

# Программа Android-разработчик

## Конспект

The smartphone screen shows a navigation bar with a menu icon, search icon, and three-dot menu icon. The main content area has a teal header with the title 'Быстрый старт в Android-разработку' and a 'Часть 3' button. Below the header are five sections listed vertically:

- Многопоточность и сетевое взаимодействие
- Архитектура Android-приложений
- Тестирование и работа с библиотеками
- Дизайн и анимации
- Облачные сервисы и периферия

# Оглавление

<b>3 Старт курсового проекта(КП)</b>	<b>2</b>
3.1 Activity авторизации . . . . .	2
3.1.1 КП. Вёрстка экрана логина . . . . .	2
3.1.2 КП. Добавление ссылок на View элементы . . . . .	4
3.1.3 КП. Валидация email и password. Ошибки в Toast . . . . .	7
3.2 Activity профиля . . . . .	10
3.2.1 КП. Верстка экрана профиля . . . . .	10
3.2.2 КП. Создание активити профиля . . . . .	12
3.2.3 КП. Создание класса User . . . . .	15
3.3 Добавление фрагментов . . . . .	18
3.3.1 КП. Создание хост активити для фрагментов . . . . .	18
3.3.2 КП. Миграция логики AuthActivity во фрагмент . . . . .	19
3.3.3 КП. Добавление фрагмента регистрации. Создание класса PreferenceHelper . . . . .	23

# Глава 3

## Старт курсового проекта(КП)

Начнём разрабатывать android-приложение со стартовым функционалом, который может понадобиться в других приложениях. Оно будет иметь три экрана — это экран авторизации, экран регистрации и экран показа профиля.

[Архив с кодом для экрана логина](#)

[Архив с кодом добавления ссылок в activity](#) [Архив с валидацией email и password](#)

### 3.1. Activity авторизации

#### 3.1.1. КП. Вёрстка экрана логина

Создадим новый проект, как на первой неделе. Start a new Android Studio project. Назовём MyFirstApplication (мы делали с company domain = elegion.com). Next. Phone and Tablet API 19. Next. Empty Activity. Next. Activity Name = MainActivity (generate layout file и Backwards Compatibility оставляем включенными). Finish.

Проект создался. Открываем app res layout и переименуем (shift+F6 по нему) activity\_main.xml в ac\_auth.xml. Откроем его во вкладке xml. И подчистим его так, чтобы осталось только:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout>
3     xmlns:android="http://schemas.android.com/apk/res/android"
4         android:layout_width="match_parent"
5         android:layout_height="match_parent"
6     </LinearLayout>
```

И добавим внутрь два поля TextView: для логина и пароля:

```
1 <EditText
2     android:id="@+id/etLogin"
3     android:layout_width="match_parent"
4     android:layout_height="wrap_content"
5     android:layout_marginLeft="8dp"
6     android:layout_marginRight="8dp"
7     android:hint="@string/login_hint"/>
8 <EditText
9     android:id="@+id/etPassword"
10    android:layout_width="match_parent"
11    android:layout_height="wrap_content"
12    android:layout_marginLeft="8dp"
13    android:layout_marginRight="8dp"
14    android:hint="@string/login_password"/>
```

layout\_marginLeft и layout\_marginRight - отступы слева и справа. hint - подсказка, причём помним, что надо создать строковый ресурс и сослаться на него, поэтому в res values strings.xml допишем:

```
1 <resources>
2     <string name="app_name">MyFirstApplication</string>
3     <string name="login_hint">Login</string>
4     <string name="login_password">Password</string>
5     <string name="login_enter">Sign in</string>
6     <string name="login_register">Sign up</string>
7 </resources>
```

Кроме того, это можно было сделать комбинацией Alt+Enter по строке, которую надо добавить в строковые ресурсы (с.м. 1 неделю). Заметим, что отображается только одно поле. Мы забыли добавить ориентацию нашему layout. Добавим её перед полями EditText:

```
1     android:orientation="vertical">
```

Теперь перейдём к созданию кнопок и регистрации в приложении. Для этого надо добавить LinearLayout:

```
1 <LinearLayout  
2     android:layout_width="match_parent"  
3     android:layout_height="wrap_content"  
4     android:orientation="horizontal">  
5     <Button  
6         android:id="@+id/buttonEnter"  
7         android:layout_width="match_parent"  
8         android:layout_height="wrap_content"  
9         android:layout_weight="1"  
10        android:text="@string/login_enter"/>  
11    <Button  
12        android:id="@+id/buttonRegister"  
13        android:layout_width="match_parent"  
14        android:layout_height="wrap_content"  
15        android:layout_weight="1"  
16        android:text="@string/login_register"/>  
17 </LinearLayout>
```

Не забываем добавить orientation, которая на этот раз горизонтальная - мы хотим две кнопочки рядом. Обеим кнопкам ставим вес =1 (android:layout\_weight="1"). Кроме того, не забудем задать кнопкам их уникальные id (в соответствии с их назначениями buttonEnter и buttonRegister). Значения кнопок соответственно войти - sign in и зарегистрироваться sign up (помним про строковые ресурсы).

### 3.1.2. КП. Добавление ссылок на View элементы

Рассмотрим, как присоединить наш макет к Activity, и в конце видео запустим это на эмуляторе. Зайдем в наш проект. Откроем package app java внутренний package. Увидим MainActivity, который нам автоматически создал Android Studio. В сочетании клавиш Shift + F6 переименуем его в AuthActivity. Нажмем Refractor и откроем его.

```
1 package com.elegion.myfirstapplication;
2 import android.os.Bundle;
3 import android.support.annotation.Nullable;
4 import android.support.v7.app.AppCompatActivity;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.EditText;
8
9 public class AuthActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(@Nullable Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.ac_auth);
15     }
16 }
```

Здесь, как мы видим, у нас есть метод `onCreate()`, и используется метод `setContentView()`. Мы хотим, чтобы наши view присоединялись к нашим объектам в коде. Давайте откроем наш layout. Как мы видим, у нас есть здесь два id-текста и два button. Соответственно, создадим эти же поля внутри нашей Activity. После этого мы сможем обращаться к view по ссылкам. Сразу добавим слушателей: кнопки `mEnter` и `mRegistration`. Эти слушатели будут вызываться при нажатии на кнопки.

