

Geekbrains

Погудин Сергей Александрович

“Разработка мобильного Android-приложения “Прогноз погоды””

Программа: Разработчик

Специализация: Android-разработка

Ижевск

2024

## Содержание

<b>Определения .....</b>	<b>4</b>
<b>Введение .....</b>	<b>6</b>
<b>1 Формирование требований к мобильному приложению и выбор подходящего погодного API .....</b>	<b>7</b>
1.1 Основные характеристики погодного приложения .....	7
1.2 Архитектура мобильного приложения .....	8
1.2.1 Анализ архитектуры мобильного приложения.....	8
1.2.2 Архитектура приложения погода.....	11
1.3 Выбор подходящего погодного API.....	14
<b>2 Изучить и подобрать технологии для реализации мобильного приложения .....</b>	<b>17</b>
2.1 Выбор языка разработки .....	17
2.2 Выбор интеграционной среды разработки .....	20
2.3 Материально-техническое обеспечение .....	23
2.4 Последовательность разработки проекта .....	24
<b>3 Реализация мобильного Android-приложения “Прогноз погоды” .....</b>	<b>26</b>
3.1 Подготовительный этап.....	26
3.1.1 Создать новый проект .....	26
3.1.2 Подключение библиотек и настройки .....	27
3.1.3 Добавить логотип (иконку) приложения.....	29
3.2 ViewModel.....	30
3.3 DataClass.....	30
3.4 Проверка разрешения .....	31
3.5 Создание основного экрана.....	32
3.5.1 Разметка основного экрана .....	32
3.5.2 Создание HoursFragment и DaysFragment .....	34
3.6 Создание data class WeatherModel .....	36
3.7 Создание разметки .....	36
3.8 Создание RecyclerViewAdapter .....	37
3.9 Получение данных через API.....	42

3.9.1	Функция <code>requestWeatherData</code> .....	42
3.9.2	Создать url для запроса .....	42
3.9.3	<code>Volley.newRequestQueue(context)</code> .....	43
3.9.4	Создать запросы <code>StringRequest()</code> .....	43
3.10	Парсинг данных полученных с API .....	44
3.10.1	Парсинг данных полученных с API .....	44
3.10.2	Парсинг массива, полученного с API .....	45
3.11	Показ прогноза на экране .....	48
3.11.1	Перенос кода в новую функцию <code>parseCurrentData()</code> .....	48
3.11.2	Функция <code>parseCurrentData()</code> для получения списка .....	49
3.12	Показ прогноза погоды по часам.....	50
3.12.1	Передача <code>WeatherModel</code> на <code>HoursFragment</code> .....	50
3.12.2	Функция <code>getHoursList</code> .....	50
3.13	Список прогноза по дням .....	51
3.14	Обработка нажатий на список дней.....	52
3.15	Получение прогноза погоды по местоположению .....	55
3.16	Проверка включения GPS .....	57
3.16.1	Функция <code>sLocationEnabled()</code> .....	57
3.16.2	<code>AlertDialog</code> .....	58
3.16.3	Функция <code>checkLocation()</code> .....	59
3.16.4	Использование функций <code>checkLocation()</code> и <code>onResume()</code> .....	59
3.17	<code>AlertDialog</code> выбор города .....	60
3.18	Запуск приложения .....	62
<b>4</b>	<b>Тестирование и подготовка к релизу .....</b>	<b>64</b>
4.1	Подключение смартфона к Android Studio.....	64
4.1.1	Настройка параметров Android Studio .....	64
4.1.2	Настройки параметров смартфона .....	65
4.2	Сохранить установочный файл приложения – apk файл .....	67
4.3	Опубликовать приложение в Google Play .....	68
	<b>Заключение .....</b>	<b>69</b>
	<b>Список использованных источников .....</b>	<b>70</b>

## Определения

**Android** – операционная система для смартфонов, планшетов, электронных книг.

**Android-приложение** (Android app) — это приложение, разработанное для платформы Android, которое работает на мобильных устройствах, таких как смартфоны, планшеты и ноутбуки.

