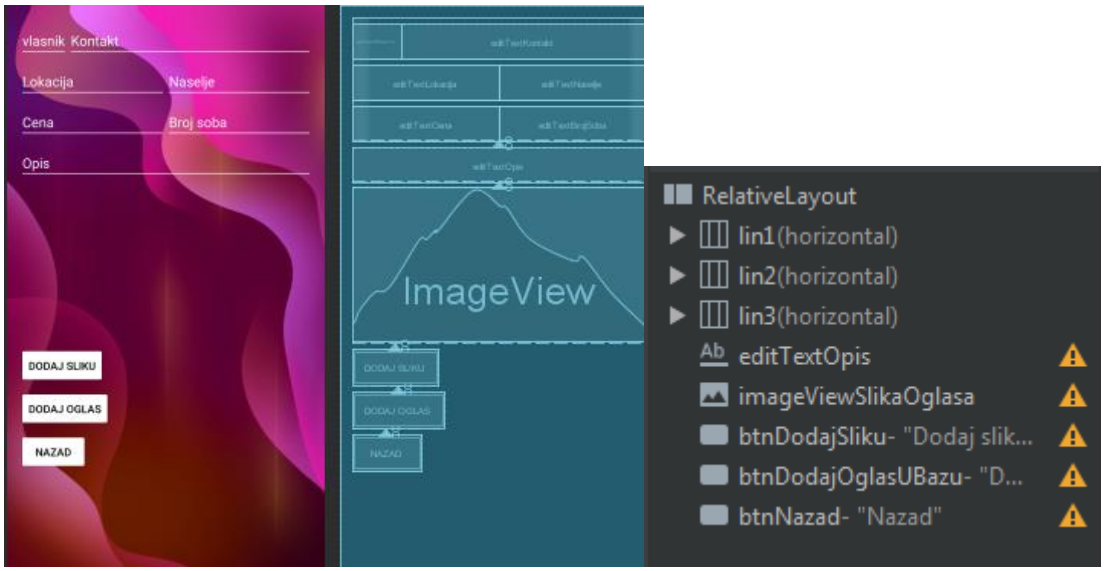
Принцип рада **getSlikaById()** је исти као и код претходно поменутих метода. Прво се прослеђује ид и затим се помоћу Cursor-а и упита дохвата слика из базе података из табеле Огласи и ставља у ByteArray па по потреби се декодује како би се приказала.

```
       public byte[] getSlikaById(long oglasId) {
    SQLiteDatabase db = this.getReadableDatabase();

    String[] columns = {COLUMN_SLIKA};
    String selection = COLUMN_ID + " = ?";
    String[] selectionArgs = {String.valueOf(oglasId)};

    Cursor cursor = db.query(
            TABLE_OGLASI,
            columns,
            selection,
            selectionArgs,
            null,
            null,
            null
    );

    byte[] slikaByteArray = null;
    if (cursor != null && cursor.moveToFirst()) {
        slikaByteArray = cursor.getBlob(cursor.getColumnIndex(COLUMN_SLIKA));
        cursor.close();
    }

    db.close();
    return slikaByteArray;
}
```

8. ДОДАВАЊЕ ОГЛАСА

8.1. activity_dodaj_oglas.xml

Прозор за додавање огласа се састоји од три хоризонтална layout-а са EditText пољима у које се уносе информације о огласу. Поље власник се аутоматски попуњава корисничким именом улогованог корисника и овој функцији немају приступ неулоговани корисници. Такође садржи ImageView у коме ће се приказати слика коју корисник дода, дугме за додавање слике, дугме за додавање огласа и дугме за повратак на главну активност.