```
1 public class AuthActivity extends AppCompatActivity {
2 //создаём каждый объект
3     private EditText mLogin;
4     private EditText mPassword;
5     private Button mEnter;
6     private Button mRegister;
7     //слушатель для кнопки входа
8     private View.OnClickListener mOnEnterClickListener
9             = new View.OnClickListener() {
10         @Override
11         public void onClick(View view) {
12             //todo Обработка нажатия по кнопке
13         }
14     };
15     //слушатель для кнопки регистрации
16     private View.OnClickListener mOnRegisterClickListener
17             = new View.OnClickListener() {
18         @Override
19         public void onClick(View view) {
20             //todo Обработка нажатия по кнопке
21         }
22     };
23     protected void onCreate(@Nullable Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         setContentView(R.layout.ac_auth);
26         //Делаем так, чтобы можно было обращаться к view по ссылкам
27         mLogin = findViewById(R.id.etLogin);
28         mPassword = findViewById(R.id.etPassword);
29         mEnter = findViewById(R.id.buttonEnter);
30         mRegister = findViewById(R.id.buttonRegister);
31
32         mEnter.setOnClickListener(mOnEnterClickListener);
33         mRegister.setOnClickListener(mOnRegisterClickListener);
34     }
35 }
```

Теперь запускаем наш эмулятор (кнопочка run):

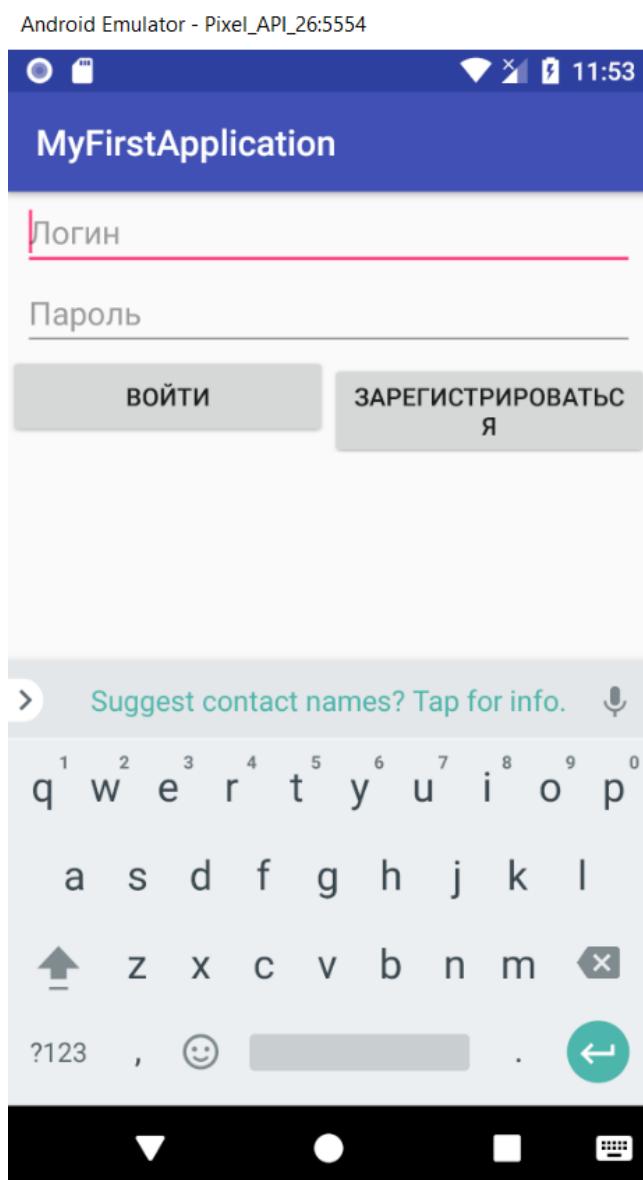


Рис. 3.1

### 3.1.3. КП. Валидация email и password. Ошибки в Toast

Добавим логику обработки данных логина, который по факту будет email'ом, пароля и нажатия на кнопку «Войти». Откроем AuthActivity, перейдём в метод onClick в mOnEnterClickListener, вместо «todo» добавим логику нажатия по кнопке. Для начала проверим email и добавим метод:

```
1 private boolean isEmailValid() {
2     return !TextUtils.isEmpty(mLogin.getText())
3             && Patterns.EMAIL_ADDRESS.matcher(mLogin.getText()).matches();
4 } //проверка на непустоту и на правильность паттерна емейла
```

Перепишем немного OnClickListener:

```
1 private View.OnClickListener mOnEnterClickListener
2                     = new View.OnClickListener() {
3             @Override
4             public void onClick(View view) {
5                 if (isEmailValid()) {
6                     //если email подходит, можем перейти в приложение
7                 }
8             }
9         };
```

Аналогично создаём isPasswordValid():

```
1 private boolean isPasswordValid() {
2     return !TextUtils.isEmpty(mPassword.getText());
3 } //берём текст из поля mPassword на совпадение с базой
```

Так же не забудем изменить соответствующего слушателя:

```
1 private View.OnClickListener mOnEnterClickListener
2                     = new View.OnClickListener() {
3             @Override
4             public void onClick(View view) {
5                 if (isEmailValid() && isPasswordValid()) {
6                     // переход в приложение
7                 } else {
8                     showMessage(R.string.login_input_error);
9                 }
10            }
11        };
```

Мы добавили ошибку. Создадим тост, который будет вылетать с ошибкой при регистрации:

```
1 private void showMessage(@StringRes int string) {  
2     Toast.makeText(this, string, Toast.LENGTH_LONG).show();  
3 } //показываем сообщение об ошибке с помощью тоста
```

Не забываем передавать контекст - this. LENGTH\_LONG потому что хотим показывать сообщение долго. Создадим строковый ресурс:

```
1 <string name="login_input_error">Error!!</string>
```

Теперь если мы запустим приложение и ничего не введём хотя бы одно из полей (или не валидный email), вылетит тост с ошибкой. Если же всё хорошо, то пока ничего не произойдёт.

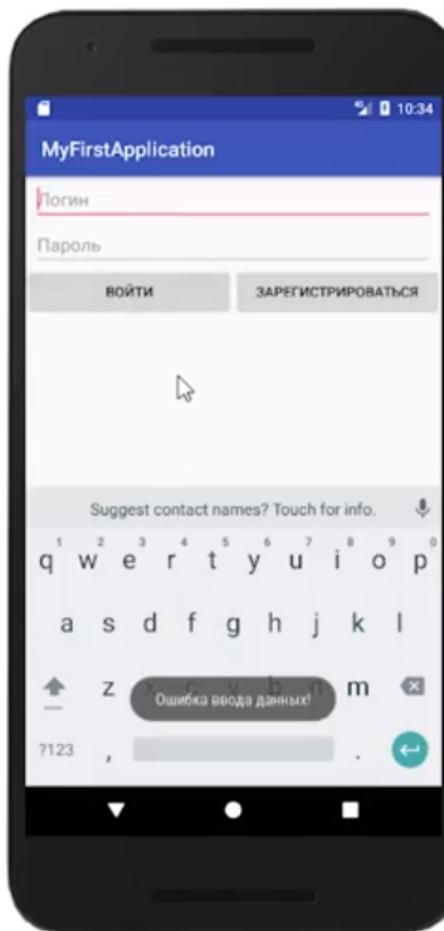


Рис. 3.2



Рис. 3.3

Но пароль отображает буквы! Для того, чтобы вместо букв были звёздочки, перейдём к ac\_auth.xml и введём в соответствующий вводу пароля EditText строчку:

```
1    android:inputType="textPassword"/>
```

Теперь сразу появляются точки и пароль не отображается.

## 3.2. Activity профиля

### 3.2.1. КП. Верстка экрана профиля

Создадим макет экрана профиля, в который можно попасть после успешной авторизации. Зададим в на проекте res layout. Правой кнопкой по layout new Layout resource file. И назовём его ac\_profile. Enter. Переходим во вкладку text

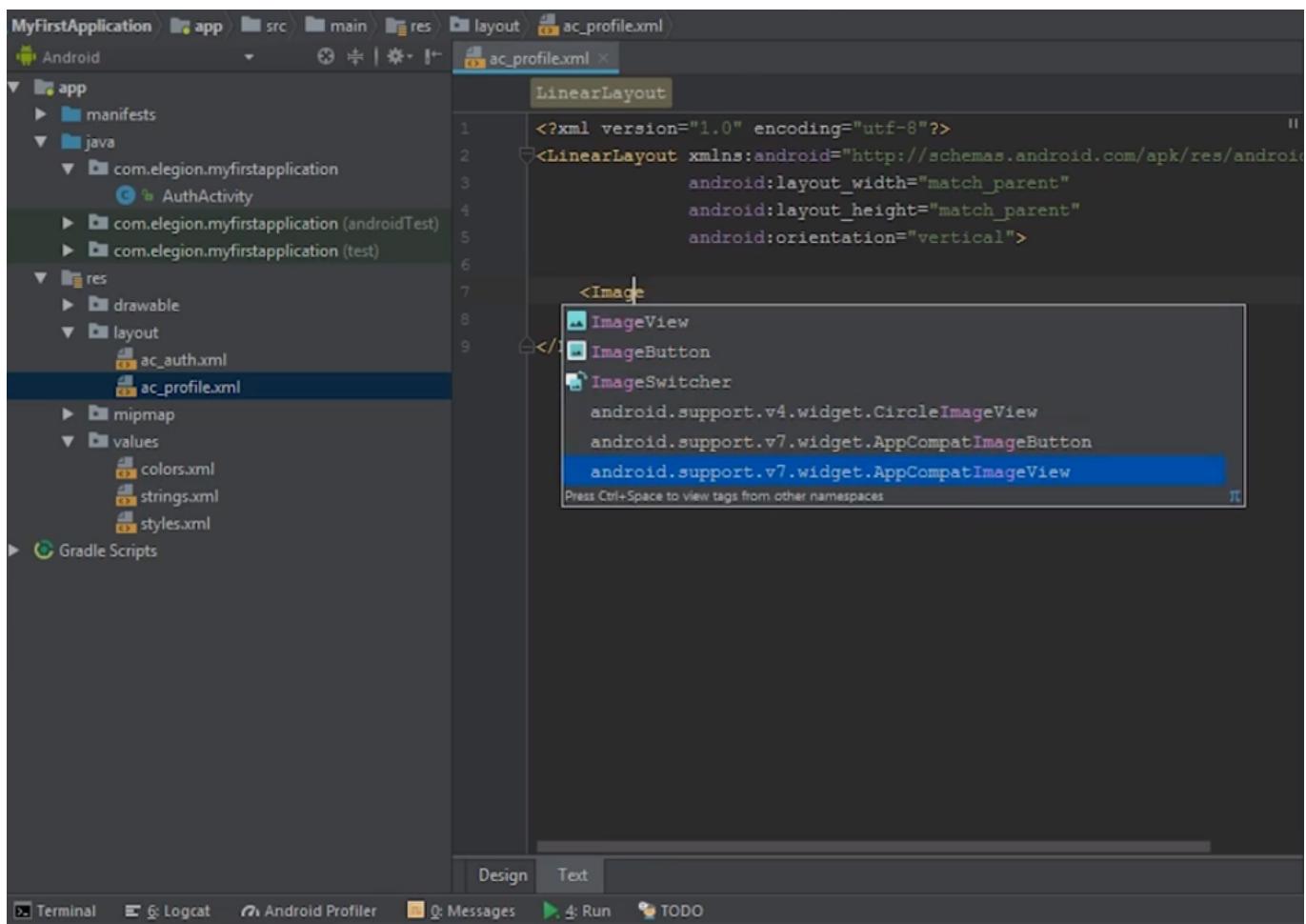


Рис. 3.4

И допишем туда параметры layout\_width, layout\_height и отступы со всех сторон (layout\_margin).

Ориентация будет горизонтальной.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4          android:layout_width="match_parent"
5          android:layout_height="match_parent"
6          android:orientation="horizontal">
7      <android.support.v7.widget.AppCompatImageView
8          android:id="@+id/ivPhoto"
9          android:layout_width="100dp"
10         android:layout_height="100dp"
11         android:layout_margin="16dp"/>
```

Далее допишем LinearLayout и в нём два TextView с полями - отображаемые email и пароль. И зададим первому текст. А второму будем передавать значение самого емейла, для этого дадим ему id. Самому же LinearLayout зададим два отступа - от верха (layout\_marginTop) и от правого края (layout\_marginRight)

```
1      <LinearLayout android:layout_width="match_parent"
2          android:layout_height="wrap_content"
3          android:layout_marginRight="16dp"
4          android:layout_marginTop="16dp"
5          android:orientation="vertical">
6      <TextView android:layout_width="match_parent"
7          android:layout_height="wrap_content"
8          android:text="@string/email"/>
9      <TextView android:id="@+id/tvEmail"
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"/>
12      <TextView android:layout_width="match_parent"
13          android:layout_height="wrap_content"
14          android:text="@string/password"/>
15      <TextView android:id="@+id/tvPassword"
16          android:layout_width="match_parent"
17          android:layout_height="wrap_content"/>
18      </LinearLayout>
19  </LinearLayout>
```

Не забываем, что каждая строка должна быть отдельным строковым ресурсом. Тогда наш res

strings.xml будет выглядеть так:

```
1 <resources>
2     <string name="app_name">MyFirstApplication</string>
3     <string name="login_hint">Login</string>
4     <string name="login_password">Password</string>
5     <string name="login_enter">Sign in</string>
6     <string name="login_register">Sign up</string>
7 </resources>
```

### 3.2.2. КП. Создание активити профиля

Настроим запуск экрана профиля через нажатие кнопки "Log in". Пока что наш ProfileActivity.java выглядит так:

```
1 public class ProfileActivity extends AppCompatActivity {
2     private AppCompatImageView mPhoto; //именно AppCompat
3     private TextView mLogin;
4     private TextView mPassword;
5     private View.OnClickListener mOnPhotoClickListener
6             = new View.OnClickListener() {
7         @Override
8             public void onClick(View view) {
9             }
10    };
11    @Override
12    protected void onCreate(Bundle savedInstanceState) {
13        super.onCreate(savedInstanceState);
14        setContentView(R.layout.ac_profile);
15        //далее инициализируем все поля (находим их по id)
16        mPhoto = findViewById(R.id.ivPhoto);
17        mLogin = findViewById(R.id.tvEmail);
18        mPassword = findViewById(R.id.tvPassword);
19        mPhoto.setOnClickListener(mOnPhotoClickListener);
20    }
21 }
```

Мы забыли добавить id для Photo и поэтому R.id.ivPhoto подсвечивается красным. Для этого в ac\_profile.xml перед android:layout\_width надо добавить:

```
1 android:id="@+id/ivPhoto"
```

Чтобы не было проблем с совместимостью надо писать AppCompatImageView. Наш UI проинициализирован.