**Архитектура мобильного приложения** – это структурный принцип, по которому создано приложение. Каждому архитектурному виду свойственны свои характеристики, свойства и отношения между компонентами.

**API** (Application Programming Interface) – что значит программный интерфейс приложения, это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов.

**Интерфейс** – это набор инструментов, который позволяет пользователю взаимодействовать с программой.

**Google Play** – это централизованный сервис компании Google, который предоставляет пользователям доступ к многочисленным приложениям, играм, книгам, музыке и фильмам для устройств на операционной системе Android.

**Консоль разработчика Google Play** – это веб-платформа, где разработчики могут публиковать, управлять и распространять свои приложения для Android.

**Presenter** – это объект, который управляет отображением данных через специальный интерфейс View.

**Парсер (parser)** – это программа, сервис или скрипт, который собирает данные с указанных веб-ресурсов, анализирует их и выдает в нужном формате.

**Java Development Kit (JDK)** – это пакет программного обеспечения, который включает в себя JRE и дополнительные инструменты для разработки приложений на Java.

**Компилятор** – программа, переводящая написанный на языке программирования текст в набор машинных кодов.

**Асинхронное программирование** – это концепция программирования, при которой результат выполнения функции становится доступным не сразу, а через некоторое время в виде асинхронного вызова.

**IDE** (Integrated Development Environment) интегрированная среда разработки – это набор программных инструментов, которые используются для создания программного обеспечения.

**SDK** (Software Development Kit) – «комплект для разработки программного обеспечения») – набор инструментов для разработки программного обеспечения, объединённый в одном пакете.

**XML** (eXtensible Markup Language – расширяемый язык разметки) – это язык программирования для создания логической структуры данных, их хранения и передачи в виде, удобном и для компьютера, и для человека.

**Empty Views Activity** – данный тип проекта значит, что все просто, нет лишних файлов, нет фрагментов.

**RelativeLayout** – это объект, отображающий дочерний ViewGroup элемент View элементы в относительные позиции.

**layout\_width="match\_parent"** – растяжение по ширине родительской группы.

**layout\_height="wrap\_content"** – по вертикали “по вписанному содержанию”.

**android:padding="8dp"** – пробел/отступ.

**android:gravity="center"** – выровнять по центру.

**TextView** – текстовое вложение (виджит) предназначено для отображения текста без возможности редактирования его пользователем.

**LinearLayout** – простейший контейнер - объект. ViewGroup, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали.

## **Введение**

В качестве дипломного проекта выбрано создание мобильного Android-приложения для просмотра прогноза погоды, которое представит пользователю: текущую температуру, максимальную/минимальную температуру на день; осадки; температуру и осадки на день по часам; предположительный прогноз погоды на ближайшие два дня по месту положения пользователя.

**Цель:** разработкой Android-приложения для просмотра основные параметров погоды закрепить на практике жизненные циклы разработки приложения, как: планирование; выбор архитектуры мобильного приложения; выбор подходящего погодного API; разработка графического дизайна и интерфейса приложения; подключение к сети и выгрузка требуемых параметров; запуск приложения на смартфоне; подготовка к релизу.

Многие популярные приложения прогноза погоды в Google Play содержат много рекламы или требуют большого количества разрешений, или содержат функционал, который большинство пользователей не используют. Поэтому в этом проекте будет разработано приложение с простым и минимальным интерфейсом, показывающий пользователю основные параметры погоды.

**Задачи:** для достижения данной цели требуется решить следующие задачи:

- формирование требований к мобильному приложению.
- изучить и подобрать технологии для реализации мобильного приложения.
- реализация данного мобильного приложения.

## **1 Формирование требований к мобильному приложению и выбор подходящего погодного API**

### **1.1 Основные характеристики погодного приложения**

Чтобы сформулировать структуру и последовательность разработки данного мобильного приложения необходимо проанализировать характеристики и требования к мобильному приложению.