8.2.DodajOglasActivity.java

Као и у претходним активностима дефинишу се променљиве па се касније у **onCreate()** иницијализују компоненте дефинисане у xml-у.

```
private EditText editTextLokacija, editTextNaselje, editTextCena..
...
   private Bitmap selectedImage;
   DatabaseHelper databaseHelper;
```

Дефинише се **ActivityResultLauncher** који служи за одабир слике. Користи **getContent()** контракт за добијање садржаја из одређеног извора. Затим ако постоји адреса **(uri)** добија се Bitmap слика из те адресе и поставља у ImageView.

```
private final ActivityResultLauncher<String> launcher =
 registerForActivityResult(new ActivityResultContracts.GetContent(),
uri -> {
     if (uri != null) {
         try {
             selectedImage =
MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
             imageViewSlikaOglasa.setImageBitmap(selectedImage);
         } ...
```

Корисничко име се добија преко Intent-а и поставља у поље власник.

```
Intent intent = getIntent();
String korisnickoIme = intent.getStringExtra("KORISNICKO_IME");
String korisnickoIme2=intent.getStringExtra("KORISNICKO_IME2");
editTextVlasnik=findViewById(R.id.editTextVlasnik);
editTextVlasnik.setText(korisnickoIme);

if (korisnickoIme != null) {
    editTextVlasnik.setText(korisnickoIme);
} else if (korisnickoIme2 != null) {
    editTextVlasnik.setText(korisnickoIme2);
}
```

Када се кликне дугме **btnDodajSliku** за додавање слике извршава се метода **odaberiSliku().**

```
btnDodajSliku.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    odaberiSliku();
...
```

Када се кликне дугме **btnDodajOglasUBazu** за додавање огласа извршава се метода **dodajOglasUBazu()** и корисник се преусмерава на главну активност којој се шаље и корисничко име да се не би губило приликом преласка активности са једне на другу.

```
       btnDodajOglasUBazu.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        dodajOglasUBazu();
        String ime=editTextVlasnik.getText().toString();
        Intent intent1=new
Intent(DodajOglasActivity.this,MainActivity.class);
        intent1.putExtra("IME_VLASNIKA",ime);
        startActivity(intent1);
```

Кликом на дугме **btnNazad** за повратак назад тренутна активност се завршава па се враћа на главну активност.

```
           btnNazad.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            finish();
        }
    });
}
```

Метода **odaberiSliku()** покреће дефинисан launcher за одабир слике.

```
       private void odaberiSliku() {
    launcher.launch("image/*");
}
```

Метода **dodajOglasUBazu()** узима унете вредности корисника и корисничко име и слику претвара у низ бајтова помоћу методе **convertBitmapToBytes()** да би се могла додати у базу. Затим помоћу **ContentValues** повезује колоне и вредности, и потом тај ред помоћу методе **insert()** убацује у табелу Огласи. Ако је додавање успешно корисник се преусмерава на главну активност.

```
private void dodajOglasUBazu() {
    String lokacija = editTextLokacija.getText().toString().trim();
    String naselje = editTextNaselje.getText().toString().trim();
    ...

    byte[] imageBytes = null;
    if (selectedImage != null) {
        imageBytes = convertBitmapToBytes(selectedImage);
    }

    if (!lokacija.isEmpty() && !naselje.isEmpty()) {
        DatabaseHelper databaseHelper = new DatabaseHelper(this);
        ContentValues values = new ContentValues();
        values.put("lokacija", lokacija);
        ...
        values.put("slika", imageBytes);

        long newRowId = databaseHelper.getWritableDatabase().insert("Oglasi", null,
values);

        if (newRowId != -1) {
            Intent intent = new Intent(DodajOglasActivity.this, MainActivity.class);
            startActivity(intent);
            finish();
        } else {
            Toast.makeText(DodajOglasActivity.this, "Greška pri dodavanju oglasa.",
Toast.LENGTH_SHORT).show();
        }
    }
```

