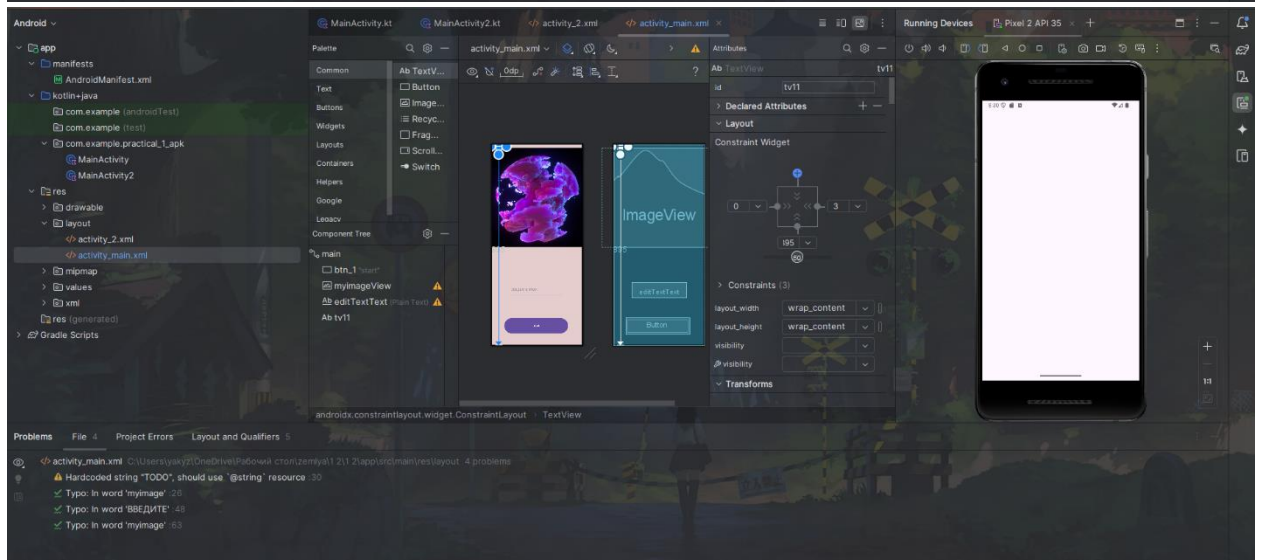
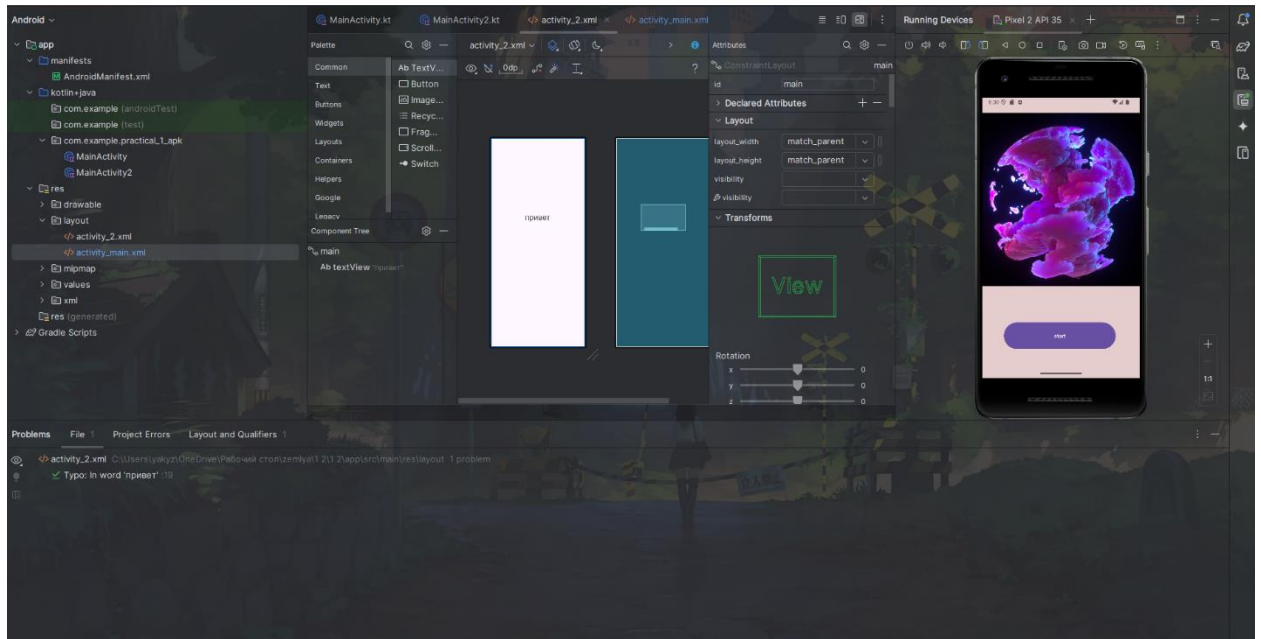
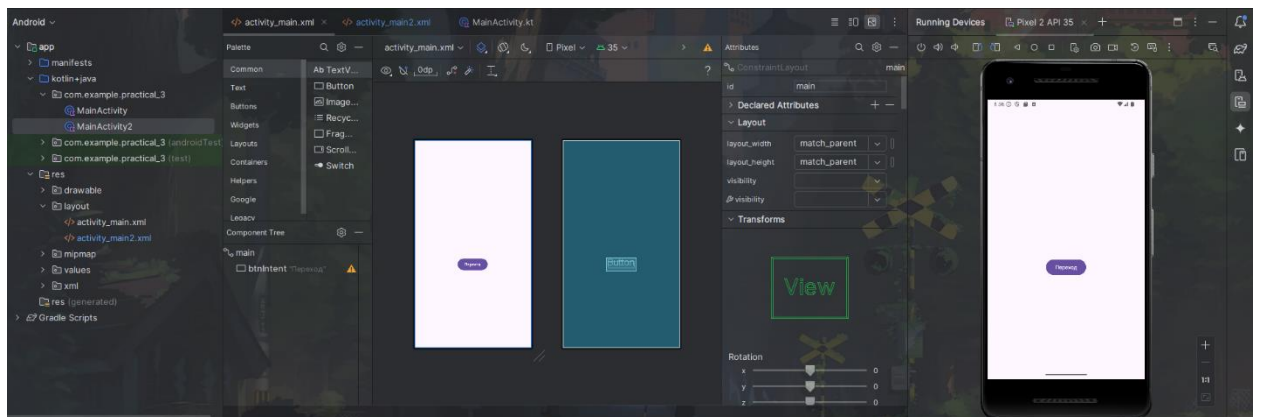