Теперь давайте попробуем передать некоторые параметры и запустить наш ProfileActivity. Для этого допишем в начало ProfileActivity.java строки:

```
1 public static final String EMAIL_KEY = "EMAIL_KEY";
2 public static final String PASSWORD_KEY = "PASSWORD_KEY";
```

Для этого перейдём в AuthActivity. По нажатию на кнопку входа при правильно введённом email и пароле мы будем его запускать; для этого мы будем использовать интенты. Для этого в View.OnClickListener допишем:

```
1 if (isValidEmail() && isValidPassword()) {
2     Intent startProfileIntent = //класс, к которому переходим
3         new Intent(AuthActivity.this, ProfileActivity.class);
4     startProfileIntent.putExtra(ProfileActivity.EMAIL_KEY,
5         mEmail.getText().toString());
6     startProfileIntent.putExtra(ProfileActivity.PASSWORD_KEY,
7         mPassword.getText().toString());
8     startActivity(startProfileIntent);
9 } else {
10     showMessage(R.string.login_input_error);
11 }
```

Получить переданные данные, можно методом getIntent() в ProfileActivity. И создадим бандл, в который передадим extra:

```
1 Bundle bundle = getIntent().getExtras();
2 mEmail.setText(bundle.getString(EMAIL_KEY));
3 mPassword.setText(bundle.getString(PASSWORD_KEY));
4 //устанавливаем значения логина и пароля из бандла
```

Важный момент, чтобы ProfileActivity работала, её обязательно надо добавить в манифест, который будет выглядеть как-то так.

```
1   ...
2   <application
3       android:allowBackup="true"
4       android:icon="@mipmap/ic_launcher"
5       android:label="@string/app_name"
6       android:roundIcon="@mipmap/ic_launcher_round"
7       android:supportsRtl="true"
8       android:theme="@style/AppTheme">
9           <activity android:name=".AuthActivity">
10              <intent-filter>
11                  <action android:name="android.intent.action.MAIN"/>
12                  <category android:name="android.intent.category.LAUNCHER"/>
13              </intent-filter>
14          </activity>
15          <activity android:name=".ProfileActivity">
16          </activity>
17      </application>
18  </manifest>
```

Запускаем приложение и пробуем ввести что-нибудь. После успешного ввода переходим на экран профиля.

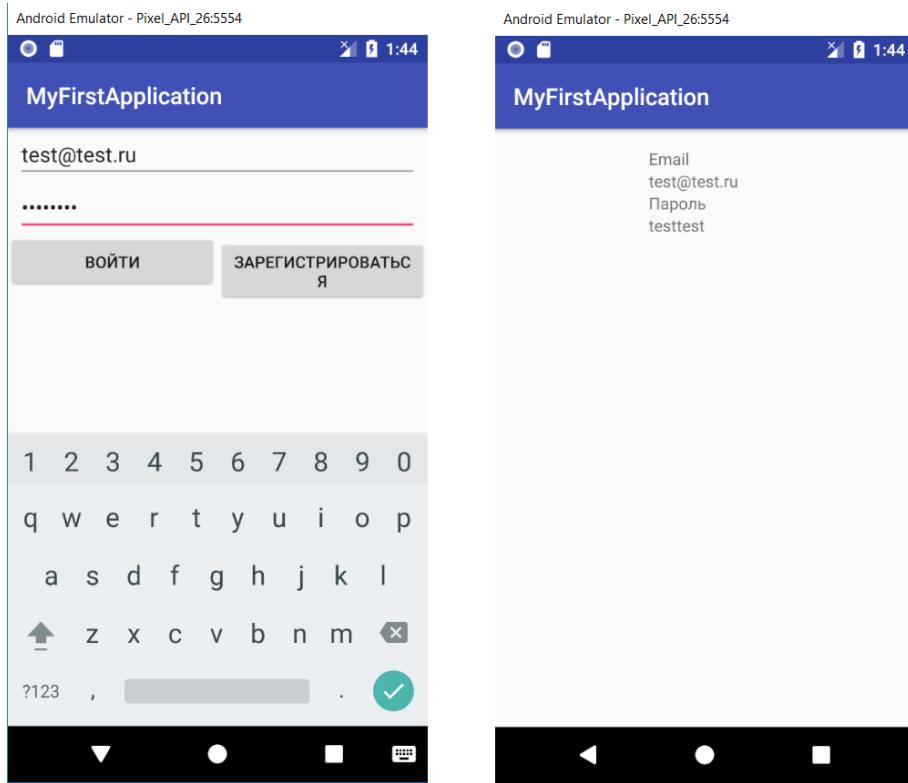


Рис. 3.5

[Архив с проектом](#), в котором работают log in и ProfileActivity

### 3.2.3. КП. Создание класса User

Теперь давайте заменим эти параметры и добавим класс User в наш проект. Откроем Android Studio package app java и создадим New Java Class. Назовем его User. В данный класс User добавим два поля — private String email, пусть будет mLogin, и private String mPassword. Сочетанием клавиш Alt + Insert добавим Getter и Setter для логина и пароля. И добавим Constructor, который принимает в себя логин и пароль. В итоге файл класс User будет выглядеть так:

```
1 package com.elegion.myfirstapplication;
2 public class User {
3     private String mLogin;
4     private String password;
5     public User(String mLogin, String password) {
6         this.mLogin = mLogin;
7         this.password = password;
8     } //конструктор, принимающий в себя логин и пароль
9
10    public String getLogin() {
11        return mLogin;
12    } //Getter логина
13
14    public String getPassword() {
15        return password;
16    } //Getter пароля
17
18    public void setLogin(String mLogin) {
19        this.mLogin = mLogin;
20    } //Setter логина
21
22    public void setPassword(String password) {
23        this.password = password;
24    } //Setter пароля
25 }
```

Далее, перейдем в AuthActivity и вместо того, чтобы передавать сюда email и пароль по отдельности, будем передавать User. Для этого перейдем в ProfileActivity, удалим PASSWORD\_KEY, а EMAIL\_KEY переделаем под USER\_KEY.

```
1 //e profileActivity.java
2 public class ProfileActivity extends AppCompatActivity {
3     public static final String USER_KEY = "USER_KEY";...
```

Do Refactor (заменяем везде, где встречалось). Перейдем в AuthActivity. Удалим строку с password. В putExtra добавим new User. И в качестве передаваемых параметров передадим в него логин, который мы возьмем из mLogin.getText().toString(), и mPassword.getText().toString(). [БЕЗ\_ЗВУКА] Android Studio ругается и не может понять, что мы хотим передать. Все правильно. Ведь мы забыли указать то, что класс User является serializable. Давайте исправим это.

```
1 public class User implements Serializable
```

И вот Android Studio перестала ругаться. С учётом изменений, AuthActivity будет выглядеть так

```
1 private View.OnClickListener mOnEnterClickListener =
2                     new View.OnClickListener() {
3                         @Override
4                         public void onClick(View view) {
5                             if (isValidEmail() && isValidPassword()) {
6                                 Intent startProfileIntent =
7                                     new Intent(
8                                         AuthActivity.this, ProfileActivity.class);
9                                 startProfileIntent.putExtra(ProfileActivity.USER_KEY,
10                                 new User(mLogin.getText().toString(),
11                                         mPassword.getText().toString()));
12                                 startActivity(startProfileIntent);
13                             } else {
14                                 showMessage(R.string.login_input_error);
15                             }
16                         }
17                     };
```

Давайте посмотрим, как это работает, вернее, исправим это в ProfileActivity. Ведь мы получаем bundle, в котором мы должны получить не строку, а уже пользователя. Соответственно, создадим его и передаем ему наш ключ

```
1 // в методе onCreate() после строк с mPassword
2         Bundle bundle = getIntent().getExtras();
3         User user = (User) bundle.get(USER_KEY);
4         mLogin.setText(user.getLogin());
5         mPassword.setText(user.getPassword());
```

И с помощью приведения типов превращаем его в User. mLogin поменяем на mPassword, а внутреннее содержимое поменяем на user.getLogin, и user.getPassword. Готово. Всё работает так же, как и в предыдущем случае (рис. 3.5)