При создании погодного приложения, важно включить функции, которые предоставляют основные параметры погоды (температура, скорость ветра, атмосферное давление, влажность, осадки) и повышают удобство использования. Существует множество полезных функций, которые можно добавить в погодное приложение, вот некоторые из ключевых аспектов, которые следует учесть при разработке погодного приложения [9, 11, 22]:

1. **Прогнозы погоды:** пользователи ценят погодные приложения за точные прогнозы. Предлагайте почасовые, ежедневные и еженедельные прогнозы и отображайте такую информацию, как температура, влажность, скорость ветра и количество осадков.

2. **Обновления в реальном времени:** доступ к информации о погодных условиях в реальном времени очень важен для пользователей, особенно во время сильных погодных явлений. Убедитесь, что ваше приложение предоставляет мгновенные обновления по мере изменения погодных условий.

3. **Радарные и спутниковые изображения:** визуальные представления погоды, такие как радиолокационные и спутниковые изображения, помогают пользователям понять и отслеживать погодные условия. Используйте высококачественные изображения, которые легко понять и сориентироваться.

4. **Оповещения о неблагоприятной погоде:** держите своих пользователей в курсе серьезных погодных ситуаций, таких как штормы, ураганы и наводнения. Отправляйте push-уведомления и оповещения, чтобы помочь им оставаться готовыми и в безопасности.

5. **Поддержка нескольких местоположений:** возможность сохранять и отслеживать погоду в нескольких местах полезна для тех, кто путешествует или имеет семью и друзей в разных регионах. Упростите пользователям переключение между местоположениями и добавление новых в их список.

6. **Настраиваемые параметры:** дайте пользователям возможность персонализировать работу с приложением, предлагая такие настройки, как единицы измерения (по Цельсию или Фаренгейту), язык и настройка темы.

7. **Социальный обмен:** дайте пользователям возможность делиться обновлениями погоды и предупреждениями с друзьями и близкими через социальные сети и приложения для обмена сообщениями [9, 22].

## 1.2 Архитектура мобильного приложения

### 1.2.1 Анализ архитектуры мобильного приложения

В современном мире существует три архитектурных паттерна – MVC, MVP, MVVM.

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Режим MVC является одним из самых классических режимов разработки, который разделен на три части модель, вид, контроллер [1, 12, 14, 21].

Модель (Model) – представляет данные. Моделью может быть один объект или некоторая структура объектов.

Представление (View) является визуальным представлением своей модели. Обычно оно выделяет одни атрибуты модели и подавляет другие. Таким образом, представление действует как фильтр представления.

Контроллер (Controller) является связующим звеном между пользователем и системой. Он предоставляет пользователю входные данные, организуя отображение соответствующих представлений в соответствующих местах на



экране. Он предоставляет средства для пользовательского вывода, предоставляя пользователю меню или другие средства предоставления команд и данных. Контроллер получает такой пользовательский вывод, переводит его в соответствующие сообщения и передает эти сообщения одному или нескольким представлениям. Контроллер не обновляет представление напрямую.

Архитектура MVC для данного приложения не подходит, так как:

- приложение на этой архитектуре сложно тестировать;
- сложно реализовать в чистом виде;
- непонятно что отвечает за UI;
- код контроллера разрастается.

На рисунке 1 можно увидеть схему архитектуры MVC.

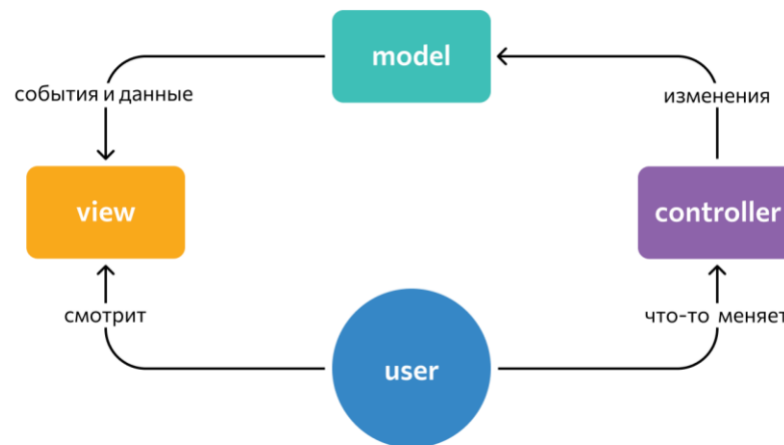


Рисунок 1 – Схема MVC

Что бы реализовать чистый MVC необходимо приложить большие усилия.