## Практика 1,2



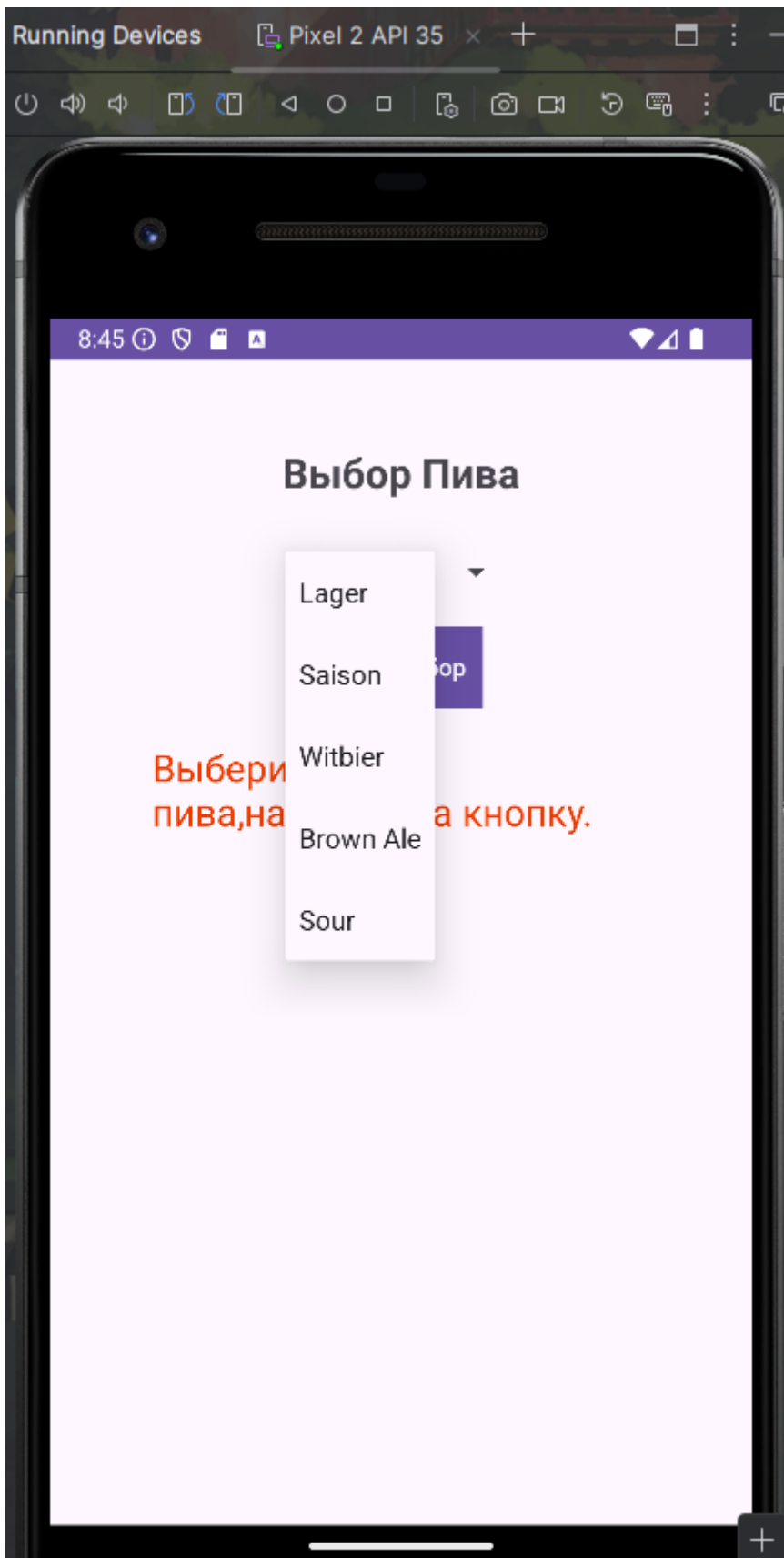
## ПРАКТИКА 3

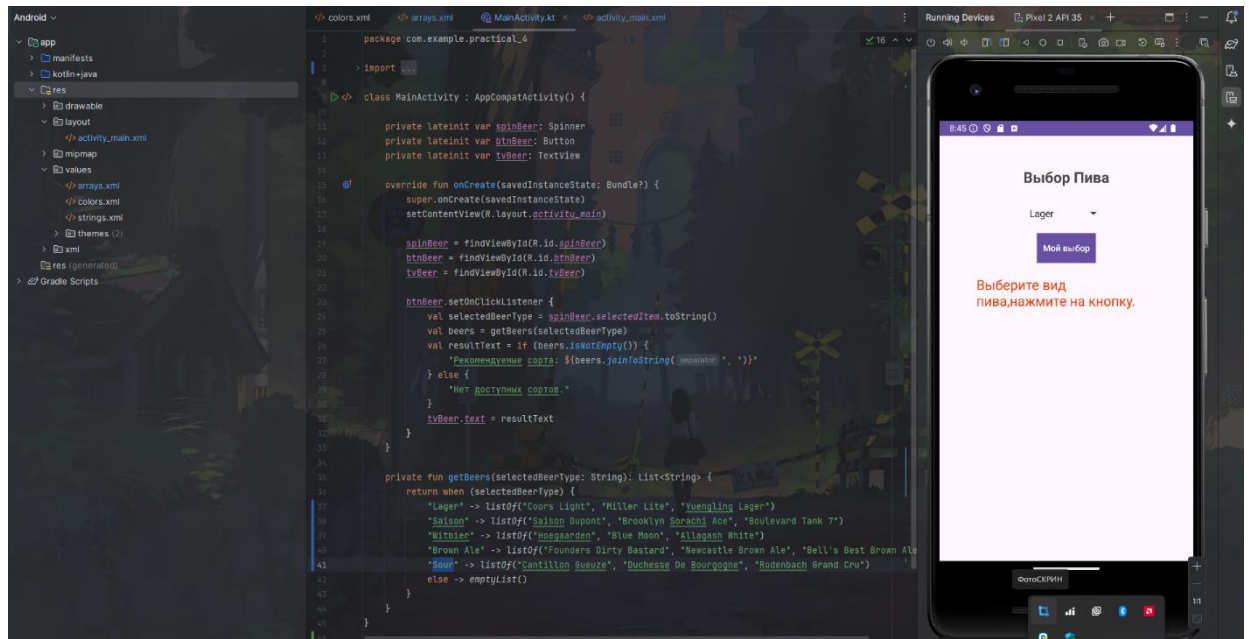


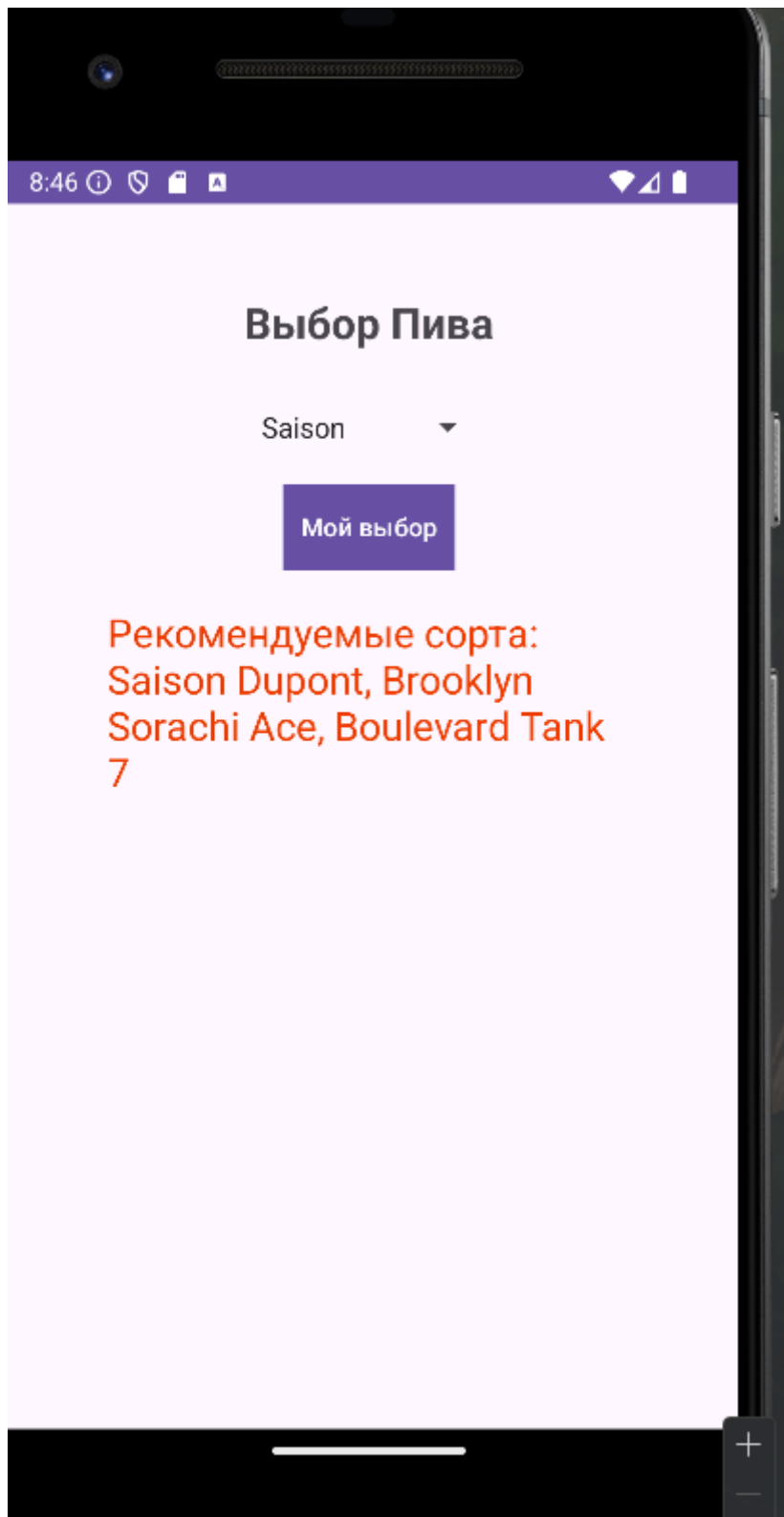
Проект состоит из двух активностей. В первой активности создается кнопка с ID btnIntent. При нажатии на кнопку происходит переход на вторую активность с помощью следующего кода:

```
val btn: Button = findViewById(R.id.btnIntent)
btn.setOnClickListener {
    intent = Intent(this, MainActivity2::class.java)
    startActivity(intent)
}
```

Практика 4







Для этого приложения с двумя активностями создаем макет и структуру следующим образом

Для этого приложения с двумя активностями создаем макет и структуру следующим образом:

1. Макет activity\_main.xml включает:

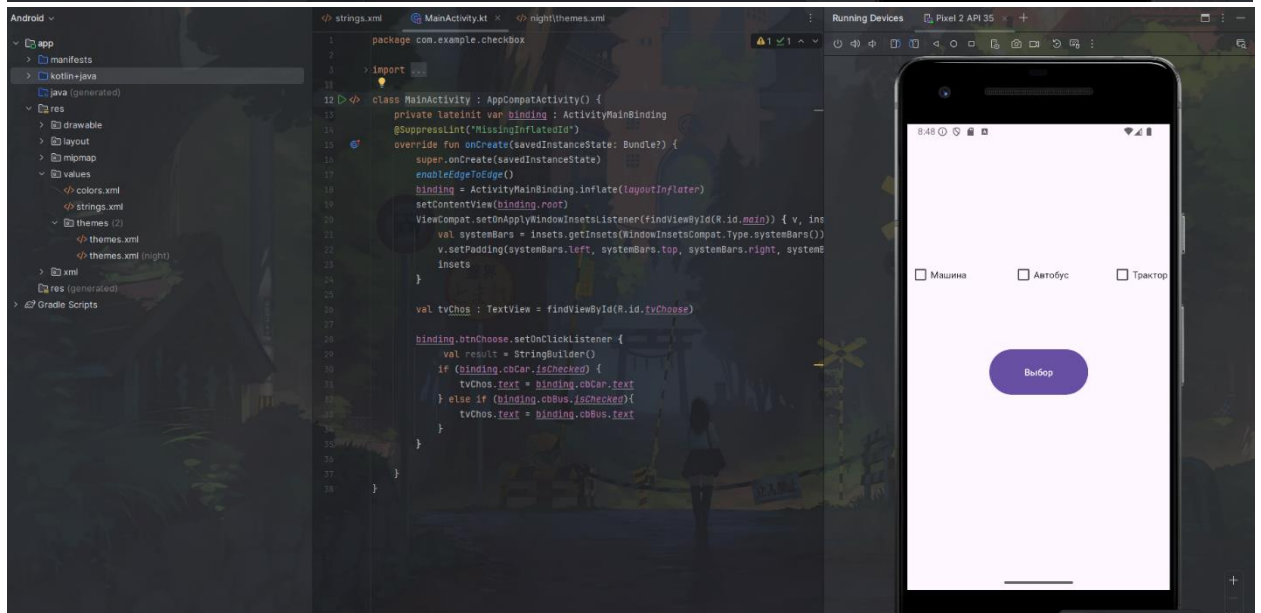
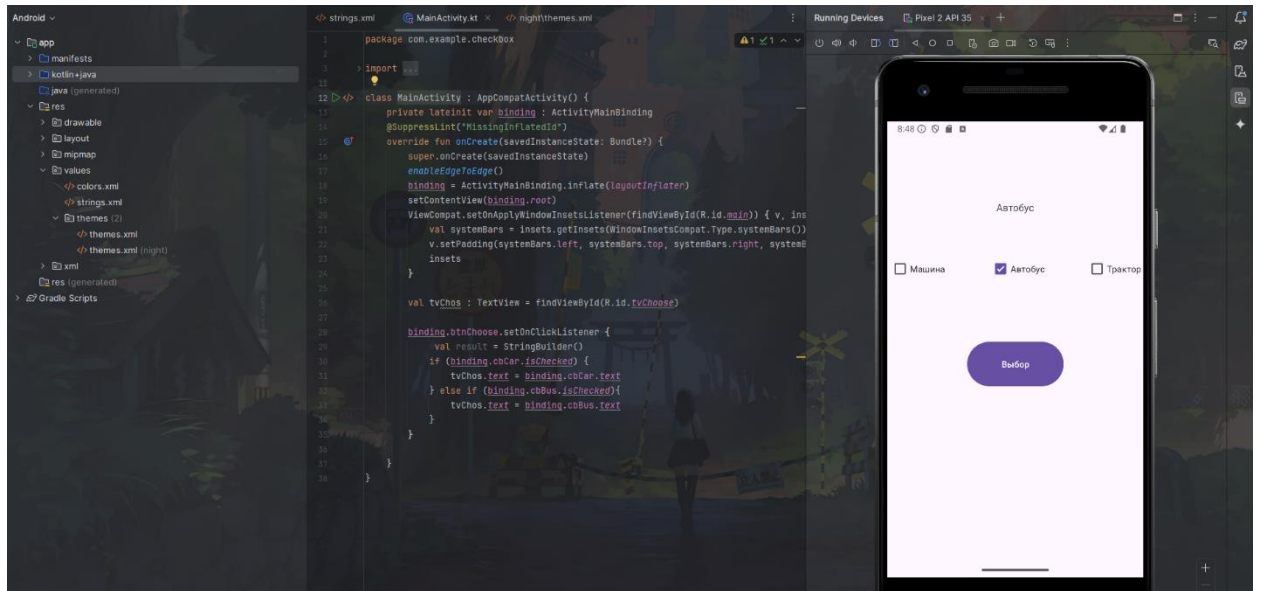
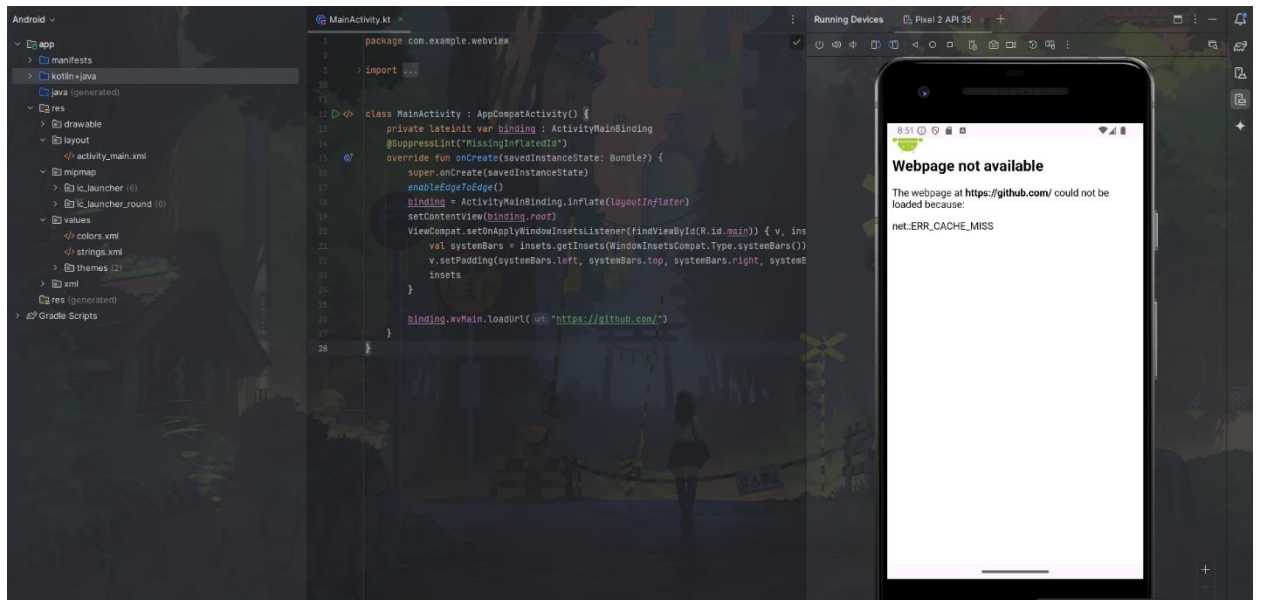
- Выпадающий список для выбора вида пива Spinner
- Кнопку для получения рекомендаций Button
- Текстовое поле для отображения результатов TextView

2. Строки в файле strings.xml хранят текстовые сообщения, такие как названия кнопок, подсказки и прочие строки для отображения.

3. Массивы в файле arrays.xml содержат список видов пива, которые можно выбрать в выпадающем списке.

4. Логика в файле MainActivity.kt обрабатывает выбор пользователя из Spinner и при нажатии кнопки обновляет текстовое поле, выводя рекомендации.

Практика 5 , 5.2



## 5. 1. Создание интерфейса:

- Я создал файл `activity_main.xml`, в котором разместил три `checkbox` для выбора: машина, автобус и трактор.

- Добавил кнопку для подтверждения выбора и текстовое поле для вывода результата.

## 2. Логика:

- В `MainActivity.kt` я применил `view binding` для удобства работы с элементами интерфейса.

- Написал обработчик для кнопки, который проверяет, какие `checkbox` были выбраны, и собирает их текст для отображения.

### 5.2 1. Разметка интерфейса:

- В файле `activity_main.xml` я добавил `WebView`, который занимает весь экран.

## 2. Загрузка страницы:

- В классе `MainActivity` я использовал метод `loadUrl` для загрузки веб-страницы с `GitHub`.

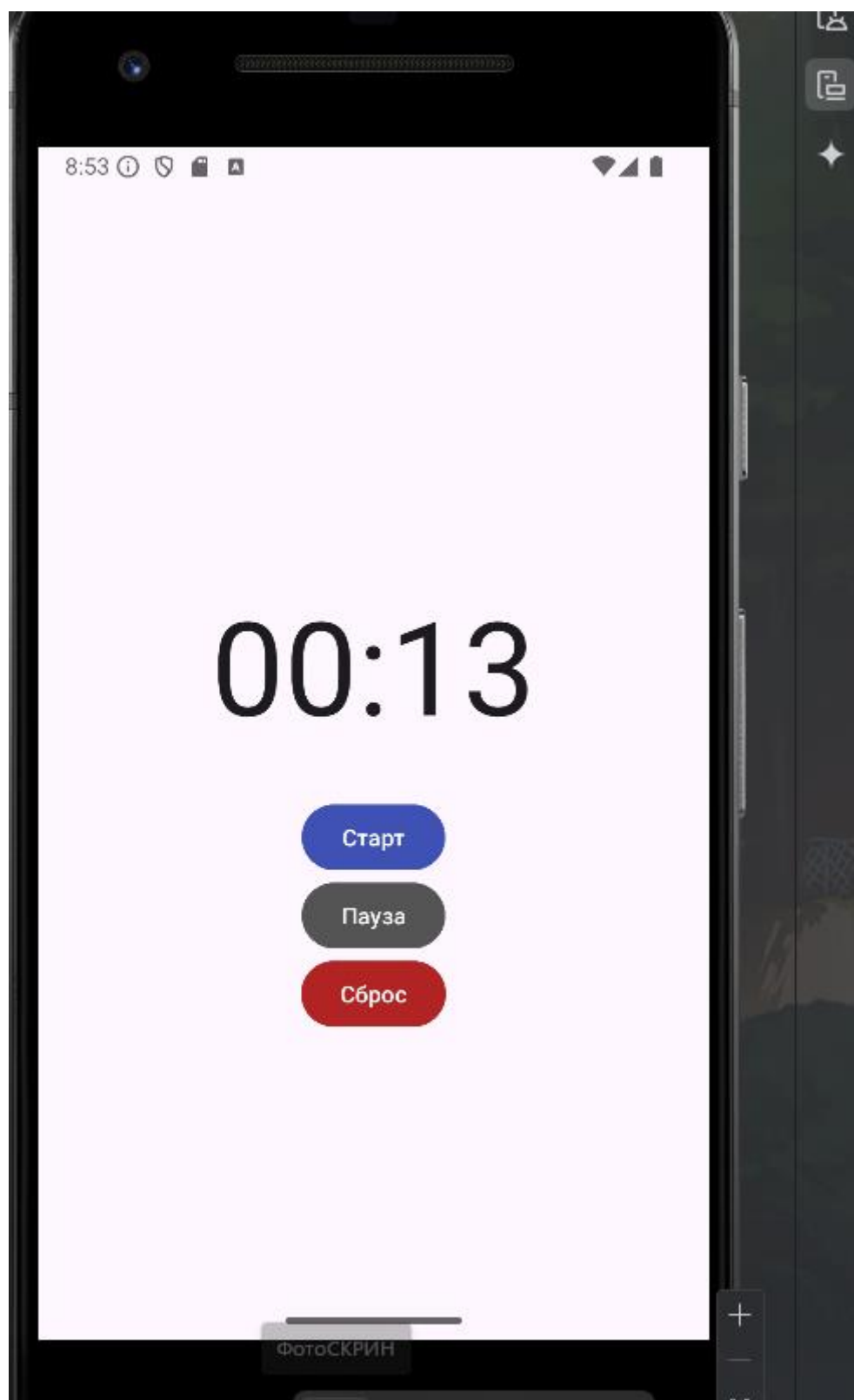
## 3. Обработка нажатий:

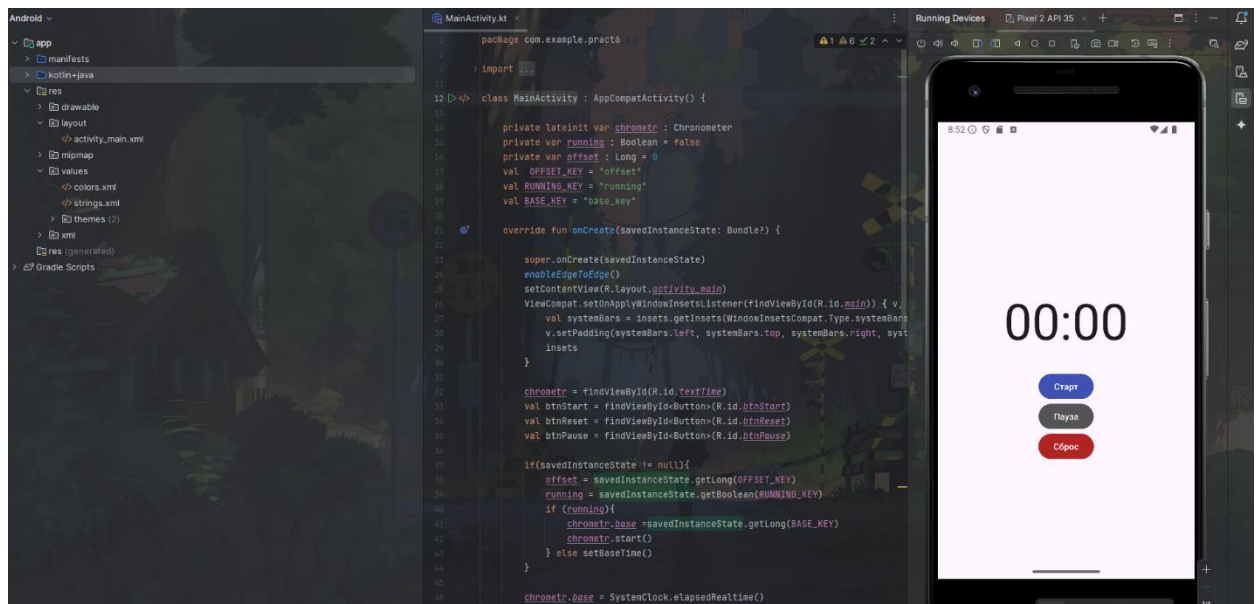
- Я внедрил `WebViewClient`, чтобы обрабатывать переходы по ссылкам и оставаться в приложении, а не открывать внешний браузер.



## Практика 6

```
12 class MainActivity : AppCompatActivity() {
21     override fun onCreate(savedInstanceState: Bundle?) {
46         chrometr.base = SystemClock.elapsedRealtime()
47
48         btnStart.setOnClickListener {
49             if(!running) {
50                 setBaseTime()
51                 chrometr.start()
52                 running = true
53             }
54         }
55
56         btnPause.setOnClickListener {
57             if (running){
58                 saveOffset()
59                 chrometr.stop()
60                 running = false
61             }
62         }
63
64         btnReset.setOnClickListener {
65             offset = 0
66             setBaseTime()
67             running = false
68         }
69
70     }
71
72     @
73     override fun onSaveInstanceState(savedInstanceState: Bundle) {
74         savedInstanceState.putLong("offset", offset)
75         savedInstanceState.putBoolean("running", running)
76         savedInstanceState.putLong("base_key", chrometr.base)
77         super.onSaveInstanceState(savedInstanceState)
78     }
79
80     private fun saveOffset() {
81         offset = SystemClock.elapsedRealtime() - chrometr.base
82     }
83
84     private fun setBaseTime() {
85         chrometr.base = SystemClock.elapsedRealtime() - offset
```





### 1. Добавление строковых ресурсов:

- Для кнопок и текстов добавил необходимые строки в ресурсы.

### 2. Макет интерфейса:

- В файле activity\_main.xml создал макет с использованием LinearLayout, добавив Chronometer и три кнопки: старт, пауза и сброс.

### 3. Реализация функционала:

- Для управления временем в Chronometer использовал свойства и методы, такие как base, start() и stop().

### 4. Проблема с ориентацией экрана:

- При повороте устройства приложение сбрасывало состояние секундомера, так как активность уничтожалась и создавалась заново. Это происходило из-за того, что значения свойств не сохранялись.

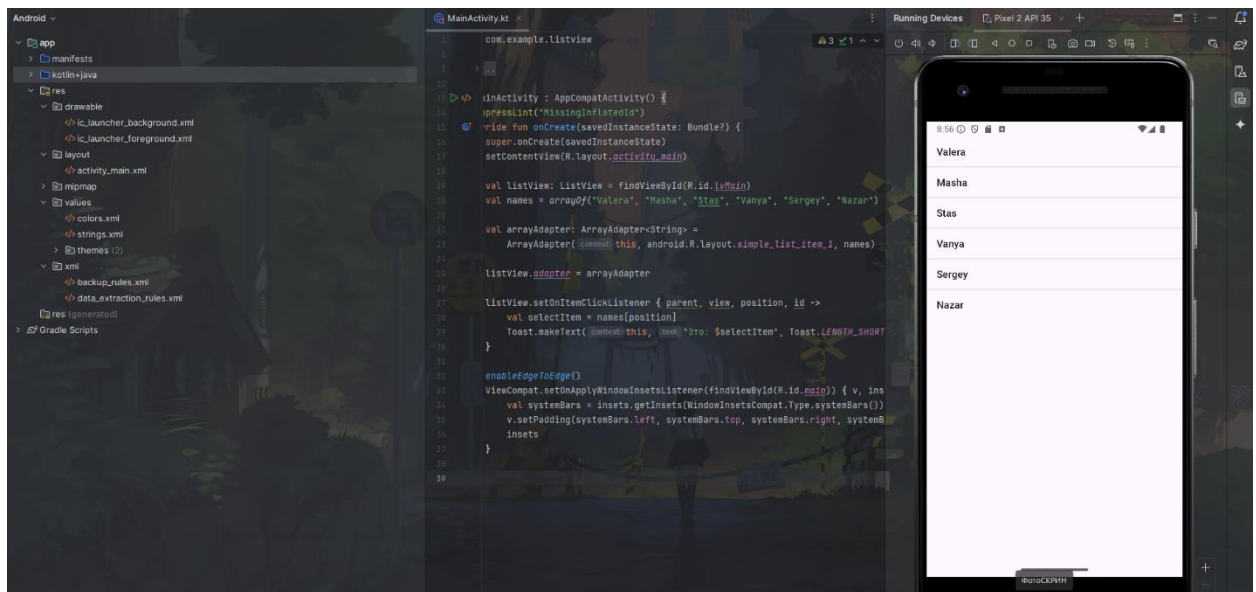
### 5. Жизненный цикл активности:

- Использовал методы жизненного цикла активности, такие как onCreate(), onStart(), onStop() и onDestroy(). Понял, как важно сохранять состояние активности.

### 6. Сохранение состояния:

- Для решения проблемы использовал объект Bundle, чтобы сохранять пары «ключ-значение», и методы put и get для сохранения и восстановления состояния секундомера.

## Практика 7



### 1. activity\_main.xml:

- В этом файле я объявил ListView внутри LinearLayout, который будет использоваться для отображения списка плейлистов.

### 2. list\_item.xml:

- Я создал отдельный XML-файл для дизайна элемента списка, который включает изображение и текст.

### 3. Создание класса данных:

- класс ListItem, который хранит информацию о каждом элементе списка, включая изображение, название и действие.

### 4. Адаптер:

- Я создал класс адаптера AdapterList, который наследуется от ArrayAdapter<ListItem>. В этом классе переопределил метод getView(), чтобы управлять отображением элементов списка.

### 5. Загрузка данных:

- В MainActivity.kt я создал список элементов items, состоящий из объектов ListItem, который будет передан адаптеру.

#### 6. Установка адаптера:

- Я получил доступ к ListView с помощью findViewById() и установил адаптер для отображения данных.

#### 7. Обработка нажатий:

- Я добавил обработчик кликов на элементы списка с помощью setOnItemClickListener, чтобы при нажатии выполнять определенные действия, например, отображать тост с информацией о выбранном элементе.

