Лабораторная работа 8. Разработка клиента под Android

Разработка клиентского приложения на Java Swing и Android имеет несколько ключевых различий, поскольку каждая платформа использует различные подходы и архитектуры.

1. Платформа и экосистема

Java Swing: Swing — это библиотека для создания графических интерфейсов на языке Java, используемая преимущественно для создания десктопных приложений. Swing предоставляет разработчикам набор компонентов (кнопки, поля ввода, меню и т.д.), которые работают на всех платформах, поддерживающих Java. Swing приложения запускаются на JVM и обычно являются кроссплатформенными.

Android: Разработка под Android использует платформу Android SDK, которая включает в себя фреймворки и инструменты для создания приложений, предназначенных для мобильных устройств. Основной язык разработки — Java или Kotlin, и приложения работают на Android Runtime (ART) на мобильных устройствах.

2. Компоненты пользовательского интерфейса (UI)

Java Swing: Swing использует собственные компоненты (например, `JButton`, `JTextField`, `JFrame`). Эти компоненты рендерятся одинаково на всех платформах.

Android: В Android UI строится с использованием XML-файлов для описания интерфейса и View-компонентов (например, `Button`, `TextView`, `Activity`). Приложения для Android ориентированы на работу с сенсорным экраном и динамически подстраиваются под разные размеры экранов и ориентации устройств.

3. Управление жизненным циклом

Java Swing: Основной поток приложения контролируется самим разработчиком. Запуск окон и взаимодействие с ними выполняется в основном цикле событий, который программист может контролировать через вызовы в потоке Event Dispatch Thread (EDT).

Android: Жизненный цикл управляется системой Android. Существуют методы `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, и `onDestroy()` в классе `Activity`, которые используются для управления состоянием приложения. Система сама контролирует управление памятью и временем выполнения.

4. Адаптация под разные устройства

Java Swing: Обычно приложения разрабатываются для стандартных экранов настольных ПК или ноутбуков. Адаптация под разные разрешения и DPI, как правило, не является большой проблемой.

Android: Приложения должны адаптироваться под огромное количество устройств с разными разрешениями экранов, соотношениями сторон, плотностью пикселей и версией операционной системы. Это требует использования адаптивных интерфейсов, фрагментов и работы с ресурсами для разных экранов.

5. Работа с ресурсами

Java Swing: Ресурсы (например, изображения, файлы) обычно загружаются напрямую из файловой системы или через классы-обработчики ресурсов Java. Конфигурация может быть реализована с использованием `.properties` файлов или аналогов.

Android: Ресурсы управляются через структуру проекта Android. Изображения, стили, строки и другие ресурсы хранятся в специальных папках (`res/drawable`, `res/layout`, `res/values`) и могут быть динамически подгружены через API Android. XML используется для описания интерфейсов и стилей.