Проблемы MVC хорошо решены в архитектурах MVP и MVVM.

Model-View-Presenter (MVP) – шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса.

Элемент Presenter в данном шаблоне берёт на себя функциональность

посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса так же, как в других шаблонах обычно отвечает представление [1, 14].

Архитектурный паттерн MVP предназначен главным образом для улучшения паттерна архитектуры MVC в Android. Наиболее отличительным моментом между MVP и MVC является то, что нет прямой связи между Model и View. Между ними существует разрыв (Рисунок 2). Это уровень Presenter, который отвечает за регулирование косвенного взаимодействия между View и Model [1, 12, 21].

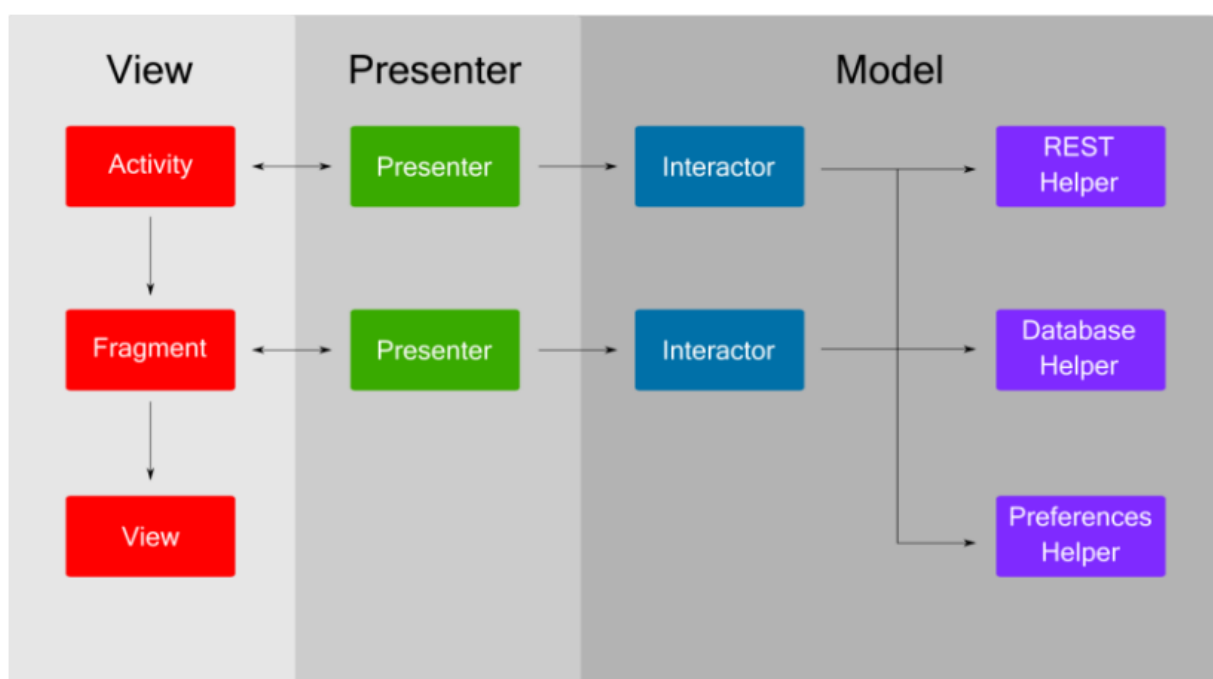


Рисунок 2 – Схема MVP

Model/View/ViewModel – это вариант модели/представления/контроллера (MVC), адаптированный для современных платформ разработки пользовательского интерфейса, где представление является обязанностью дизайнера, а не классического разработчика. Шаблон архитектуры MVVM (Рисунок 3) в основном нацелен на улучшение шаблонов архитектуры переднего плана и архитектуры MVC, снижение нагрузки на уровень контроллера или уровень представления и

достижение более четкого кода. Благодаря инкапсуляции уровня ViewModel: инкапсуляция обработки бизнес-логики, инкапсуляция сетевой обработки, инкапсуляция кэширования данных и т. д. логическая обработка разделена, и нет необходимости обрабатывать данные модели, делая структуру уровня контроллера или уровня слоя простой и понятной [1, 12, 21].