### Практика 8

9:00 ⓘ 🔒 📶 🔋



## мазда миата



Виды настроения мазды миата



Злая мазда миата

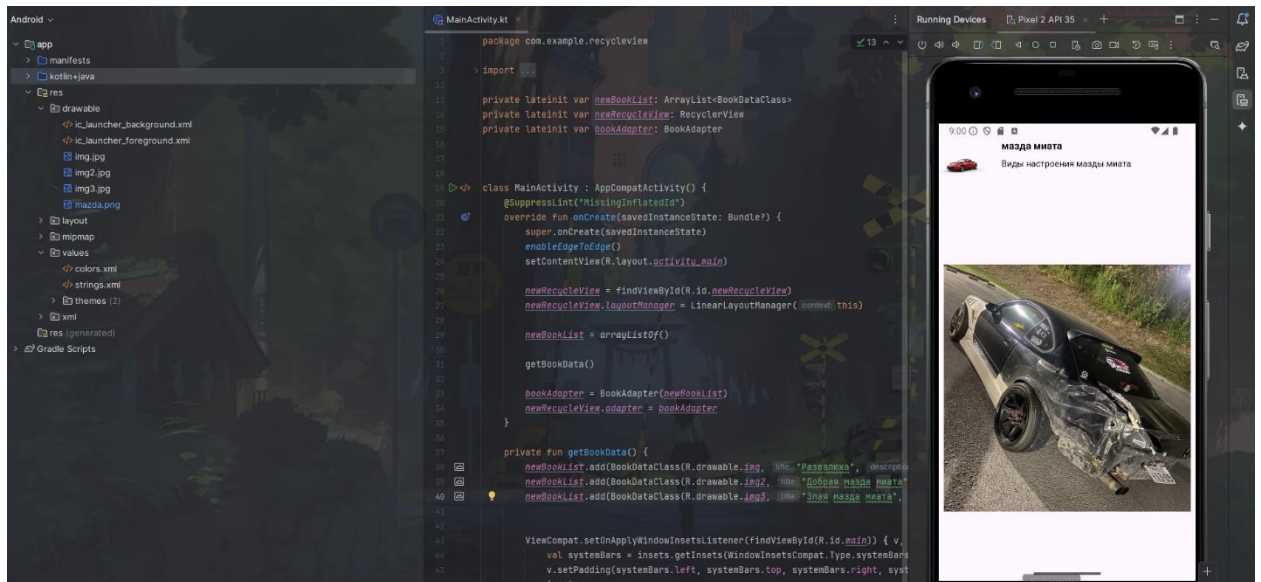




Добрая мазда миата  
мазда миата 2







### 1. activity\_main.xml:

- В этом файле я добавил элемент RecyclerView, который будет использоваться для отображения списка элементов (Машинок).

### 2. list\_item.xml:

- Я создал отдельный XML-файл для дизайна каждого элемента списка. Каждый элемент включает изображение (Мазда миата!) и два текстовых поля для названия и описания элемента.

### 3. Data class:

- DataClass для хранения информации о каждом элементе списка. В нем содержатся три свойства: ресурс изображения, название элемента и его описание.

### 4. Адаптер:

- MyAdapter, который наследуется от RecyclerView.Adapter. В нем реализованы методы для создания, привязки и подсчета элементов. Также создан вложенный класс MyViewHolder для эффективного управления ресурсами и представлением каждого элемента.

### 5. Загрузка данных:

- В MainActivity.kt я инициализировал массивы с изображениями и строковыми данными. Для упрощения работы использовал массивы из strings.xml.



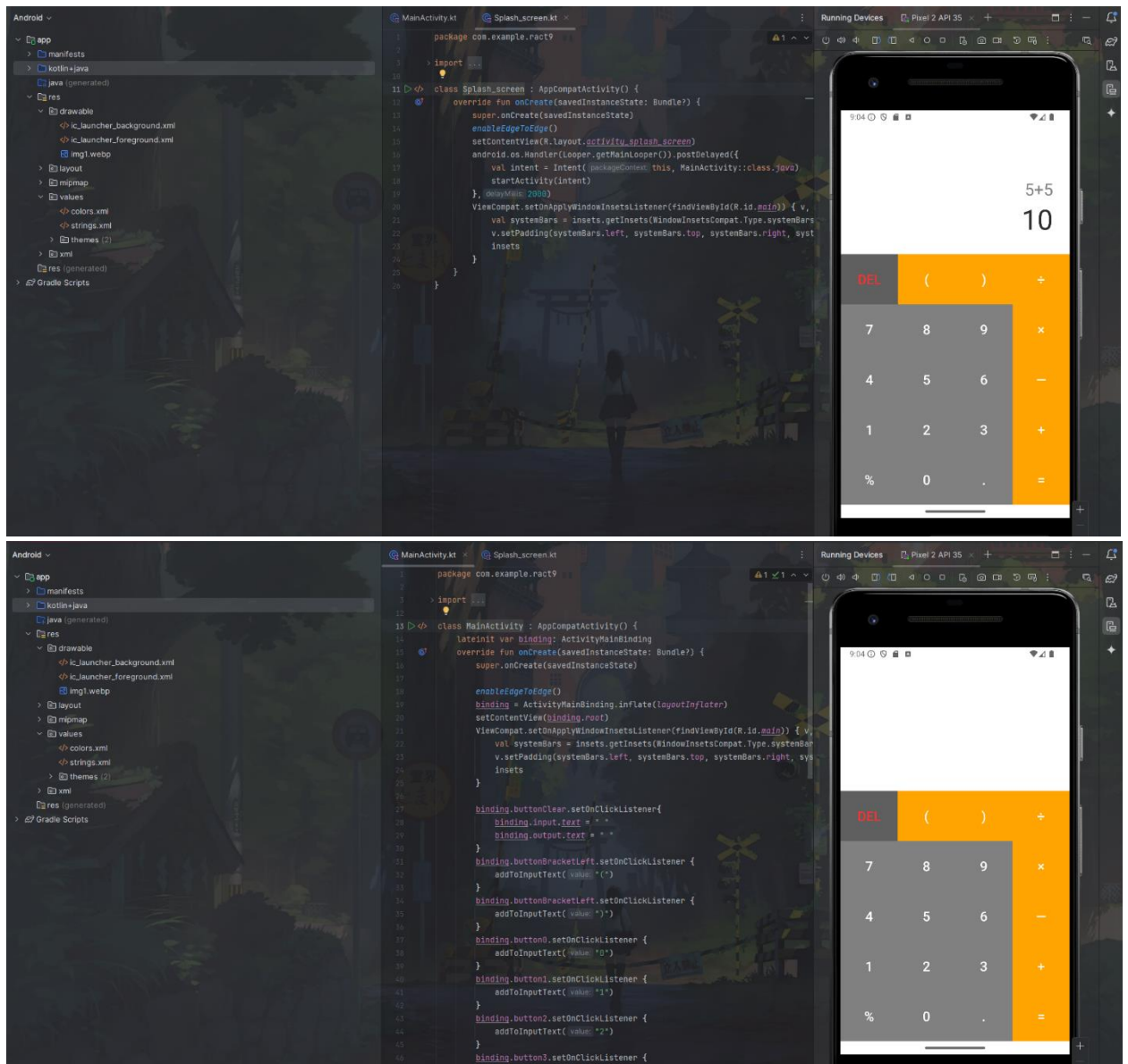
## 6. Установка менеджера компоновки:

- Для RecyclerView я применил LinearLayoutManager, чтобы элементы отображались в вертикальном списке, и указал, что размеры дочерних элементов не будут изменяться.

## 7. Наполнение списка данными:

- Я написал метод getUserdata(), который заполняет список newArrayList элементами из массивов, и затем установил адаптер для RecyclerView.

## Практика 9



## 1. Добавление цветов:

- В файл `colors.xml` я добавил необходимые цвета для оформления интерфейса приложения. Это включает основные цвета фона, текста и кнопок, а также стиль для кнопок.

## 2. Разработка интерфейса:

- В файле `activity_main.xml` использовал `TableLayout` для создания структуры калькулятора. Добавил текстовые поля для ввода и вывода, а также кнопки для цифр и операций.

## 3. Использование ViewBinding:

- Я подключил `ViewBinding` в `Gradle` и инициализировал его в `MainActivity`, чтобы упростить доступ к элементам интерфейса.

## 4. Обработка нажатий на кнопки:

- Для каждой кнопки я установил обработчики нажатий. При нажатии кнопки соответствующий символ добавляется в поле ввода.

## 5. Подключение библиотеки `exp4j`:

- Я добавил зависимость `exp4j` в `build.gradle.kts`. Эта библиотека помогает обрабатывать математические выражения в формате строк.

## 6. Вычисление результата:

- Я реализовал метод `showResult()`, который использует `exp4j` для вычисления введенного выражения. Обрабатываются как правильные выражения, так и ошибки, с соответствующим выводом результата или сообщения об ошибке.