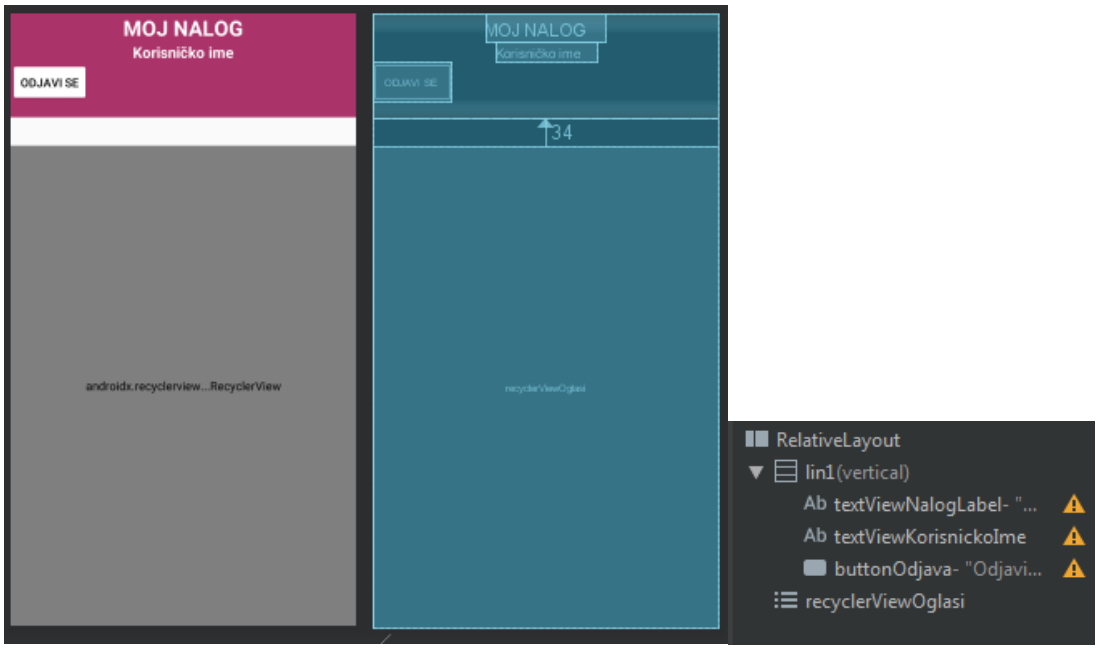
Метода **convertBitmapToBytes()** служи за претварање Bitmap-е у низ бајтова како би слика била убачена у базу података.

```
    private byte[] convertBitmapToBytes(Bitmap bitmap) {
    ByteArrayOutputStream stream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, stream);
    return stream.toByteArray();
}
```

9.МОЈ НАЛОГ

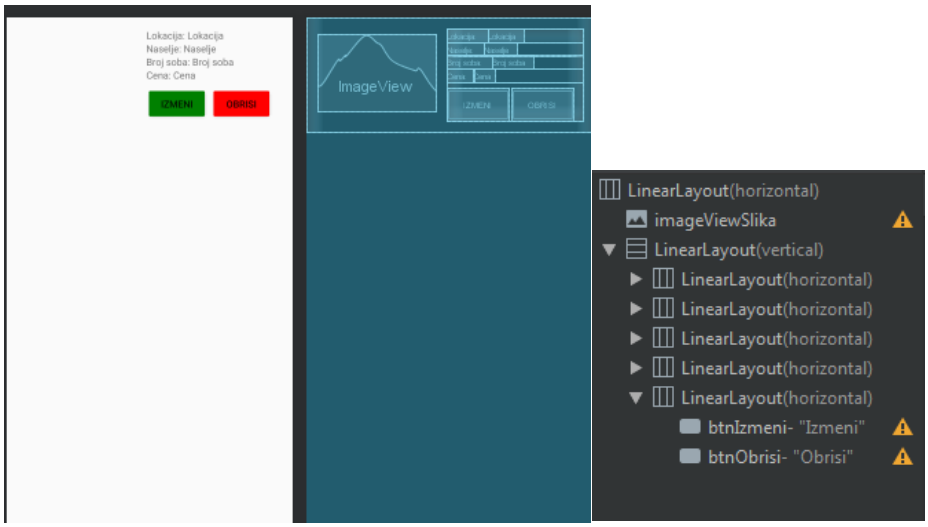
9.1.moj_nalog.xml

Када је корисник улогован и када у MainActivity кликне на дугме „мој налог“ приказује му се овај поглед. Састоји се од навигације са насловом , корисничким именом улогованог корисника, дугметом за одјаву и recyclerView-а за приказ огласа које је тај корисник поставио.



9.2.item_oglas2.xml

Овај XML представља како ће корисник видети своје избачене огласе када уђе на свој налог. Дизајн је исти као код огласа у главној активности само са додатим дугметом за измену и брисање огласа.