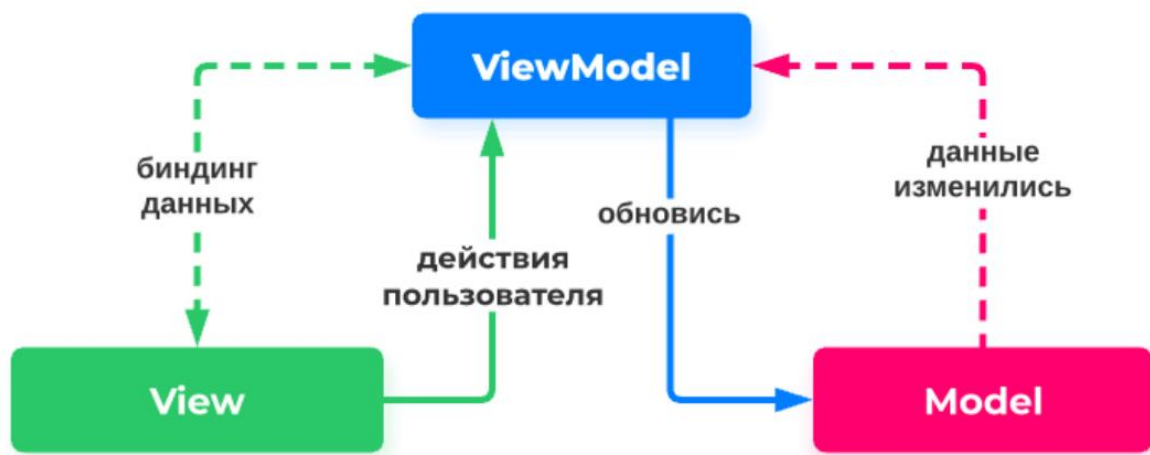


Рисунок 3 – Схема MVVM

Архитектуры MVP и MVVM отлично решают проблемы с тестированием и разделяют Вид и Модель, все компоненты растут сбалансировано. Рисунок 2 и рисунок 3 показывают схемы работы архитектурных паттернов MVP и MVVM соответственно. В отличие от MVC, в MVVM бизнес-логику не всегда легко переиспользовать, а ViewModel может разрастаться. Архитектура MVVM подходит для сложных проектов с средней или большой командой разработчиков [1, 12, 14, 21].

### 1.2.2 Архитектура приложения погода

Архитектурные модули приложения погода можно разделить на три части – Controller, View и Model (Рисунок 4). Controller отвечает за передачу из Model в View и реагирования на все действия View – произведения каких-то

действий с Model. В идеале Controller не должен знать с какой Model он работает. Первое что следует учесть – протоколирование. Когда на экране должен показаться прогноз, Controller от Model нужно получить массив прогнозов. Можно описать, что Controller нужен массив каких-то объектов, которые удовлетворяют протокол. Протокол нужен, потому что при выгрузке данных из различных источников, которые могут предоставлять разные данные. Например, Yahoo Погода показывает диапазон от минимальной погоды в течение дня до максимального, чтобы показать просто погоду, то в протоколе прогнозу указать, что прогноз ожидается, что в объекте будет атрибут свойства температура, для Yahoo Погода температура тип String, т.к. это отобразить на ячейке и для Yahoo Погода рассчитать среднее значение между максимальным и минимальным, привести его к целому значению и в виде String вернуть Controller. Controller не знает какое представление находится в базе, что пришло из сети, он знает что ему нужны такие данные, чтобы их отобразить на View [2, 12, 14].