## 3.3. Добавление фрагментов

### 3.3.1. КП. Создание хост активити для фрагментов

В последующих уроках мы с вами будем использовать фрагменты. Но для начала напишем базовый класс для того, чтобы в нашей Activity был всего один фрагмент. Откроем проект. app java New Java Class. Назовем нашу активность SingleFragmentActivity. Ok.

```

1  public abstract class SingleFragmentActivity
2      extends AppCompatActivity {//наследовали
3
4      @Override
5      protected void onCreate(@Nullable Bundle
6          savedInstanceState){//переопределели Create
7          super.onCreate(savedInstanceState);
8          if (savedInstanceState != null) { //добавленная логика
9              FragmentManager fragmentManager =//обязательно support
10                 getSupportFragmentManager();
11             //начнём транзацию по запуску фрагмента
12             fragmentManager.beginTransaction()
13                 .replace(R.id.fragmentContainer, getFragment())
14                 .commit();
15         }
16     }

```

В этом случае нам нужен контейнер ViewId. Давайте его добавим. Идем res layout New Layout resource file. и назовём его ac\_single\_fragment. Ok. Переходим во вкладочку «Текст» и сюда вместо LinearLayout добавляем самый простой FrameLayout и добавим ему id. Назовем его fragmentContainer.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout xmlns:android="http://schemas.android.com
3                  /apk/res/android"
4                  android:id="@+id/fragmentContainer"
5                  android:layout_width="match_parent"
6                  android:layout_height="match_parent"
7                  android:orientation="vertical"/>

```

Перейдем в SingleFragmentActivity. Добавим setContentView(R.layout.ac\_single\_fragment) после

super.onCreate() д. Наш Android Studio опять-таки не видит вновь добавленный ресурс, поэтому синхронизируемся. Ресурсы успешно добавились.

Добавим наш фрагмент. FragmentManager.beginTransaction().replace(R.id.fragmentContainer) — это то, куда мы будем добавлять наш фрагмент. Название фрагмента. Так как это базовый класс, у нас нет конкретного фрагмента, поэтому создадим абстрактный метод.

```
1 public class SingleFragmentActivity
2     extends AppCompatActivity {//унаследовали
3     @Override
4     protected void onCreate(@Nullable Bundle
5         savedInstanceState){//переопределели Create
6         super.onCreate(savedInstanceState);           //добавим ContentView
7         setContentView(R.layout.ac_single_fragment);
8         if (savedInstanceState != null) { //добавленная логика
9             FragmentManager fragmentManager =//обязательно support
10            getSupportFragmentManager();
11            //начнём транзацию по запуску фрагмента
12            fragmentManager.beginTransaction()
13                .replace(R.id.fragmentContainer, getFragment())
14                .commit();//после реплейса вызвали commit
15        }
16    }//абстрактный метод получения фрагмента
17    protected abstract Fragment getFragment();
18 }
```

Наша Android Studio ругается, что мы не назвали наш класс абстрактным. Исправим это:

```
1 public abstract class SingleFragmentActivity ...
```

Соответственно, в метод replace передадим наш getFragment. И по желанию мы можем передать тег. Этого мы делать не будем. Вот и все. Базовый класс для одного фрагмента готов.

### 3.3.2. КП. Миграция логики AuthActivity во фрагмент

На прошлых занятиях мы с вами написали Activity для одного фрагмента. Давайте теперь изменим AuthActivity на AuthFragment.

Зайдем в SingleFragmentActivity в условие if savedInstanceState не равно null. Это неправильно. Наша транзакция должна выполняться, когда savedInstanceState равен null.  
После всех изменений наш SingleFragmentActivity должен будет выглядеть так:

```
1 public abstract class SingleFragmentActivity extends AppCompatActivity {
2     @Override
3     protected void onCreate(@Nullable Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.ac_single_fragment);
6         if (savedInstanceState == null) { //изменили на равенство
7             FragmentManager fragmentManager = getSupportFragmentManager();
8             fragmentManager.beginTransaction()
9                 .replace(R.id.fragmentContainer, getFragment())
10                .commit();
11         }
12     }
13     protected abstract Fragment getFragment();
14 }
```

Теперь перейдем в AuthActivity. Нажимаем на него и с помощью сочетаний Shift + F6 меняем на AuthFragment и экстендишем его от Fragment (с комментарием android.support.v4.app).

```
1 public class AuthFragment extends Fragment {
```

Android Studio подсвечивает некоторые блоки красным. Давайте посмотрим, почему, и попытаемся это исправить.

Заходим в java com.elegion AuthFragment.java и видим, что AuthFragment.this не является контекстом, а Activity является. Соответственно, заменим его на метод getActivity(). Теперь все работает. Скопируем его, он нам в будущем пригодится. То же самое с showMessage.

```
1 //было:
2 new Intent(AuthFragment.this, ProfileActivity.class);
3 //заменили на
4 new Intent(getActivity(), ProfileActivity.class);
5 //аналогично чуть дальше в showMessage было:
6 Toast.makeText(this, string, Toast.LENGTH_LONG).show();
7 //стало
8 Toast.makeText(getActivity(), string, Toast.LENGTH_LONG).show();
```

В том же AuthActivity, видим что метод onCreate() отличается от того метода, который есть в Activity. Вместо этого в фрагменте используется метод onCreateView(). Давайте заменим метод onCreate() на onCreateView() (когда начинаем вводить, в контекстном меню выбираем нужный нам onCreateView()) Копируем все, что у нас было в методе onCreate, и удаляем его. Метод setContentView() нам не пригодится. Вместо этого нам пригодится такая строка "View v = inflater.inflate (R.layout.ac\_auth)". Сразу же ac\_auth переименуем в fr\_auth. Далее, передаем container и в качестве Boolean parameter передаем false. Далее, у нас нет метода findViewById() внутри Activity. Но зато он есть у нашей View, которую мы заинфлейтили. Соответственно, делаем везде v.findViewById(). После всех преобразований получим:

```
1 //вместо onCreate() получили onCreateView() после ShowMessage
2     @Nullable
3     @Override
4         public View onCreateView(LayoutInflater inflater,
5                             @Nullable ViewGroup container,
6                             @Nullable Bundle savedInstanceState) {
7             View v = inflater.inflate(R.layout.fr_auth,
8                                     container, false);
9             //везде поменяли на v.findViewById
10            mLogin = v.findViewById(R.id.etLogin);
11            mPassword = v.findViewById(R.id.etPassword);
12            mEnter = v.findViewById(R.id.buttonEnter);
13            mRegister = v.findViewById(R.id.buttonRegister);
14            mEnter.setOnClickListener(mOnEnterClickListener);
15            mRegister.setOnClickListener(
16                mOnRegisterClickListener);
17            return v;
18        }
19    }
```

Все вроде бы должно работать, за исключением одного но. Когда мы переименовывали наш AuthFragment в AuthActivity, он переименовался также в манифесте. Давайте зайдем туда и исправим это.