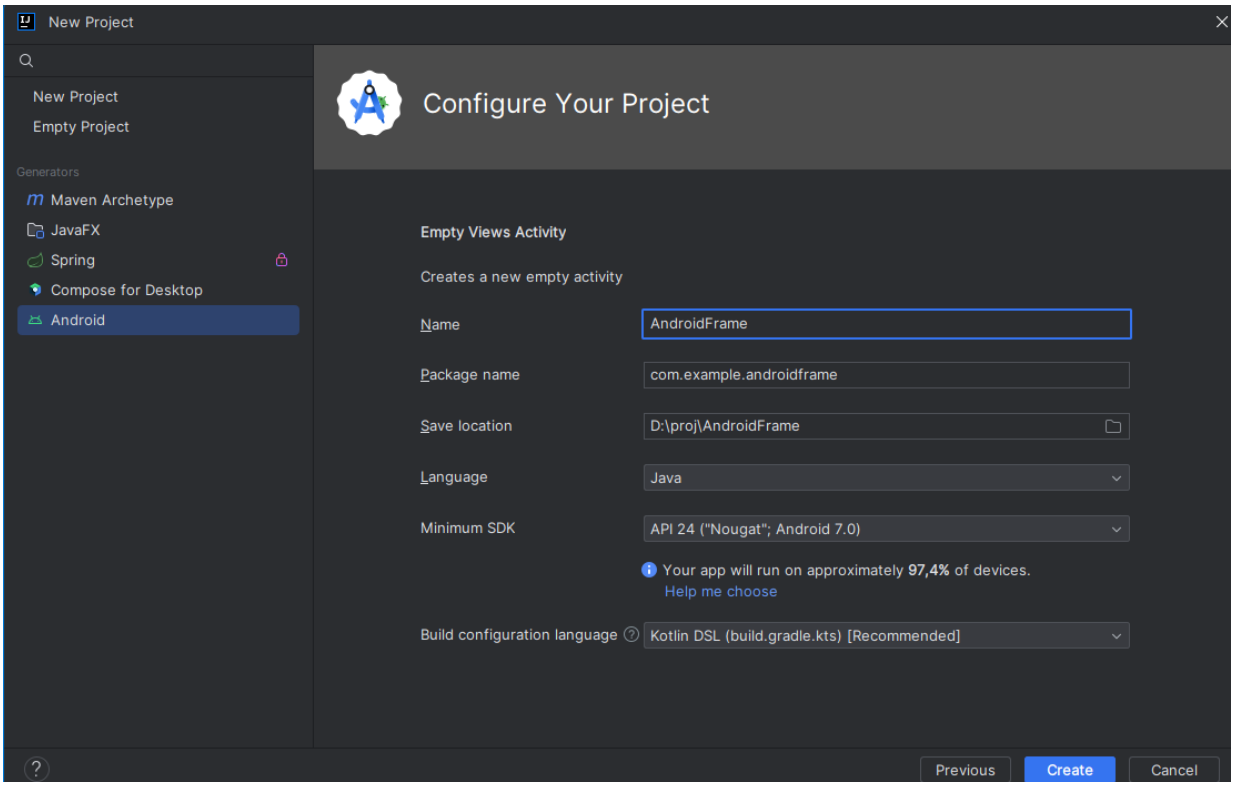
6. Взаимодействие с системой

Java Swing: Десктопные приложения могут взаимодействовать с системой напрямую (например, с файловой системой, сетевыми ресурсами) и имеют доступ к широкому набору API, предоставляемых Java SE.

Android: Мобильные приложения взаимодействуют с операционной системой через предоставленные API Android. Приложение должно запрашивать разрешения на доступ к камере, файлам, интернету и другим системным ресурсам, и управляться в рамках ограничений безопасности мобильной платформы.

Цель работы: получить практические навыки работы с Spring Framework. Разработать клиент под Android

Создадим новое приложение под Android.



Добавим в проект необходимые зависимости. Для этого необходимо модифицировать файл build.gradle.kts дополнив его раздел *dependencies*

```
dependencies {

    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation ("com.squareup.retrofit2:retrofit:2.9.0")
    implementation ("com.squareup.retrofit2:converter-gson:2.9.0")
    implementation ("androidx.recyclerview:recyclerview:1.2.1")

    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.2.1")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.6.1")
}
```

Описание зависимостей:

1. **`androidx.appcompat:appcompat:1.7.0`**
   Это библиотека поддержки, которая обеспечивает совместимость с более старыми версиями Android (предоставляет компоненты для

работы с ActionBar, поддержка фрагментов и т.д.). Она позволяет использовать новейшие функции на старых версиях Android.

2. **`com.google.android.material:material:1.12.0`**
   Эта библиотека содержит компоненты Material Design от Google. Она включает различные UI-элементы (кнопки, карточки, поля ввода и т.д.), которые следуют концепции дизайна Material Design.
3. **`androidx.constraintlayout:constraintlayout:2.1.4`**
   ConstraintLayout — это мощный виджет для построения гибких и сложных макетов интерфейса на Android. Он позволяет легко создавать сложные макеты с минимальным вложением элементов и улучшенной производительностью по сравнению с другими макетами.
4. **`com.squareup.retrofit2:retrofit:2.9.0`**
   Retrofit — это библиотека для упрощения работы с сетевыми запросами. Она помогает разработчику отправлять HTTP-запросы к API и обрабатывать ответы. Retrofit поддерживает асинхронные запросы, обработку ошибок и гибкую настройку.
5. **`com.squareup.retrofit2:converter-gson:2.9.0`**
   Это расширение для Retrofit, которое позволяет автоматически преобразовывать JSON-ответы из API в объекты Java, используя библиотеку Gson. Это упрощает работу с данными, полученными из API.
6. **`androidx.recyclerview:recyclerview:1.2.1`**
   RecyclerView — это более гибкий и эффективный заменитель ListView для отображения списков и сеток данных. Он поддерживает кэширование элементов.