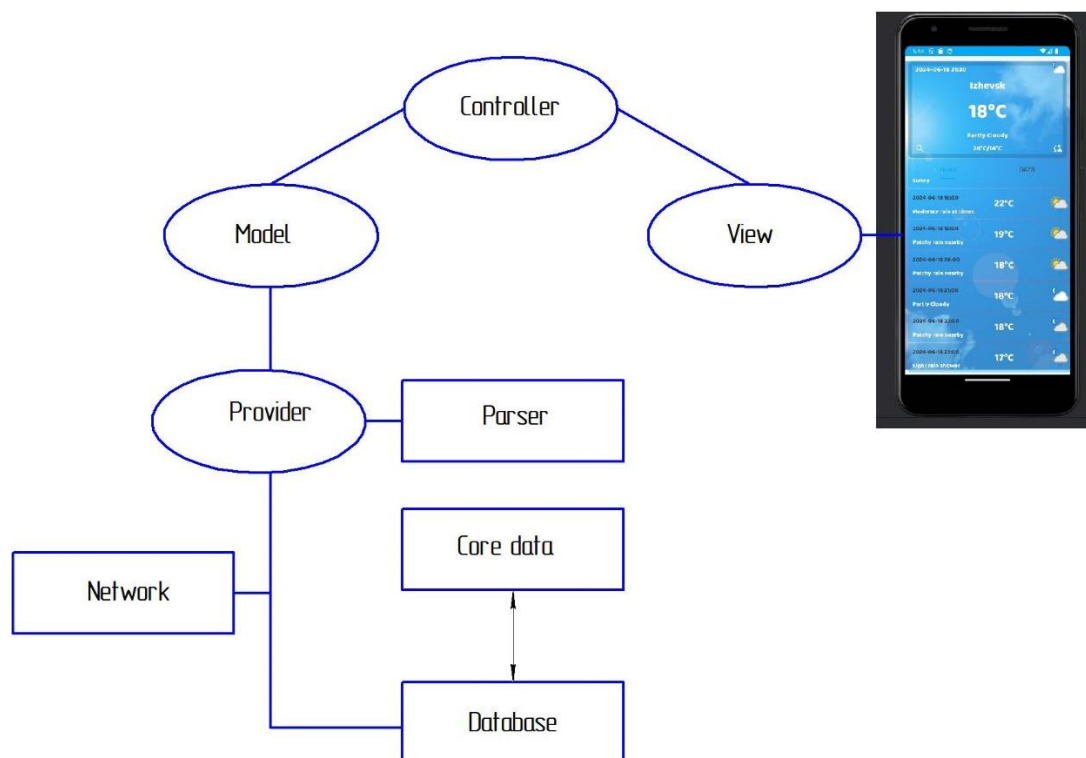


Рисунок 4 – Схема MVC для приложения погода

Модули, как Database база данных не может быть привязана к какой-то модели. Предположим, не требуется хранить какие-то данные, они меняются

с каждым запросом или данных слишком много, или необходимо мигрировать из одного хранилища на другое. Если данные используются из базы в Controller, то проще переписать все приложение, потому что меняя структуру данных придется изменять все то, где они используются, менять Controller, менять Model. При протоколировании (настройка Core data) возвращаются объекты, удовлетворяющие протоколу прогноза. Не важно реальная структура базы, а также, как представлены данные (SQL, XML), важно чтобы этот объект мог вернуть температуру. Слой, отвечающий за сохранение данных [2, 12, 14].