```
1 <activity android:name=".AuthActivity">
```

AuthFragment становится AuthActivity, которой у нас нет. Знаете, почему? Потому что её нужно создать. Заходим в наш package, создаем Java Class, называем его AuthActivity.

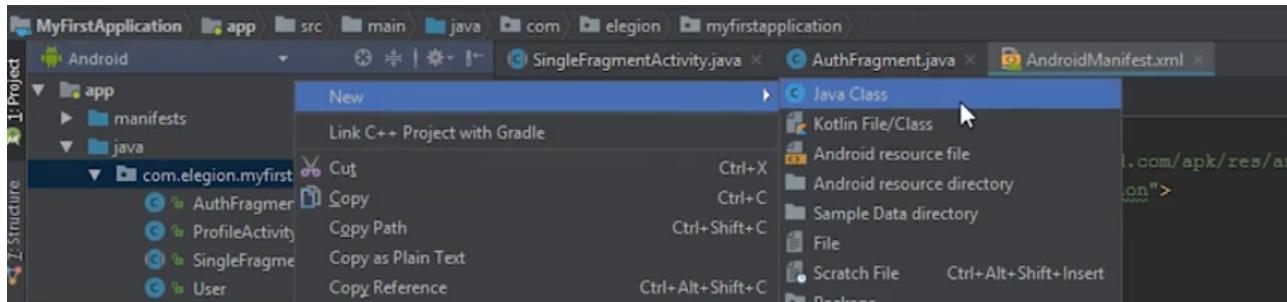


Рис. 3.6

extends SingleFragmentActivity, который мы написали до этого.

```
1 public class AuthActivity extends SingleFragmentActivity{
2     @Override
3     protected Fragment getFragment() {
4         return null; //потом изменим
5     }
6 }
```

И, соответственно, он просит нас реализовать один метод. Давайте сделаем это. В качестве возвращаемого параметра в GetFragment() мы должны передать instance фрагмента, который должен показываться на экране. Перед этим нужно создать метод, который будет создавать этот instance. Соответственно, нужно перейти в AuthFragment и создать метод newInstance() (сам сгенерируется когда начнём вводить newInstance), его мы поменяем потом.

```
1 public static AuthFragment newInstance() {
2     Bundle args = new Bundle();
3     AuthFragment fragment = new AuthFragment();
4     fragment.setArguments(args);
5     return fragment;
6 }
```

Вернемся в AuthActivity и передадим ему AuthFragment.newInstance.

```
1 protected Fragment getFragment() {
2     return AuthFragment.newInstance(); //поменяли
3 }
```

Вот и все. Но в эмуляторе пустой экран. А все потому, что в AuthFragment, в onCreateView в качестве возвращаемого параметра мы забыли вернуть нашу View, в которую мы успешно зainфлейтили.

```
1     return super.onCreateView(inflater, container,
2                               savedInstanceState); //было
3
4     return v; //стало
```

Запускаем и всё работает.

Можем даже попробовать ввести e-mail и перейти на следующий экран, чтобы убедиться, что мы ничего не сломали. Нажмем кнопочку «войти». Все прекрасно работает как и в [3.5](#).

Мы успешно использовали SingleFragmentActivity class, который обычно пригождается, когда нам нужно использовать один фрагмент в качестве основного.

### 3.3.3. КП. Добавление фрагмента регистрации.

#### Создание класса PreferenceHelper

В предыдущих занятиях мы с вами создали экран авторизации и экран показа профиля. Но откуда же возьмутся пользователи? Для этого нам нужно создать экран регистрации.

[Мы предварительно подготовили для вас весь UI и его инициализацию.](#)

Нам осталось только добавить логику. Но перед этим давайте посмотрим, что у нас уже есть. Откроем проект, перейдём в RegistrationFragment, увидим четыре поля: mLogin, mPassword, PasswordAgain и Registration. Первые три — EditText и кнопка Registration. Посмотрим, как он выглядит в XML

```
1 //начало RegistrationFragment.java
2 public class RegistrationFragment extends Fragment {
3     private EditText mLogin; //email
4     private EditText mPassword; //введите пароль
5     private EditText mPasswordAgain; //введите пароль ещё раз
6     private Button mRegistration; //зарегистрироваться
7
8     public static RegistrationFragment newInstance() {
9         return new RegistrationFragment();
10    }
11    private View.OnClickListener mOnRegistrationClickListener
12        = new View.OnClickListener() {
13            @Override //здесь будет вся логика регистрации
14            public void onClick(View view) {
15                }
16            };

```

Ниже представлена xml разметка:

```
1 <LinearLayout  
2     android:layout_width="match_parent"  
3     android:layout_height="wrap_content"  
4     android:orientation="vertical">  
5  
6     <EditText  
7         android:id="@+id/etLogin"  
8         android:layout_width="match_parent"  
9         android:layout_height="wrap_content"  
10        android:layout_marginLeft="16dp"  
11        android:layout_marginRight="16dp"  
12        android:hint="@string/email"  
13        android:inputType="textEmailAddress"  
14        android:textSize="16sp"/>  
15     <EditText  
16         android:id="@+id/etPassword"  
17         android:layout_width="match_parent"  
18         android:layout_height="wrap_content"  
19         android:layout_marginLeft="16dp"  
20         android:layout_marginRight="16dp"  
21         android:hint="@string/password_enter"  
22         android:inputType="textPassword"  
23         android:textSize="16sp"/>  
24     <EditText  
25         android:id="@+id/tvPasswordAgain"  
26         android:layout_width="match_parent"  
27         android:layout_height="wrap_content"  
28         android:layout_marginLeft="16dp"  
29         android:layout_marginRight="16dp"  
30         android:hint="@string/password_enter_again"  
31         android:inputType="textPassword"  
32         android:textSize="16sp"/>  
33  
34     <Button  
35         android:id="@+id/btnRegistration"  
36         android:layout_width="match_parent"  
37         android:layout_height="wrap_content"  
38         android:layout_margin="16dp"  
39         android:text="@string/register"/>  
40 </LinearLayout>
```

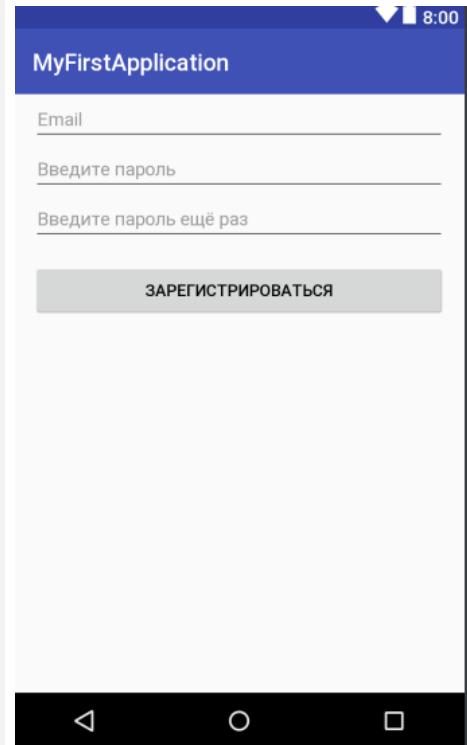


Рис. 3.7

Смотрим RegistrationFragment.java дальше:

```
1 //продолжение RegistrationFragment.java
2     @Nullable
3     @Override//обычная инициализация View
4     public View onCreateView(LayoutInflater inflater,
5                             @Nullable ViewGroup container,
6                             @Nullable Bundle savedInstanceState) {
7         View view = inflater.inflate(
8             R.layout.fr_registration, container, false);
9         mSharedPreferencesHelper = new
10            SharedPreferencesHelper(getActivity());
11        //обычная инициализация View
12        mLogin = view.findViewById(R.id.etLogin);
13        mPassword = view.findViewById(R.id.etPassword);
14        mPasswordAgain = view.findViewById(R.id.tvPasswordAgain);
15        mRegistration = view.findViewById(R.id.btnRegistration);
16        mRegistration.setOnClickListener
17            (mOnRegistrationClickListener);
18        return view;
19    }
```

Далее идёт логика для проверки ввода данных (как в авторизации)