В разделе res/xml необходимо теперь добавить файл конфигурации сетевого подключения, назовем его *network_security_config.xml*. Его содержимое представлено ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">10.0.2.2</domain>
    </domain-config>
</network-security-config>
```

Данная конфигурация используется для настройки сетевых параметров, таких как использование небезопасного трафика (HTTP) или разрешение трафика только по HTTPS.

Разбор содержимого:

1. **`<network-security-config>`**:
   Корневой элемент конфигурации сетевой безопасности. Он указывает, что для приложения будут настроены правила по обработке сетевых соединений.

2. **<domain-config>**:
   Этот элемент определяет конфигурацию для конкретного домена (в данном случае — локального хоста), с которым приложение может взаимодействовать. Внутри него задаются параметры безопасности для соединений с данным доменом.
3. **cleartextTrafficPermitted="true"**:
   Этот атрибут разрешает использование незашифрованного трафика (HTTP) для указанного домена. Обычно HTTP считается небезопасным, так как данные передаются без шифрования, но в этом случае приложение явно разрешает использование незашифрованных соединений с данным доменом. Это может быть полезно, например, в целях тестирования.
4. **<domain includeSubdomains="true">10.0.2.2</domain>**:
   Здесь указан домен (в данном случае это IP-адрес `10.0.2.2`), для которого применяется эта конфигурация.
   - **10.0.2.2** — это специальный IP-адрес, используемый в Android-эмуляторе для обращения к хост-машине. Эмулятор Android не может напрямую обращаться к `localhost` на хосте, поэтому для этого используется IP `10.0.2.2`.
   - **includeSubdomains="true"** означает, что данная конфигурация применяется ко всем поддоменам этого IP-адреса.

Немного дополним файл AndroidManifest.xml. Его содержимое:

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

    <application
            android:networkSecurityConfig="@xml/network_security_config"
            android:allowBackup="true"
            android:dataExtractionRules="@xml/data_extraction_rules"
            android:fullBackupContent="@xml/backup_rules"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/Theme.MyApplication"
            tools:targetApi="31">

        <activity
                android:name=".LoginActivity"
                android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>

        </activity>
        <activity android:name=".TaskManagerActivity" />
    </application>
    <uses-permission android:name="android.permission.INTERNET" /> <!--
Запрос разрешения на доступ в интернет -->
</manifest>
```

Разбор элементов:

1. **`<manifest>`**: Корневой элемент файла манифеста. В нем указываются пространства имен и используется для объявления всех компонентов приложения.
2. **`<application>`**: Описывает само приложение и его конфигурации, включая иконки, темы, настройки безопасности и бэкапа. В данном блоке также описываются активности (экранные компоненты) приложения.
    - **`android:networkSecurityConfig="@xml/network_security_config"`**:
      Указывает на файл конфигурации сетевой безопасности (`network_security_config.xml`), который задает правила для сетевых подключений (этот файл вы показали ранее). В данном случае приложение использует файл для разрешения незашифрованного HTTP-трафика к локальному хосту (`10.0.2.2`).
    - **`android:allowBackup="true"`**:
      Разрешает создание бэкапов данных приложения в случае необходимости.
    - **`android:dataExtractionRules="@xml/data_extraction_rules"`**:
      Ссылка на правила извлечения данных, которые управляют, какие данные приложения могут быть сохранены и восстановлены. Это полезно при переносе данных между устройствами или создании резервных копий.
    - **`android:fullBackupContent="@xml/backup_rules"`**:
      Указывает правила для полного бэкапа, которые настраиваются через отдельный XML-файл (`backup_rules.xml`). Этот файл определяет, какие данные нужно включить в резервную копию, а какие исключить.
    - **`android:icon="@mipmap/ic_launcher"`**:
      Указывает на основную иконку приложения, которую система будет использовать для отображения в меню приложений.
    - **`android:label="@string/app_name"`**:
      Указывает имя приложения, которое будет отображаться на устройстве. Значение берется из ресурсов строкового файла (`@string/app_name`).
    - **`android:roundIcon="@mipmap/ic_launcher_round"`**:
      Указывает на круглую иконку приложения, используемую на некоторых устройствах, поддерживающих круглые иконки.
    - **`android:supportsRtl="true"`**:
      Поддержка RTL (Right-to-Left) ориентации для языков, которые читаются справа налево, таких как арабский или иврит.
    - **`android:theme="@style/Theme.MyApplication"`**:
      Указывает тему (стиль) для приложения. В данном случае используется тема, определенная в файле стилей как `Theme.MyApplication`.
    - **`tools:targetApi="31"`**:
      Это указывает на целевую версию API (Android 12), с которой разрабатывается приложение. `tools:targetApi` — это атрибут, используемый для инструментов сборки и анализа, чтобы указать, на какую версию API ориентироваться при разработке.
3. **`<activity>`**: Описывает активити — компоненты интерфейса приложения. Здесь заданы две активности:
    - **`LoginActivity`**:
      Это активность для экрана входа в приложение.
        - **`android:name=".LoginActivity"`**: Имя класса активности.
        - **`android:exported="true"`**: Указывает, что данная активность может быть запущена извне (например, через другой компонент системы или другое приложение).