Слой, отвечающий за интернет. Рабочий цикл погодного приложения заключается в следующем: загружается MVC, где Controller содержит какой-то объект, который должен ему вернуть прогнозы погоды для текущего города. Ему нужно вывести на экран значения. Controller может спросить у Model сколько и каких элементов надо отобразить. View через Controller общается с Model. View попросит отобразить скорость ветра Controller обращается к Model – дать модель объекта для этого значения, которое удовлетворяет протоколу с свойством и описанием скорости ветра. Model должна сходить в базу, сделать выборку и вернуть Controller, обратиться к OpenWeather за обновленными данными, получить их, сохранить их в базу и показать обновленные данные в Controller. Model отвечает за несколько слоев: слой сохранения, слой общения с сервисом, а также объект, который каким-то провайдером (Provider) получать информацию от сервиса сохранять и преобразовывать информацию в базу. Нужен объект, который будет провайдером данных для Model, который будет содержать объект, работающий с базой, интернетом и объект, который умеет из интернета создать объект в базе (Parser). Провайдер состоит из нескольких частей, часть которого отвечает за базу, другая часть за интернет, а третья часть отвечает за Parser [2].

Необходим базовый провайдер, который умеет по переданному ему серверу, сервису и месту, сгенерировать объект, который сходит к этому сервису, запросит данные, сохранит их в базу, преобразует в требуемый вид и выведет.

Controller должен минимизировать количество данных, которые он знает о Model, все данные должны фокусироваться в слое Model, View также не должно ничего знать о Model. Если Controller какие-то кусочки еще может знать о Model, например, знать, что модель вернет ему какой-то объект, то View ничего не знает [2, 12, 21].

### 1.3 Выбор подходящего погодного API

Возможность работать с сетью делает приложение клиентом. Возможность отправлять запросы и получать требуемые для работы данные. Большинство современных приложений так или иначе взаимодействуют с сетью.

API (application programming interface) – интерфейс прикладного программирования. Набор способов и правил, которыми различные программы общаются между собой и обмениваются данными (Рисунок 5) [13].

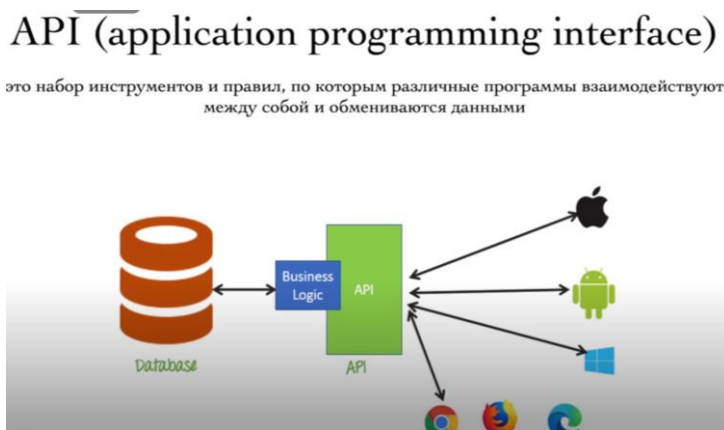


Рисунок 5 – Структура API

В приложении погоды представлена сводка с большим количеством информации, как: осадки, влажность, атмосферное давление все это выгружается благодаря API (Рисунок 6), приложение тянет данные с помощью этого инструмента из всех сервисов, например, портал мониторинга погоды, метеоцентров или лабораторий.

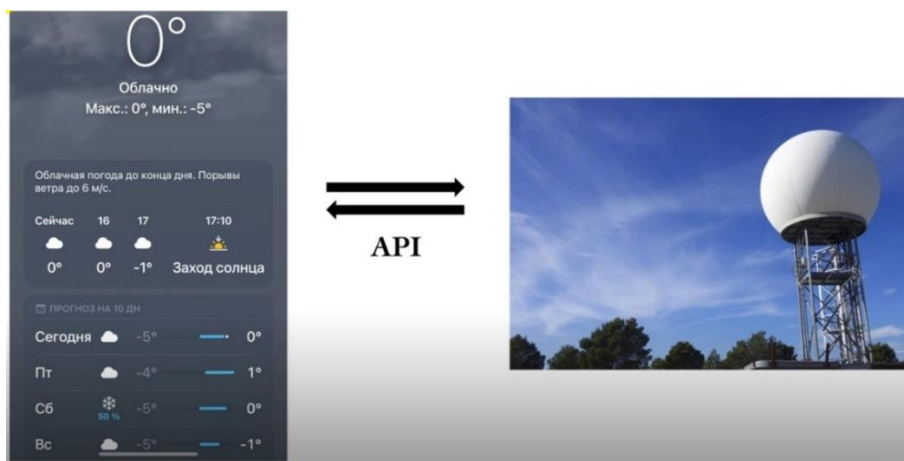


Рисунок 6 – API в сервисах погоды

API – это интерфейс, набор команд доступных для выполнения процессов между двумя системами, граница, на которой происходит взаимодействие и обмен информацией. При этом процессы внутри каждой из систем скрыты друг от друга.