```
1     private boolean isValid(){//проверка мейла и пароля
2         String email = mLogin.getText().toString();
3         if (isValidEmail(email) && isPasswordsValid()) {
4             return true;
5         }
6         return false;
7     }
8     private boolean isValidEmail(String email){//мейл
9         return !TextUtils.isEmpty(email)
10            && Patterns.EMAIL_ADDRESS.matcher(email).matches();
11     }
12     private boolean isPasswordsValid() { //правильность пароля
13         String password = mPassword.getText().toString();
14         String passwordAgain = mPasswordAgain.getText().toString();
15         return password.equals(passwordAgain)
16            && TextUtils.isEmpty(password)
17            && TextUtils.isEmpty(passwordAgain);
18     }
```

Метод showMessage() нужен для показа сообщений; пока что он у нас не используется.

```
1  private void showMessage(@StringRes int string) {
2      //todo показ сообщений
3      Toast.makeText(getActivity(), string, Toast.LENGTH_LONG).show();
4  }
5 } //конец RegistrationFragment.java
```

Вся логика должна происходить при нажатии на кнопку Registration; для этого у нас есть mOnRegistrationClickListener. Всех пользователей, с которыми мы будем работать внутри приложения, мы будем хранить в SharedPreferences. Для этого создадим класс SharedPreferencesHelper.

```
1  // в том же RegistrationFragment.java переписываем OnClickListener
2  private View.OnClickListener mOnRegistrationClickListener
3  = new View.OnClickListener() {
4      @Override
5      public void onClick(View view) {
6          if (isValid()) {
7              boolean isAdded = mSharedPreferencesHelper.
8                  addUser(new User(
9                      mLogin.getText().toString(),
10                     mPassword.getText().toString()
11                 ));
12
13              if (isAdded) {
14                  showMessage(R.string.login_register_success);
15              } else {
16                  showMessage(R.string.login_register_error);
17              }
18          } else {
19              showMessage(R.string.input_error);
20          }
21      }
22  };
```

Перейдём в Package New Java Class SharedPreferencesHelper. OK.

```

1 public class SharedPreferencesHelper {
2     public static final String SHARED_PREF_NAME = "SHARED_PREF_NAME";
3     private SharedPreferences mSharedPreferences; //создали
4     //инициализируем его в конструкторе, передим туда контекст
5     public SharedPreferencesHelper(Context context) {
6         //и получим его из контекста. 1 аргумент - название файла.
7         //2 аргумент - режим работы самого контекста. Делаем приватным
8         mSharedPreferences = context.getSharedPreferences(
9             SHARED_PREF_NAME, Context.MODE_PRIVATE);
10    }

```

Нам нужно описать саму логику сохранения пользователей в SharedPreferences. Для этого нам нужно создать два метода — getUsers и addUser. Для того чтобы хранить нам список пользователей внутри SharedPreferences, необходимо использовать формат данных JSON, потому что сам SharedPreferences не позволяет сохранять ни Serializable, ни Parcelable объекты, но позволяет сохранять строки, поэтому мы наш список пользователей будем форматировать в JSON и сохранять как строку. Далее пишем в SharedPreferencesHelper.java

```

1     public static final String USERS_KEY = "USERS_KEY";
2     //ключ, который будем передавать в mSharedPreferences.getString
3     //далее начинаем вводить и выбираем Type (Reflect).
4     //в TypeToken в качестве генерикового параметра передаём лист юзеров
5     public static final Type USERS_TYPE =
6         new TypeToken<List<User>>() {}.getType();
7     //объявим Gson-овский объект
8     private Gson mGson = new Gson();
9     //чтобы получить список пользователей, из JSON делаем список
10    public List<User> getUsers() {
11        List<User> users = mGson.fromJson(mSharedPreferences
12            .getString(USERS_KEY, ""), USERS_TYPE);
13        //в mSharedPreferences.getString создаём лист юзеров
14        //2ой аргумент (USERS_TYPE) это тип (объявили в TypeToken)
15        return users == null ? new ArrayList<User>() : users;
16        //если пустой - выводим null. Иначе выводим список юзеров
17    }

```

Метод getUsers() готов. Теперь добавим метод addUser(). boolean нам нужен для того, чтобы определить, был добавлен пользователь до этого или нет. Для того чтобы добавить пользователя в SharedPreferences, сначала нужно получить всех пользователей оттуда.

```

1     public boolean addUser(User user) {
2         List<User> users = getUsers();
3         //пробегаемся в цикле и проверяем пользователя, которого мы создаём
4         for (User u : users) {
5             if (u.getLogin().equalsIgnoreCase(user.getLogin())) {
6                 return false; //если существует - просто не добавляем
7             }
8         } //если юзера нет, сохраняем его в mSharedPreferences
9         users.add(user);
10        mSharedPreferences.edit().putString(USER_KEY, mGson.toJson(users,
11            → USERS_TYPE)).apply();
12        //параметром 2 передаём USERS_TYPE, чтобы он перевёл нам правильно
13        return true;
14    }

```

Метод addUser() и основная логика сохранения пользователей в SharedPreferences готовы. Давайте добавим её внутрь RegistrationFragment.

Переходим в RegistrationFragment, в mOnRegistrationClickListener. Допишем:

```

1  public class RegistrationFragment extends Fragment {//...
2      private SharedPreferenceHelper mSharedPreferenceHelper;
3      private View.OnClickListener mOnRegistrationClickListener
4          = new View.OnClickListener() {
5              @Override
6              public void onClick(View view) {
7                  if (isValid()) { //если всё верно добавим юзера
8                      boolean isAdded = mSharedPreferenceHelper.
9                          addUser(new User(
10                              mLogin.getText().toString(),
11                              mPassword.getText().toString()));
12                      if (isAdded) { //если добавился, покажем "Успех"
13                          showMessage(R.string.login_register_success);
14                      } else { //если не добавился, покажем сообщение
15                          "Ошибка"
16                          showMessage(R.string.login_register_error);
17                      }
18                  } else {
19                      showMessage(R.string.input_error);
20                  }
21              }
22          };

```

Не забудем изменить метод onCreateView()