9.3. OglasAdapter2.java

Пошто имамо други recyclerView прави се други адаптер за приказ корисникових огласа који ради исто као први само са додатим функционалностима. Постављају се **listener-и** за **брисање и измену** огласа који ће служити за обавештавање друге активности да је дошло до промена у бази.

```
private OnOglasDeleteListener onOglasDeleteListener;
private OnOglasEditListener onOglasEditListener;
```

Постављају се **onClickListener-и** за дугмиће за измену и брисање огласа где се позивају методе **onOglasDelete()** и **onOglasEdit()** и прослеђује им се позиција огласа на који је корисник кликнуо. Остали део кода је исти као први адаптер.

```
       holder.btnDeleteOglas.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (onOglasDeleteListener != null) {
            onOglasDeleteListener.onOglasDelete(holder.getAdapterPosition());
        }
    }
});
holder.btnIzmeni.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (onOglasEditListener != null) {
            onOglasEditListener.onOglasEdit(holder.getAdapterPosition());
        }
```

9.4.MojNalog.java

Moj налог имплементира два интерфејса из адаптера, дефинише код за захтев приликом покретања измене огласа и ActivityResultLauncher за покретање измене огласа. Ако је резултат успешан позива се метода **updateOglasiList().**

```
       public class MojNalog extends AppCompatActivity implements
OglasiAdapter2.OnOglasDeleteListener,OglasiAdapter2.OnOglasEditListener {
    private static final int EDIT_OGLAS_REQUEST_CODE = 1;
    private TextView textViewKorisnickoIme;
    ...
    private final ActivityResultLauncher<Intent> editOglasLauncher =
registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == RESULT_OK) {
                    updateOglasiList();
                }
            }
    );
```

Узима се кориничко име из Intenta и поставља се у TextView, затим се преко добијеног корисничког имена добија листа огласа корисника помоћу методе **getOglasiByVlasnik()** из DatabaseHelper-a.

```
       Intent intent = getIntent();
...
String korisnickoIme = textViewKorisnickoIme.getText().toString();
List<Oglas> oglasiKorisnika=databaseHelper.getOglasiByVlasnik(korisnickoIme);
```

Након тога се RecycerView-у поставља адаптер са том листом огласа.

```
recyclerViewOglasi.setLayoutManager(new LinearLayoutManager(this));
oglasiAdapter2 = new OglasiAdapter2(this, oglasiKorisnika, this,this);
oglasiAdapter2.setOnOglasEditListener(this);
recyclerViewOglasi.setAdapter(oglasiAdapter2);
```

Када корисник кликне на дугме **buttonOdjava** за одјаву преусмерава се на прозор за логовање.

```
        buttonOdjava.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent odjavaintent=new Intent(MojNalog.this,LoginActivity.class);
        startActivity(odjavaintent);

    }
});
```

Имплементира се метода **onOglasDelete()** из адаптера за брисање огласа, добија се позиција огласа у листи и онда се он брише преко ид-а уз помоћ методе **deleteOglas()** из databaseHelper-a. Оглас се уклања из листе огласа и онда се адаптер обавештава о промени да би се приказ освежио. Поставља се резултат активности.

```
        @Override
public void onOglasDelete(int position) {
    Oglas oglas = oglasiAdapter2.getOglasiList().get(position);
    databaseHelper.deleteOglas(oglas.getId());
    oglasiAdapter2.getOglasiList().remove(position);
    recyclerViewOglasi.getAdapter().notifyItemRemoved(position);
    setResult(RESULT_OK);
    finish();
}
```

Имплементира се метода **onOglasEdit()** за измену огласа из адаптера, добија се позиција огласа и онда се корисник преусмерава на активност за измену огласа помоћу дефинисаног launcher-a којој се прослеђује ид изабраног огласа .

```
        @Override
public void onOglasEdit(int position) {
    Oglas oglas = oglasiAdapter2.getOglasiList().get(position);
    //POKRECE SE IZMENI OGLAS I PROSLEDJUJE ID OGLASA
    Intent editIntent = new Intent(this, IzmeniOglasActivity.class);
    editIntent.putExtra("OGLAS_ID", oglas.getId());
    editOglasLauncher.launch(editIntent);

}
```