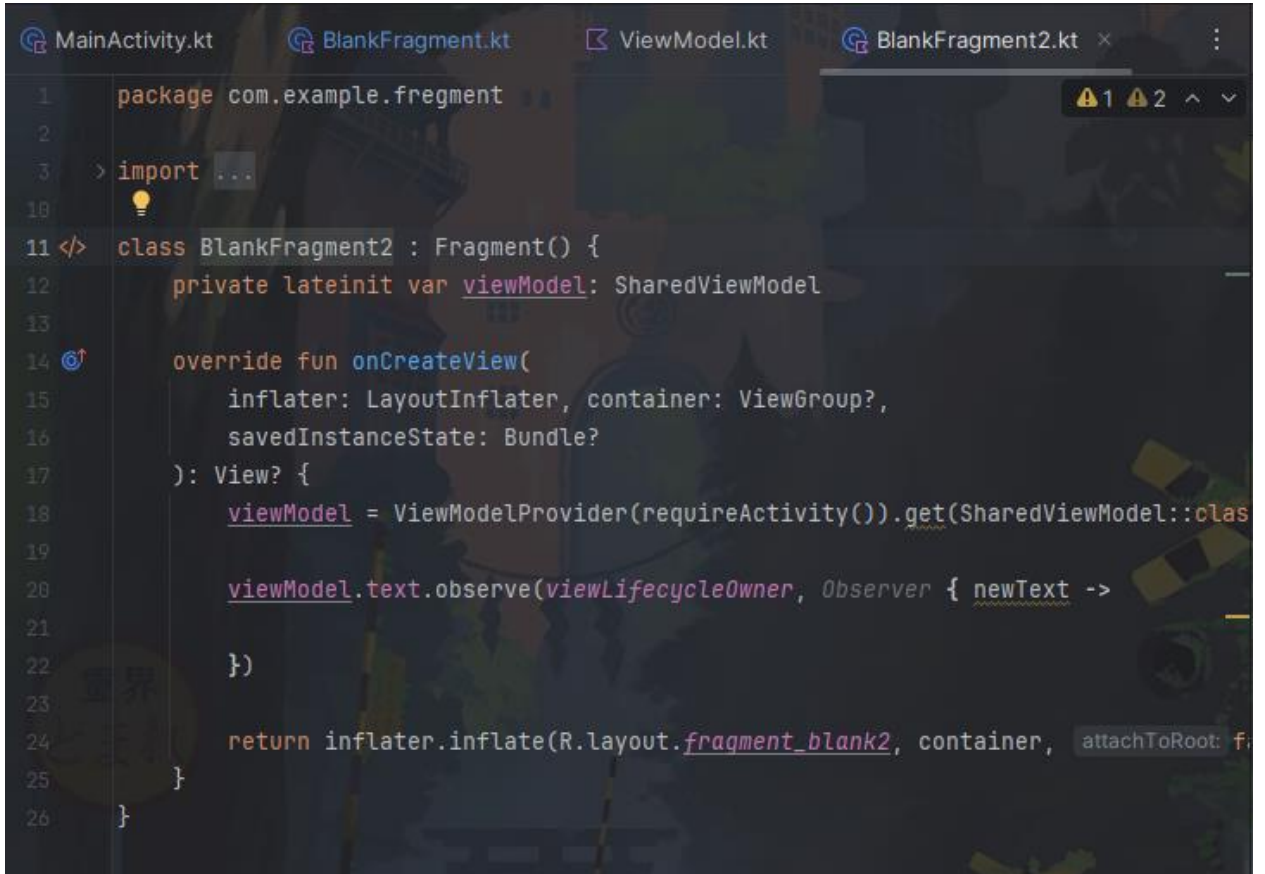
## 7. Создание заставки:

- Я создал новую активность `Splash_screen`, которая отображает заставку перед запуском основного калькулятора. Заставка длится 2 секунды, после чего происходит переход к `MainActivity`.

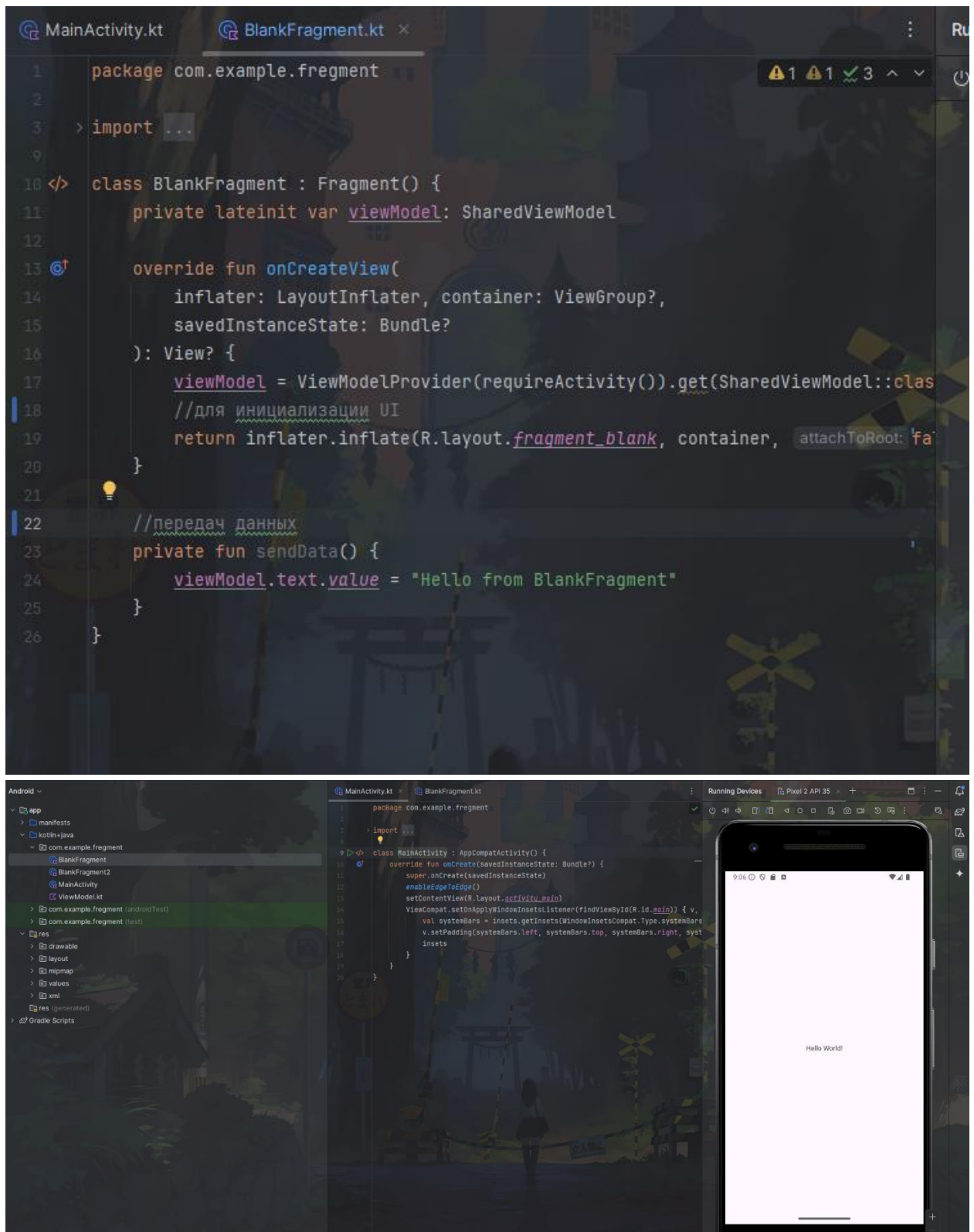
## 8. Изменения в манифесте:

- Я внес изменения в `AndroidManifest.xml`, чтобы указать, что `Splash_screen` является начальной активностью приложения.

## Пратика 10



```
1 package com.example.fregment
2
3 > import ...
4
5
6
7
8
9
10
11 </> class BlankFragment2 : Fragment() {
12     private lateinit var viewModel: SharedViewModel
13
14     override fun onCreateView(
15         inflater: LayoutInflater, container: ViewGroup?,
16         savedInstanceState: Bundle?
17     ): View? {
18         viewModel = ViewModelProvider(requireActivity()).get(SharedViewModel::class)
19
20         viewModel.text.observe(viewLifecycleOwner, Observer { newText ->
21
22         })
23
24         return inflater.inflate(R.layout.fragment_blank2, container, attachToRoot: false)
25     }
26 }
```



## 1. Подключение библиотек:

- Для начала я подключил библиотеки для работы с фрагментами и ViewModel, чтобы использовать их функционал в проекте.

## 2. Создание фрагментов:

- Я создал два пустых фрагмента и удалил ненужный код, чтобы структура была проще.

### 3. activity\_main.xml:

- В этом файле я добавил два `FrameLayout`, которые будут контейнерами для фрагментов. Это позволяет динамически заменять содержимое.

### 4. Реализация MainActivity:

- В `MainActivity` я использовал `ViewBinding` для работы с элементами интерфейса. В методе `onCreate` заменил первый `FrameLayout` на первый фрагмент.

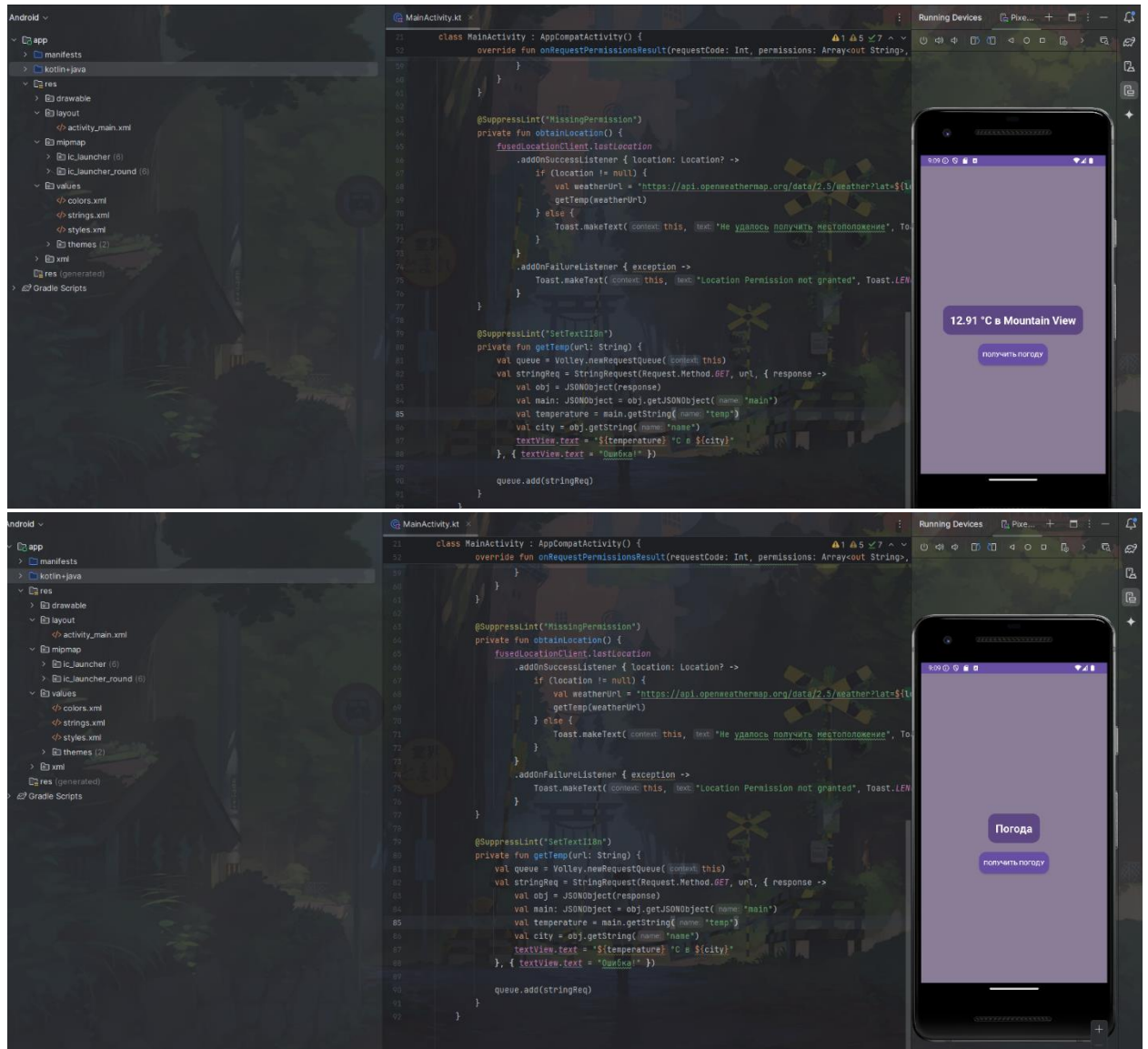
### 5. Упрощение кода с помощью функции:

- Я создал функцию `openFragment`, которая помогает легче добавлять фрагменты и делает код чище.