        - **`<intent-filter>`**: Описывает, как эта активность может быть запущена. В данном случае:
            - **`<action android:name="android.intent.action.MAIN"/>`**: Эта активность является точкой входа в приложение (главная активность).
            - **`<category android:name="android.intent.category.LAUNCHER"/>`**: Делает активность доступной в лаунчере (на экране приложений).
    - **`TaskManagerActivity`**:
      Еще одна активность, но без дополнительной конфигурации (возможно, она будет запускаться из кода).
4. **`<uses-permission>`**:
    - **`android.permission.INTERNET`**:
      Приложение запрашивает разрешение на доступ в интернет. Это нужно для того, чтобы приложение могло выполнять сетевые запросы, такие как взаимодействие с API через HTTP или HTTPS.

Можно приступать к разработке самого приложения. Для начала в папке Res/layout создадим несколько xml файлов для интерфейса приложения.

Первым делом создадим форму логина activity_login.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:padding="16dp">

    <EditText
            android:id="@+id/username"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Username" />

    <EditText
            android:id="@+id/password"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Password"
            android:inputType="textPassword" />

    <Button
            android:id="@+id/loginButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Login" />
</LinearLayout>
```

Разбор элементов:

1. **`<LinearLayout>`**:
    - **`android:layout_width="match_parent"`**: Ширина контейнера `LinearLayout` занимает всю ширину экрана.
    - **`android:layout_height="match_parent"`**: Высота контейнера также занимает всю доступную высоту экрана.
    - **`android:orientation="vertical"`**: Элементы внутри контейнера будут расположены вертикально (один под другим).
    - **`android:padding="16dp"`**: Весь макет будет иметь внутренний отступ (паддинг) в 16dp от границ экрана.
2. **`<EditText android:id="@+id/username">`**:
    - **`android:id="@+id/username"`**: Уникальный идентификатор для этого элемента, который можно использовать для доступа к нему в коде.
    - **`android:layout_width="match_parent"`**: Поле ввода будет занимать всю ширину контейнера.
    - **`android:layout_height="wrap_content"`**: Высота будет подстраиваться под содержимое (текст).
    - **`android:hint="Username"`**: Подсказка внутри поля ввода (текст, который исчезает при вводе) с указанием, что это поле для ввода имени пользователя.
3. **`<EditText android:id="@+id/password">`**:
    - **`android:id="@+id/password"`**: Идентификатор для доступа к элементу в коде.
    - **`android:layout_width="match_parent"`**: Поле ввода пароля будет занимать всю ширину контейнера.
    - **`android:layout_height="wrap_content"`**: Высота автоматически подстраивается под содержимое.
    - **`android:hint="Password"`**: Подсказка с текстом "Password" для поля ввода пароля.
    - **`android:inputType="textPassword"`**: Указывает, что это поле ввода пароля, где вводимые символы будут скрываться (заменяться символами, например, точками).
4. **`<Button android:id="@+id/loginButton">`**:
    - **`android:id="@+id/loginButton"`**: Уникальный идентификатор для кнопки.
    - **`android:layout_width="match_parent"`**: Кнопка будет занимать всю ширину контейнера.
    - **`android:layout_height="wrap_content"`**: Высота кнопки будет подстраиваться под содержимое (текст).
    - **`android:text="Login"`**: Текст, который будет отображаться на кнопке — "Login"

Далее создадим форму, которая будет выводить список задач.

activity_task_manager.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">

    <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"/>
</LinearLayout>
```

Разбор элементов:

1. **`<LinearLayout>`**:
    - **`android:layout_width="match_parent"`**: Ширина контейнера `LinearLayout` будет занимать всю ширину экрана.
    - **`android:layout_height="match_parent"`**: Высота контейнера будет занимать всю доступную высоту экрана.
    - **`android:orientation="vertical"`**: Элементы внутри контейнера будут располагаться вертикально, один под другим (в данном случае у нас только один элемент — `RecyclerView`).
2. **`<androidx.recyclerview.widget.RecyclerView>`**:
    - **`android:id="@+id/recyclerView"`**: Уникальный идентификатор для компонента `RecyclerView`, через который к нему можно будет обращаться в коде (например, в активности или фрагменте).
    - **`android:layout_width="match_parent"`**: Ширина `RecyclerView` будет равна ширине родительского контейнера (то есть экрана).
    - **`android:layout_height="match_parent"`**: Высота `RecyclerView` также занимает всю доступную высоту экрана, позволяя ему отобразить список с возможностью вертикальной прокрутки.