Хвата се резултат активности, ако су измене успешне ажурира се листа огласа помоћу методе **updateOglasiList().**

```
        @Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == EDIT_OGLAS_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            updateOglasiList();
        }
    }
}
```

Метода **updateOglasiList()** узима корисничко име и листу огласа корисника и позива методу **onOglasChanged()** из интерфејса **onOglasChangeListener** чиме се обавештава главна активност да је дошло до промена. Поставља се нова листа огласа и обавештава адаптер да је дошло до промена.

```
        private void updateOglasiList() {
    String korisnickoIme = textViewKorisnickoIme.getText().toString();
    List<Oglas> oglasiKorisnika = databaseHelper.getOglasiByVlasnik(korisnickoIme);
    if (MainActivity.oglasChangeListener != null) {
        MainActivity.oglasChangeListener.onOglasChanged();
        oglasiAdapter2.setOglasiList(oglasiKorisnika);
        oglasiAdapter2.notifyDataSetChanged();
    }

}
```

9.5. OnOglasChangeListener

Овај интерфејс има методу **onOglasChaned()** и служи за комуникацију активности **измени оглас** са другим активностима, у овом случају **главном активности**. Имплементација је дата већ у коду и објашљена.

```
public interface OnOglasChangeListener {
    void onOglasChanged();
}
```

9.6. методе из DatabaseHelpera

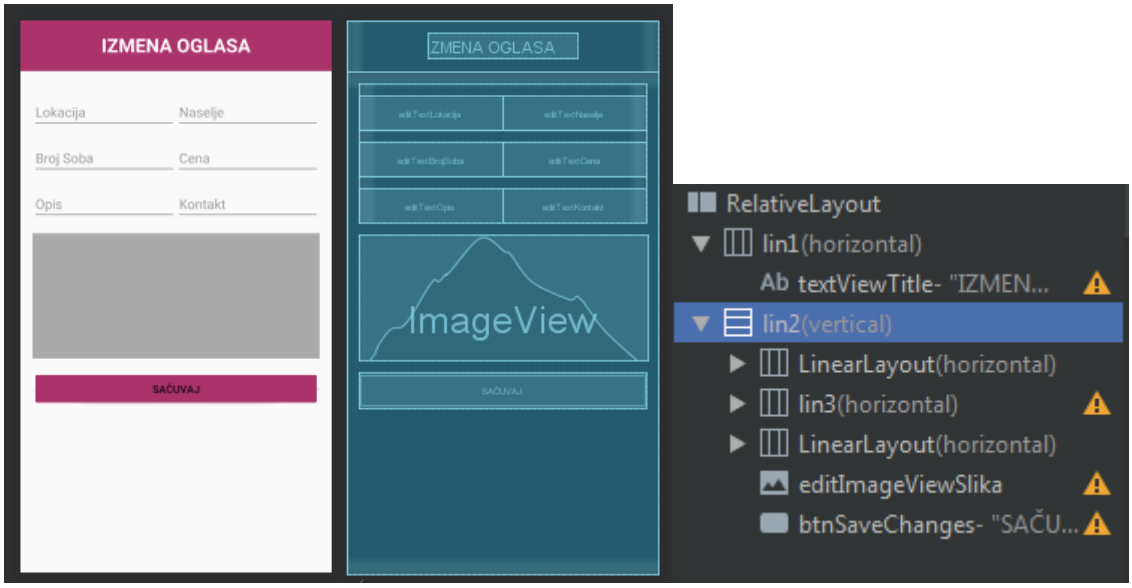
Метода **deleteOglas()** служи за брисање огласа. Прослеђује јој је ид огласа, добија се база у режиму за писање и онда се помоћу методе **delete()** брише оглас у табели Огласи чија је вредност ид-а једнака прослеђеном ид-у.

```
public void deleteOglas(long oglasId) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_OGLASI, COLUMN_ID + " = ?", new
String[]{String.valueOf(oglasId)});
    db.close();
}
```

10.ИЗМЕНА ОГЛАСА

10.1.izmeni_oglas.xml

Када корисник кликне на измену огласа преусмерава се на активност за измену огласа чији се дизајн састоји од наслова и одговарајућих EditText поља , приказа слике и дугмета за чување измена.