```
1  public View onCreateView(LayoutInflater inflater,
2   @Nullable ViewGroup container, @Nullable Bundle
3   savedInstanceState) {
4   View view = inflater.inflate(R.layout.fr_registration,
5   container, false);
6   //проинициализировали и передаём контекст
7   mSharedPreferencesHelper = new
8   SharedPreferencesHelper(getActivity());
9   //...
10 }
```

Вроде готово. Давайте посмотрим: всё запускается, нажимаем на кнопку регистрации, вводим все данные, но ничего не происходит. Давайте дебажить (кнопка Attach debugger to Android во время работы нашего приложения на эмуляторе):

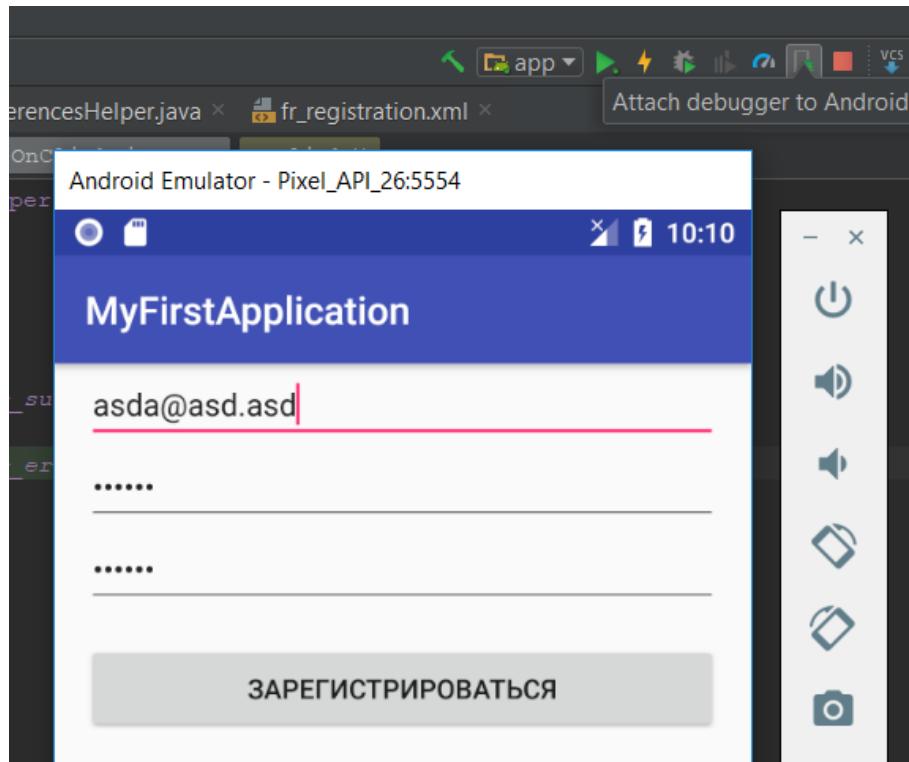


Рис. 3.8

Посмотрим на поведение метода IsInputValid().

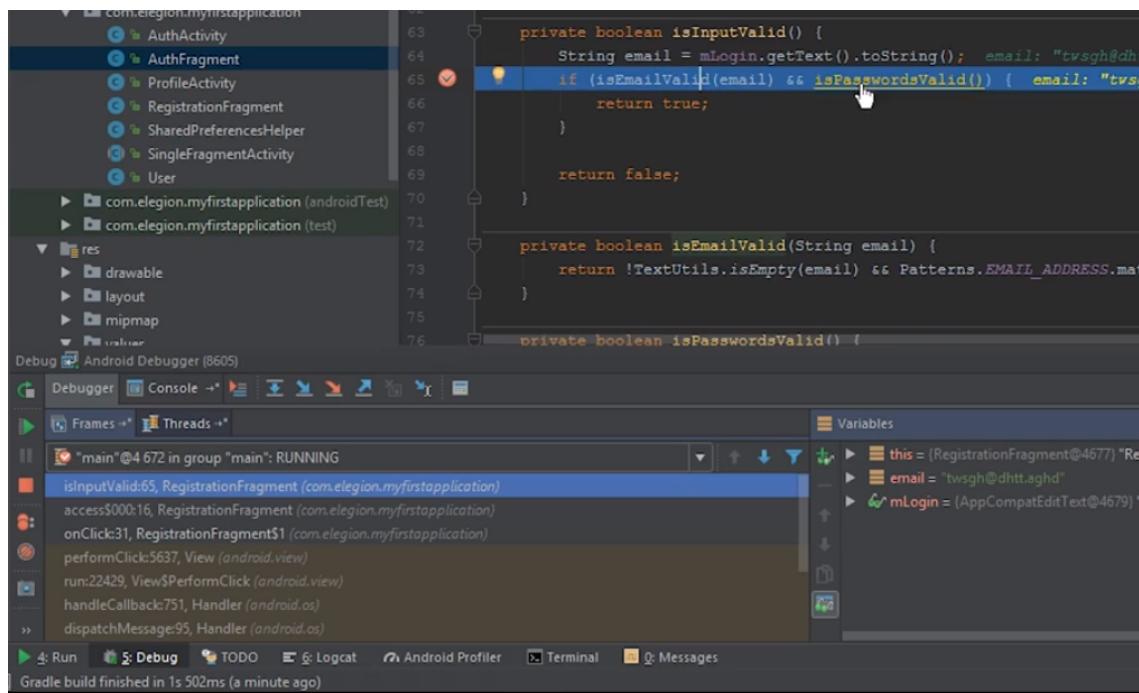


Рис. 3.9

`isEmailValid()` — true, `isPasswordValid()` — false. Переайдём в метод `isPasswordValid()`; `password.equals()` — true, `isEmpty()` — false, `isEmpty()` — false. Всё верно, мы допустили ошибку в логике сравнения паролей. Добавим «не» перед ними; то есть если они не пустые, то они нам подходят

```

1 // RegistrationFragment.java :
2     private boolean isPasswordsValid() { //правильность пароля
3         String password = mPassword.getText().toString();
4         String passwordAgain = mPasswordAgain.getText().toString();
5         return password.equals(passwordAgain)
6             && TextUtils.isEmpty(password)
7             && TextUtils.isEmpty(passwordAgain);
8     }

```

Запустим ещё раз и видим, что всё хорошо работает - при правильном вводе появляется тост с надписью "Успешно". Нажмём ещё раз - "Уже занято".

Мы научились использовать `SharedPreferences` и научились писать какую-то бизнес-логику для приложения; эта бизнес-логика может подойти, когда вы не используете сервер, а пишете просто тестовые приложения.

## О проекте

Академия e-Legion – это образовательная платформа для повышения квалификации в мобильной разработке. Слушайте лекции топовых разработчиков, выполняйте практические задания и прокачивайте свои скиллы. Получите высокооплачиваемую профессию – разработчик мобильных приложений.

## Программа “Android-разработчик”

### Блок 1. Быстрый старт в Android-разработку

- Описание платформы Android
- Знакомство с IDE – Android Studio и системой сборки – Gradle
- Дебаг и логгирование
- Знакомство с основными сущностями Android-приложения
- Работа с Activity и Fragment
- Знакомство с элементами интерфейса – View, ViewGroup

### Блок 2. Многопоточность и сетевое взаимодействие

- Работа со списками: RecyclerView
- Средства для обеспечения многопоточности в Android
- Работа с сетью с помощью Retrofit2/Okhttp3
- Базовое знакомство с реактивным программированием: RxJava2
- Работа с уведомлениями
- Работа с базами данных через Room

### Блок 3. Архитектура Android-приложений

- MVP- и MVVM-паттерны
- Android Architecture Components
- Dependency Injection через Dagger2
- Clean Architecture

### Блок 4. Тестирование и работа с картами

- Google Maps

- Оптимизация фоновых работ
- БД Realm
- WebView, ChromeCustomTabs
- Настройки приложений
- Picasso и Glide
- Unit- и UI-тестирование: Mockito, PowerMock, Espresso, Robolectric

### **Блок 5. Дизайн и анимации**

- Стили и Темы
- Material Design Components
- Анимации
- Кастомные элементы интерфейса: Custom View

### **Блок 6. Облачные сервисы и периферия**

- Google Firebase
- Google Analytics
- Push-уведомления
- Работа с сенсорами и камерой