Этот макет задает структуру экрана с одним компонентом — `RecyclerView`, который обычно используется для отображения списков данных. `RecyclerView` является гибким и мощным элементом для отображения больших наборов данных, поддерживающий эффективную прокрутку, повторное использование элементов, добавление анимаций и обработку кликов.

Далее создадим XML-файл, который будет описывать макет для одного элемента списка, который можно использовать в `RecyclerView`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:orientation="vertical"
              android:padding="16dp">

    <TextView
            android:id="@+id/descriptionTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Task Description"
            android:textSize="16sp" />
</LinearLayout>
```

Разбор элементов:

1. **`<LinearLayout>`**:
   - **`xmlns:android="http://schemas.android.com/apk/res/android"`**: Указывает пространство имен для атрибутов Android.
   - **`android:layout_width="match_parent"`**: Ширина контейнера `LinearLayout` будет равна ширине родительского контейнера, что позволит ему занимать всю доступную ширину.
   - **`android:layout_height="wrap_content"`**: Высота контейнера будет автоматически подстраиваться под содержимое (в данном случае — текст внутри `TextView`).
   - **`android:orientation="vertical"`**: Элементы внутри контейнера будут расположены вертикально (но здесь у нас только один элемент).
   - **`android:padding="16dp"`**: Внутренний отступ в 16dp со всех сторон для создания пространства вокруг текста.
2. **`<TextView>`**:
   - **`android:id="@+id/descriptionTextView"`**: Уникальный идентификатор для текстового поля, который позволит обращаться к нему в коде.
   - **`android:layout_width="wrap_content"`**: Ширина `TextView` будет подстраиваться под текст.
   - **`android:layout_height="wrap_content"`**: Высота `TextView` также будет автоматически подстраиваться под текст.
   - **`android:text="Task Description"`**: Текст, который будет отображаться по умолчанию в этом элементе.
   - **`android:textSize="16sp"`**: Размер текста установлен в 16sp (использование sp рекомендуется для текстов, чтобы обеспечить масштабируемость на разных устройствах).

Можем приступать к разработке логики приложения.

Для начала создадим класс для авторизации `LoginActivity`.java.

```
import android.content.Intent;
import android.os.Bundle;
import android.util.Base64;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class LoginActivity extends AppCompatActivity {

    private EditText usernameField;
    private EditText passwordField;
    private Button loginButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login); // Подключаем XML макет

        usernameField = findViewById(R.id.username);
        passwordField = findViewById(R.id.password);
        loginButton = findViewById(R.id.loginButton);

        loginButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = usernameField.getText().toString();
                String password = passwordField.getText().toString();
                String encodedAuth = Base64.encodeToString((username + ":" +
password).getBytes(), Base64.NO_WRAP);

                // Переход на TaskManagerActivity
                Intent intent = new Intent(LoginActivity.this,
TaskManagerActivity.class);
                intent.putExtra("authHeader", encodedAuth); // Передаем токен
в Intent
                startActivity(intent);
            }
        });
    }
}
```

Разбор кода:

1. **Импорт необходимых классов**:

   ```
   import android.content.Intent;
   import android.os.Bundle;
   import android.util.Base64;
   import android.view.View;
   import android.widget.Button;
   import android.widget.EditText;
   import android.widget.Toast;
   import androidx.appcompat.app.AppCompatActivity;
   ```

- Импортируются классы, необходимые для работы с элементами интерфейса, интентами, кодированием и другими функциями.

2. **Определение класса `LoginActivity`**:

```
public class LoginActivity extends AppCompatActivity {
```

   - Класс `LoginActivity` наследуется от `AppCompatActivity`, что позволяет использовать функции совместимости и темы Material Design.

3. **Объявление переменных**:

```
private EditText usernameField;
private EditText passwordField;
private Button loginButton;
```

   - Переменные для полей ввода имени пользователя, пароля и кнопки входа.

4. **Метод `onCreate`**:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login); // Подключаем XML макет
```

   - Этот метод вызывается при создании активности. Здесь происходит установка макета, определенного в XML-файле (`activity_login.xml`).