### 6. Передача данных между фрагментами:

- Для передачи данных между фрагментами я использовал `ViewModel`. Это помогает сохранить данные при изменении конфигурации, например, при повороте экрана.

## Практика 11



### 1. Регистрация и получение API-ключа:

- Зарегистрировались на сайте OpenWeatherMap и получили API-ключ, который используется для запросов к сервису погоды.

### 2. Разрешения в Android Manifest:

- В `AndroidManifest.xml` добавлены разрешения для доступа к интернету и местоположению:

- `ACCESS_COARSE_LOCATION` — для получения приблизительного местоположения.

- `ACCESS_FINE_LOCATION` — для точного местоположения.

- `INTERNET` — для доступа к интернету.

### 3. Дизайн интерфейса:

- Создал макет `activity_main.xml`, включающий кнопку для получения погоды и текстовое поле для вывода температуры и влажности.

### 4. Подключение библиотек:

- Добавил зависимости для работы с местоположением и сетевыми запросами:
  - `implementation(libs.play.services.location)` — для работы с местоположением.
  - `implementation(libs.volley)` — для выполнения HTTP-запросов.

### 5. Реализация MainActivity:

- В `MainActivity` инициализировал элементы интерфейса и добавили обработчик кнопки. При нажатии запускается процесс получения местоположения и, если всё успешно, выполняется запрос к API погоды.

### 6. Проверка разрешений:

- Реализовал методы для проверки и запроса разрешений на доступ к местоположению. Если разрешения даны, запускается метод `obtainLocation()` для получения координат.

### 7. Получение местоположения:

- Использовал `FusedLocationProviderClient` для получения последнего местоположения устройства. С этими координатами создаём URL для запроса к API.

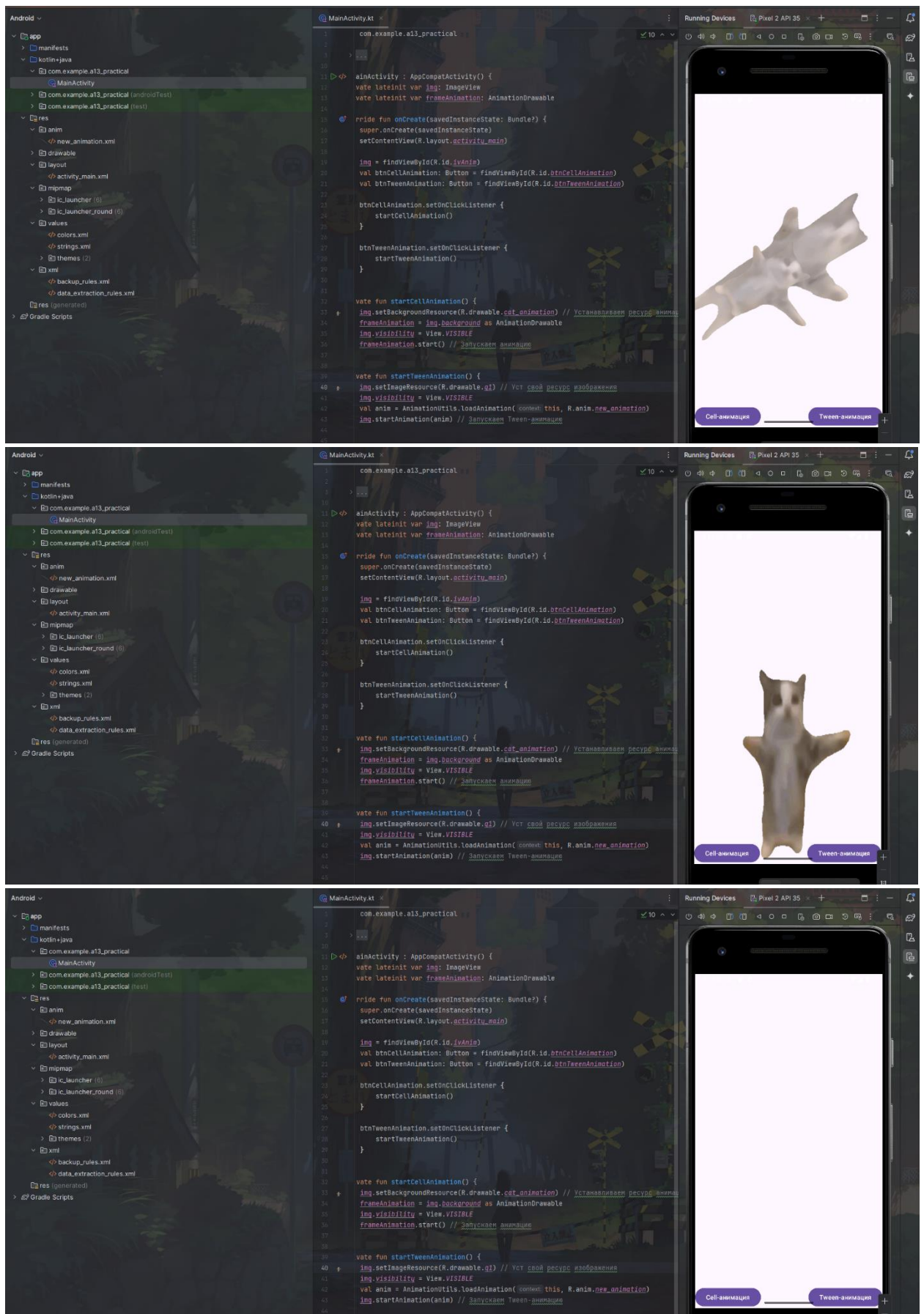
### 8. Выполнение запроса к API:

- Реализовал метод `getTemp()`, который выполняет запрос с помощью библиотеки `Volley`. Извлекаем данные о температуре и влажности из JSON-ответа.

### 9. Отображение данных:

- Выводим полученные данные в `TextView` на экране.





cell-анимация



#### 1. Подготовка изображений:

- Я добавил несколько кадров анимации в папку с ресурсами.

#### 2. Создание xml-файла:

- В папке drawable был создан файл анимации, в котором указаны все кадры и время их отображения.

#### 3. Добавление элемента:

- В разметке интерфейса добавил элемент ImageView, который будет показывать анимацию.

#### 4. Запуск анимации:

- В коде основной активности настроил запуск анимации при открытии приложения.

#### 5. Горизонтальная ориентация:

- В манифесте приложения установил горизонтальную ориентацию экрана.

### tween-анимация

#### 1. Создание папки anim:

- В папке с ресурсами создал новую папку для хранения анимационных файлов.

#### 2. Создание xml-файла для анимации:

- В папке anim создал файл с эффектами анимации, такими как масштабирование и вращение.

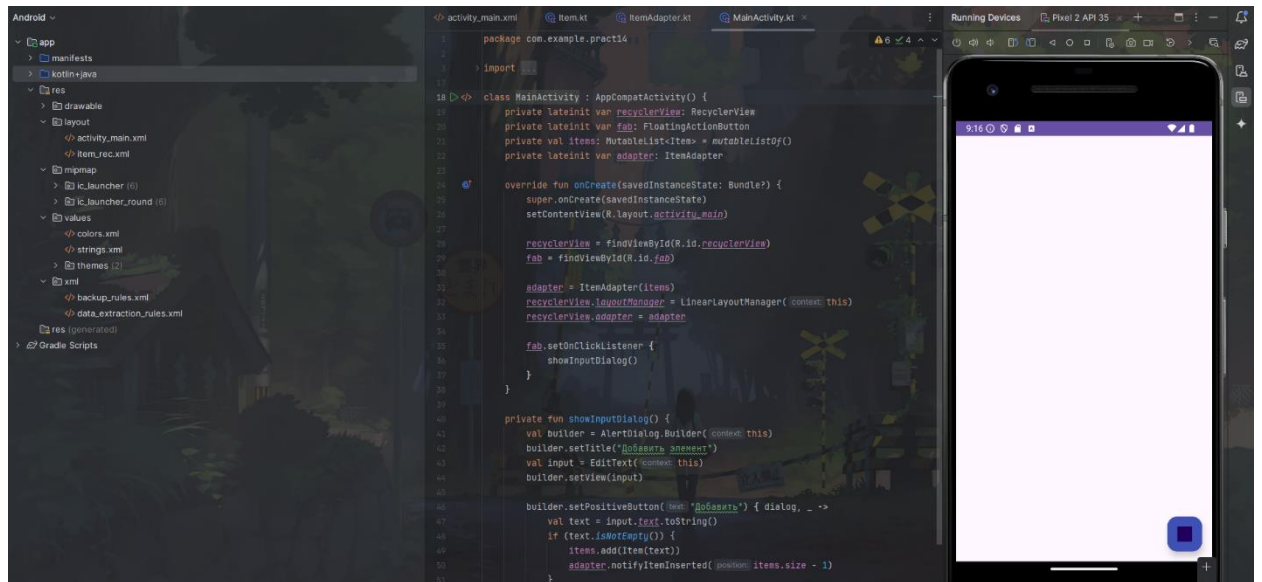
#### 3. Добавление элемента:

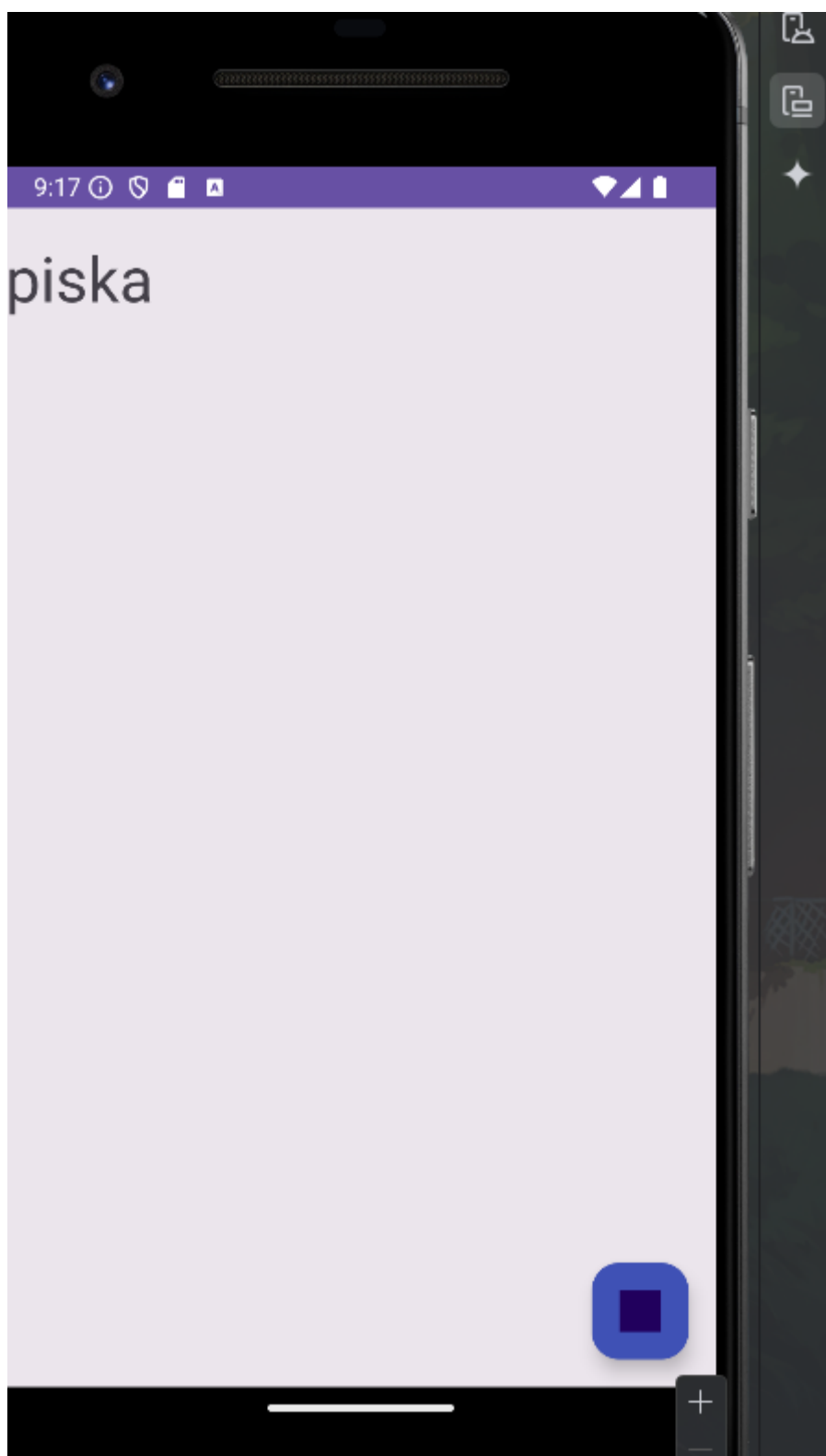
- В разметке второй активности добавил еще один элемент ImageView для анимации.

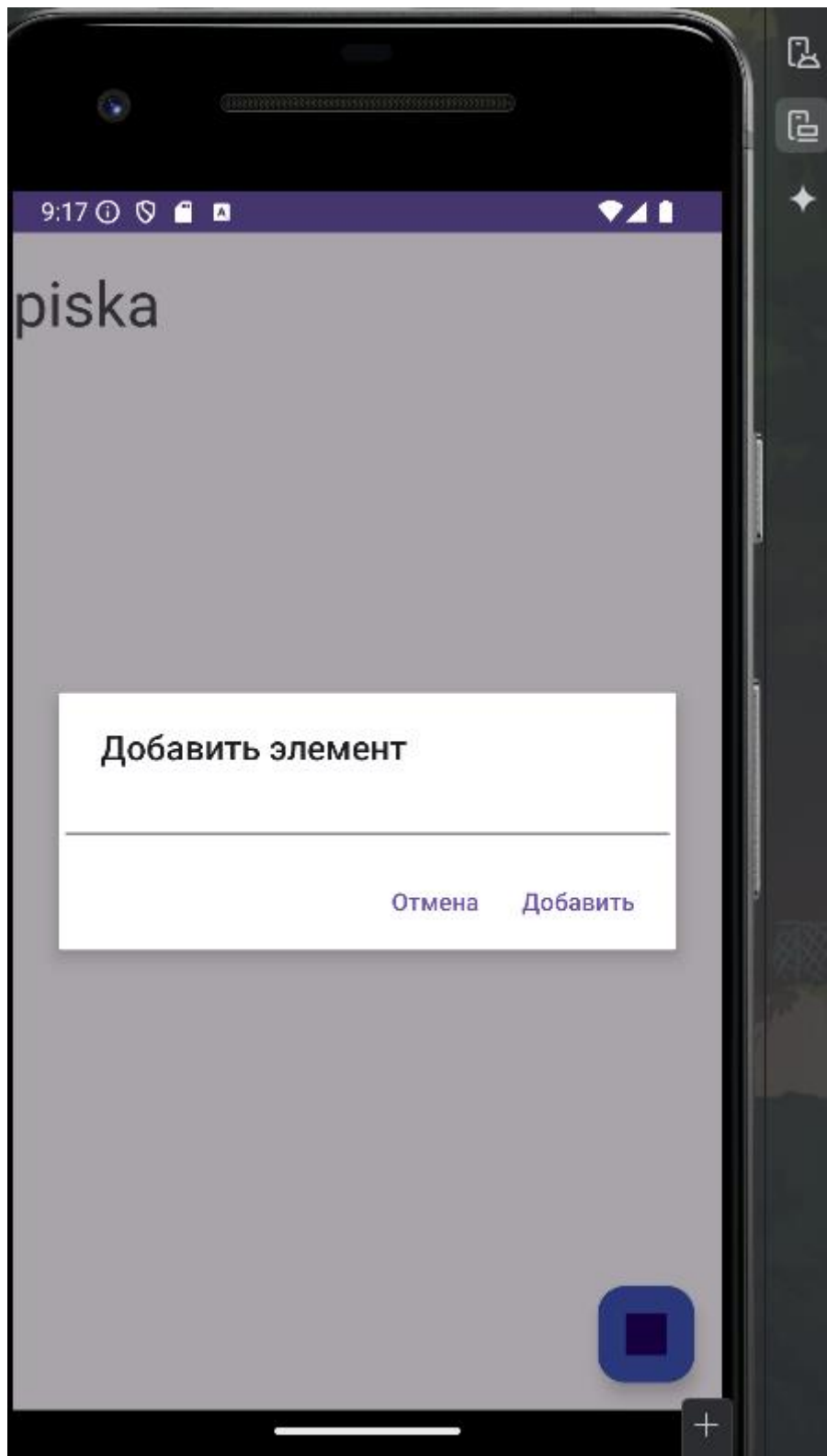
#### 4. Запуск анимации:

- В коде второй активности внедрил запуск tween-анимации.

Практика 14







1. Обновление разметки activity\_main.xml:

- Я добавил в activity\_main.xml элемент RecyclerView для отображения списка и FloatingActionButton для добавления новых элементов.

- RecyclerView используется для отображения списка, а FloatingActionButton — для добавления новых элементов.

## 2. Создание класса Item.kt:

- Я создал класс Item, который представляет собой data class с одним свойством text. Этот класс хранит данные, которые будут отображаться в списке.

## 3. Создание файла item\_rec.xml:

- В папке res/layout я создал файл item\_rec.xml, который определяет внешний вид элементов списка. В нем используется TextView для отображения текста.

## 4. Создание класса адаптера ItemAdapter.kt:

- Я создал класс ItemAdapter, который наследуется от RecyclerView.Adapter. Этот класс отвечает за создание и привязку элементов списка к RecyclerView.

- Адаптер управляет отображением данных в списке и обновляет интерфейс, когда данные меняются.

## 5. Изменение MainActivity.kt:

- В MainActivity я добавил компоненты RecyclerView и FloatingActionButton.

- Реализовал список элементов, который отображается в RecyclerView.

- Добавил обработчик для FloatingActionButton, который открывает диалоговое окно для добавления нового элемента.

- Новый элемент добавляется в список и отображается в RecyclerView после нажатия кнопки "Добавить".