10.2. IzmeniOglas.java

Преко Intent-a се узима ид огласа и помоћу методе **getOglasById()** из DatabaseHelper-a узимају се информације огласа са тим ид-јем које се затим заједно са сликом која се декодира стављају у одговарајућа поља.

```
        Intent intent = getIntent();

             oglasId = intent.getLongExtra("OGLAS_ID",-1l);
             if (oglasId != -1) {

                 Oglas oglas = databaseHelper.getOglasById(oglasId);

                 editTextLokacija.setText(oglas.getLokacija());
                 ..
                 editTextKontakt.setText(oglas.getKontakt());

                 byte[] imageBytes = oglas.getSlika();
                 if (imageBytes != null) {
                     Bitmap bitmap = BitmapFactory.decodeByteArray(imageBytes,
0, imageBytes.length);
                     imageViewSlika.setImageBitmap(bitmap);
                 }
```

Кликом на дугме **btnSaveChanges** позива се метода **saveChanges()** за чување промена.

```
       btnSaveChanges.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        saveChanges();
    }
});
```

У методи **saveChanges()** се узимају унете вредности и методи **updateOglasi()** из DatabaseHelper-a се те вредности прослеђују на ажурирање огласа. Поставља се резултат и обавештава се активност мој налог о променама.

```
private void saveChanges() { .......
    databaseHelper.updateOglas(oglasId, lokacija, naselje, cena,
brojSoba,opis,kontakt);
    Intent resultIntent = new Intent();
    setResult(RESULT_OK, resultIntent);
    finish();
}
```

10.3. методе из DatabaseHelpera

Методи **getOglasByID()** се прослеђује ид огласа и добија се база у читљивом режиму. Помоћу Cursor-a и упита се узимају потребне вредности из табеле Огласи и придружују објекту Оглас који се враћа као резултат методе. Ова метода узима оглас који корисник жели да измени.

```
public Oglas getOglasById(long oglasId) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(
            TABLE_OGLASI,
            new String[]{COLUMN_ID, COLUMN_LOKACIJA, COLUMN_NASELJE,
COLUMN_CENA, COLUMN_BROJ_SOBA, COLUMN_OPIS,COLUMN_KONTAKT, COLUMN_SLIKA},
            COLUMN_ID + " = ?",
            new String[]{String.valueOf(oglasId)},
            null, null, null, null);

    Oglas oglas = null;

    if (cursor != null && cursor.moveToFirst()) {
        oglas = new Oglas(
                cursor.getLong(cursor.getColumnIndex(COLUMN_ID)),
                ...
                cursor.getString(cursor.getColumnIndex(COLUMN_OPIS)),
                cursor.getString(cursor.getColumnIndex(COLUMN_KONTAKT)),
                null,
                cursor.getBlob(cursor.getColumnIndex(COLUMN_SLIKA))
        );
    }

    if (cursor != null) {
        cursor.close();
    }

    db.close();

    return oglas;
}
```

Метода **updateOglas()** служи за ажурирање огласа и као параметре прима информације о огласу. Затим узима базу у режиму писања и помоћу ContentValues везује колоне са новим вредностима. Након тога, помоћу методе **update()** ажурира тај ред који је измењен у табели Огласи.

```
public void updateOglas(long oglasId, String lokacija, String naselje, int
cena, int brojSoba, String opis, String kontakt) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(COLUMN_LOKACIJA, lokacija);
    values.put(COLUMN_NASELJE, naselje);
    values.put(COLUMN_CENA, cena);
    values.put(COLUMN_BROJ_SOBA, brojSoba);
   values.put(COLUMN_OPIS,opis);
   values.put(COLUMN_KONTAKT,kontakt);

    db.update(TABLE_OGLASI, values, COLUMN_ID + " = ?", new
String[]{String.valueOf(oglasId)});

    db.close();
}
```

11.ЗАКЉУЧАК

У семинарском раду је приказана и објашњена имплементација пројекта „Стамбени огласи“. На примеру је показано како Андроид студио пружа лак и брз дизајн корисничког интерфејса и велик скуп различитих функционалности.

ЛИТЕРАТУРА

[1] https://developer.android.com

[2] https://www.geeksforgeeks.org