5. **Инициализация компонентов интерфейса**:

```
usernameField = findViewById(R.id.username);
passwordField = findViewById(R.id.password);
loginButton = findViewById(R.id.loginButton);
```

   - Находим элементы интерфейса по их идентификаторам и присваиваем их соответствующим переменным.

6. **Обработка нажатия кнопки входа**:

```
loginButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String username = usernameField.getText().toString();
        String password = passwordField.getText().toString();
        String encodedAuth = Base64.encodeToString((username + ":" +
password).getBytes(), Base64.NO_WRAP);
```

   - Устанавливаем обработчик нажатия для кнопки. Когда кнопка нажата, получаем текст из полей ввода имени пользователя и пароля.
   - Создаем строку `encodedAuth`, содержащую закодированную в Base64 комбинацию имени пользователя и пароля в формате `username:password`. Это может быть полезно для отправки заголовков авторизации на сервер.

7. **Переход на `TaskManagerActivity`**:

```
        // Переход на TaskManagerActivity
        Intent intent = new Intent(LoginActivity.this,
TaskManagerActivity.class);
        intent.putExtra("authHeader", encodedAuth); // Передаем токен в
Intent
        startActivity(intent);
    }
```

```
});
```

- Создаем новый `Intent`, чтобы перейти на `TaskManagerActivity`.
- Передаем закодированный заголовок авторизации (`encodedAuth`) через `putExtra()`.
- Запускаем `TaskManagerActivity` с помощью `startActivity(intent)`.

Далее создадим класс для управления активностями TaskManagerActivity.java

```
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class TaskManagerActivity extends AppCompatActivity {

    private RecyclerView recyclerView;
    private TaskAdapter taskAdapter;
    private String authHeader;

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

        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        // Получаем токен авторизации из LoginActivity
        authHeader = getIntent().getStringExtra("authHeader");

        loadTasks();
    }

    private void loadTasks() {
        // Используем Retrofit для обращения к API
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://10.0.2.2:8080/") // localhost для эмулятора
                .addConverterFactory(GsonConverterFactory.create())
                .build();

        TaskApi taskApi = retrofit.create(TaskApi.class);
        Call<List<Task>> call = taskApi.getTasks("Basic " + authHeader);

        call.enqueue(new Callback<List<Task>>() {
            @Override
            public void onResponse(Call<List<Task>> call,
Response<List<Task>> response) {
                if (response.isSuccessful() && response.body() != null) {
```

```
                    // Передаем данные в адаптер для отображения
                    List<Task> tasks = response.body();
                    taskAdapter = new TaskAdapter(tasks);
                    recyclerView.setAdapter(taskAdapter);
                } else {
                    Toast.makeText(TaskManagerActivity.this, "Ошибка загрузки
задач", Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onFailure(Call<List<Task>> call, Throwable t) {
                Log.e("TaskManagerActivity", "Ошибка загрузки", t);
            }
        });
    }
}
```

Разбор кода:

1. **Импорт необходимых классов**:

```
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
```

   - Импортируются необходимые классы для работы с интерфейсом, логированием и сетевыми запросами через Retrofit.

2. **Определение класса `TaskManagerActivity`**:

```
public class TaskManagerActivity extends AppCompatActivity {
```

3. **Объявление переменных**:

```
private RecyclerView recyclerView;
private TaskAdapter taskAdapter;
private String authHeader;
```

   - `RecyclerView` для отображения списка задач.
   - `TaskAdapter`, который будет связывать данные с `RecyclerView`.
   - `authHeader` для хранения заголовка авторизации.

4. **Метод `onCreate`**:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_task_manager);
```

- Этот метод вызывается при создании активности. Здесь происходит установка макета, определенного в XML-файле (`activity_task_manager.xml`).

5. **Инициализация `RecyclerView` и загрузка задач**:

```
recyclerView = findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(this));

// Получаем токен авторизации из LoginActivity
authHeader = getIntent().getStringExtra("authHeader");

loadTasks();
```

- Находим `RecyclerView` по идентификатору и устанавливаем `LinearLayoutManager` для управления расположением элементов.
- Получаем токен авторизации, переданный из `LoginActivity`, с помощью `getIntent().getStringExtra("authHeader")`.
- Вызываем метод `loadTasks()` для загрузки задач с сервера.

6. **Метод `loadTasks`**:

```
private void loadTasks() {
    // Используем Retrofit для обращения к API
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://10.0.2.2:8080/") // localhost для
эмулятора
            .addConverterFactory(GsonConverterFactory.create())
            .build();
```

- Создаем экземпляр `Retrofit`, указывая базовый URL для API (используем `10.0.2.2` для обращения к localhost из эмулятора) и добавляя конвертер Gson для преобразования JSON в объекты Java.

7. **Создание API-интерфейса и отправка запроса**:

```
java

TaskApi taskApi = retrofit.create(TaskApi.class);
Call<List<Task>> call = taskApi.getTasks("Basic " + authHeader);
```

- Создаем экземпляр интерфейса `TaskApi`, который определяет методы для обращения к API.
- Создаем вызов для получения списка задач, добавляя заголовок авторизации.

8. **Обработка ответа от API**:

```
call.enqueue(new Callback<List<Task>>() {
    @Override
    public void onResponse(Call<List<Task>> call, Response<List<Task>>
response) {
        if (response.isSuccessful() && response.body() != null) {
            // Передаем данные в адаптер для отображения
            List<Task> tasks = response.body();
            taskAdapter = new TaskAdapter(tasks);
            recyclerView.setAdapter(taskAdapter);
        } else {
```

```
            Toast.makeText(TaskManagerActivity.this, "Ошибка загрузки
задач", Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onFailure(Call<List<Task>> call, Throwable t) {
        Log.e("TaskManagerActivity", "Ошибка загрузки", t);
    }
});
```

- Вызываем `enqueue()` для асинхронного выполнения запроса.
- В методе `onResponse()` проверяем успешность ответа и наличие данных. Если данные получены успешно, создаем экземпляр адаптера `TaskAdapter` и устанавливаем его для `RecyclerView`. Если возникла ошибка, выводим сообщение с помощью `Toast`.
- В методе `onFailure()` логируем ошибку, если запрос не удался.

Далее создадим класс TaskAdapter.java. Этот класс представляет собой адаптер для RecyclerView, который используется для отображения списка задач в Android-приложении. Адаптер связывает данные (в данном случае, список задач) с пользовательским интерфейсом. Разберем код по частям.

Разбор кода:

1. **Импорт необходимых классов**:

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
```

   - Импортируются необходимые классы для работы с элементами пользовательского интерфейса и адаптером `RecyclerView`.

2. **Определение класса `TaskAdapter`**:

```
public class TaskAdapter extends
RecyclerView.Adapter<TaskAdapter.TaskViewHolder> {
```

   - Класс `TaskAdapter` наследуется от `RecyclerView.Adapter` и использует внутренний класс `TaskViewHolder`.

3. **Объявление переменной `taskList`**:

```
private List<Task> taskList;

public TaskAdapter(List<Task> taskList) {
    this.taskList = taskList;
}
```

   - Список задач (`taskList`) передается через конструктора адаптера.

4. **Метод `onCreateViewHolder`**:

```
@NonNull
@Override
public TaskViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
    View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_task,
parent, false);
    return new TaskViewHolder(view);
}
```

   - Этот метод вызывается, когда требуется новый `ViewHolder`. Он создает новый элемент пользовательского интерфейса для задачи, используя `LayoutInflater` для преобразования XML-макета (`item_task.xml`) в объект `View`.
   - Возвращает новый экземпляр `TaskViewHolder`, который будет управлять элементом списка.

5. **Метод `onBindViewHolder`**:

```
@Override
public void onBindViewHolder(@NonNull TaskViewHolder holder, int
position) {
    Task task = taskList.get(position);
    holder.descriptionTextView.setText(task.getDescription());
}
```

   - Этот метод связывает данные с `ViewHolder`. Он вызывается для каждого элемента в списке.
   - Получает задачу по позиции и устанавливает описание задачи в `TextView` внутри `TaskViewHolder`.

6. **Метод `getItemCount`**:

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

   - Возвращает общее количество задач в списке. Этот метод нужен, чтобы `RecyclerView` знал, сколько элементов отображать.

7. **Внутренний класс `TaskViewHolder`**:

```
public static class TaskViewHolder extends RecyclerView.ViewHolder {
    TextView descriptionTextView;

    public TaskViewHolder(@NonNull View itemView) {
        super(itemView);
        descriptionTextView =
itemView.findViewById(R.id.descriptionTextView);
    }
}
```

   - Этот класс представляет отдельный элемент списка и содержит ссылку на `TextView`, в котором будет отображаться описание задачи.
   - В конструкторе `TaskViewHolder` происходит инициализация `TextView` с помощью `findViewById`.

Далее создадим классы Task.java и TaskApi.java

```
public class Task {
    private String description;

    public String getDescription() {
        return description;
    }
}
```

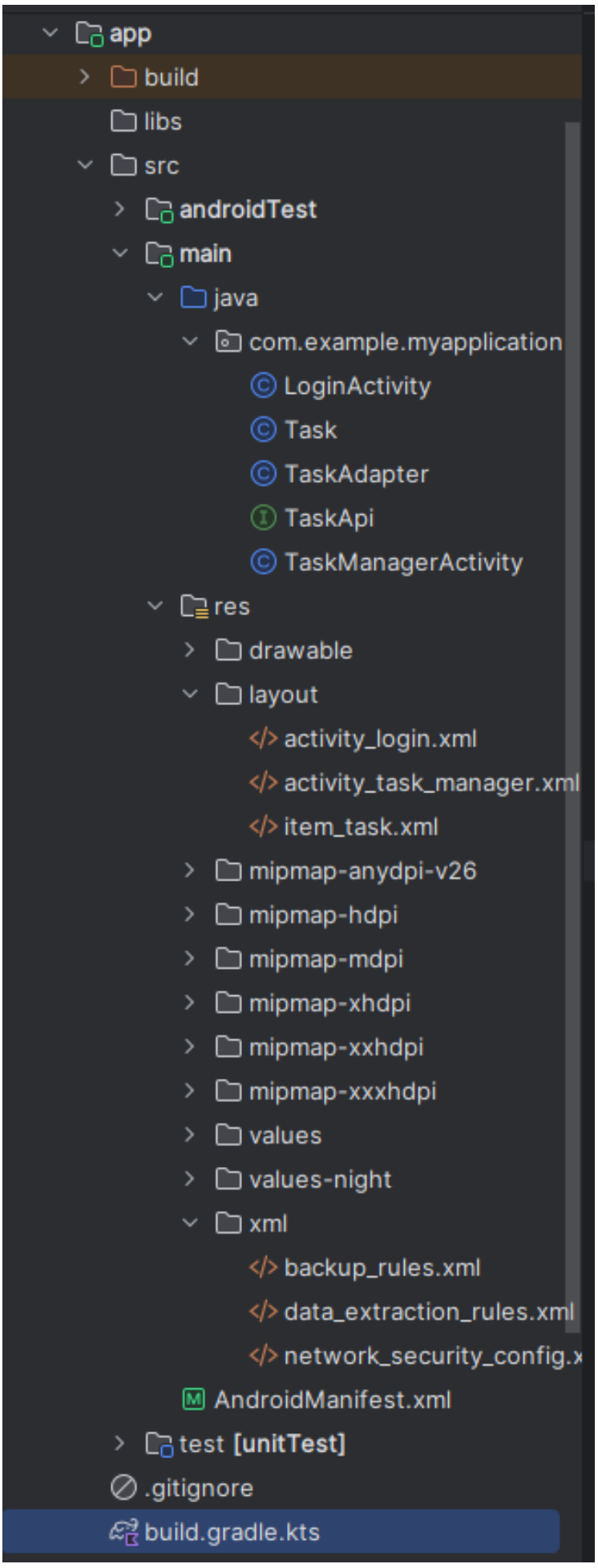
```
import java.util.List;
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Header;

public interface TaskApi {
    @GET("/api")
    Call<List<Task>> getTasks(@Header("Authorization") String authHeader);
}
```

Task, который используется для описания сущности «задача» в приложении.

TaskApi, используется для описания методов взаимодействия с RESTful API с помощью библиотеки Retrofit. Интерфейс определяет, как приложение будет обращаться к серверу для получения данных о задачах.

Структура всего проекта представлена на следующей странице.


app
build
libs
src
androidTest
main
java
com.example.myapplication
LoginActivity
Task
TaskAdapter
TaskApi
TaskManagerActivity
res
drawable
layout
activity_login.xml
activity_task_manager.xml
item_task.xml
mipmap-anydpi-v26
mipmap-hdpi
mipmap-mdpi
mipmap-xhdpi
mipmap-xxhdpi
mipmap-xxxhdpi
values
values-night
xml
backup_rules.xml
data_extraction_rules.xml
network_security_config.x
AndroidManifest.xml
test [unitTest]
.gitignore
build.gradle.kts

Проверим как работает приложение. Запустим Серверное приложение на Spring:

Запустим эмулятор в среде разработки:

Введем логин и пароль и авторизуемся. Приложение выводит нам список дел:

Задание:

1. Изучить теоретический материал;

2. Написать приложение следуя примеру;

3. Сделать отчет