API позволяет разработчикам не создавать сервис с нуля, а подключиться к другому сервису с готовым решением, что позволяет снизить стоимость и временные затраты на разработку приложения [13].

Крупные компании зарабатывают на продаже API, для выполнения этой работы были использованы доступные ресурсы OpenWeather [23].

Одним из основополагающих аспектов создания погодного приложения является поиск и интеграция надежного API (интерфейс прикладного программирования). Выбранный вами API предоставит все необходимые данные для отображения прогноза погоды, текущих условий и другой сопутствующей информации в вашем приложении. При выборе погодного API учитывайте следующие факторы:

1. **Точность и надежность:** API должен предоставлять пользователям точную и актуальную информацию. Проверьте отзывы пользователей и отраслевые рейтинги, чтобы убедиться в качестве API, прежде чем интегрировать его в свое приложение.

2. **Охват данных:** убедитесь, что API охватывает все географические регионы и предлагает поддержку нескольких языков, если ваше приложение поддерживает различные регионы. Это гарантирует, что пользователи из разных регионов смогут получить доступ к точным данным о погоде.

3. **Типы и формат данных:** API должен поддерживать широкий спектр типов данных, включая температуру, влажность, давление, осадки, скорость ветра и другие погодные измерения. Данные также должны быть доступны в легко воспринимаемом формате, например, JSON или XML.

4. **Документация и поддержка:** исчерпывающая документация и отзывчивая служба поддержки жизненно важны для разработчиков при интеграции API. Убедитесь, что поставщик API предлагает надлежащую документацию и своевременную поддержку для решения любых технических проблем или запросов.

5. **Стоимость:** стоимость использования API погоды может варьироваться в зависимости от поставщика и требований к использованию. Выберите API, который предлагает тарифный план, подходящий для вашего проекта, сохраняя при этом высокую производительность и качество.

Некоторые популярные погодные API включают:

- **OpenWeatherMap:** популярный API, предлагающий данные о текущей погоде, прогнозы погоды и исторические данные о погоде. Он имеет бесплатный уровень с ограниченным доступом и платные тарифные планы для широкого использования.

- **Weatherbit:** предоставляет ежедневные прогнозы, почасовые прогнозы и исторические данные о погоде. Weatherbit предлагает щедрый бесплатный тарифный план, а также премиум-планы для более широкого использования.

- **Tomorrow.io (ранее Climacell):** предлагает гиперлокальные прогнозы, поминутные прогнозы, пользовательские погодные оповещения и многое другое. API имеет бесплатный уровень и различные платные уровни для удовлетворения ваших потребностей.



- **API The Weather Channel:** предлагает доступ к тем же данным, которые используются в популярном приложении Weather Channel, включая ежедневные и почасовые прогнозы, предупреждения о неблагоприятной погоде и данные радара.

Исходя из доступности, популярности и информативности API от WeatherAPI.com подходит для данного приложения [9, 11, 16, 22, 23, 23].

## **2 Изучить и подобрать технологии для реализации мобильного приложения**

### **2.1 Выбор языка разработки**

В современном мире Android-разработка стоит на двух языках: Java и Kotlin. Именно эти два языка являются официальными и поддерживаются компанией Google – разработчиком операционной системы Android. Как и основной платформой для программирования для этой ОС – Android Studio.

Каждый проект уникален имеет свои особенности и требования, для эффективности разработки необходимо сравнить и выбрать язык программирования.

Что такое Java? [1, 3, 18, 20]

Java относится к типизированным и объектно-ориентированным языкам программирования. Создан одним из первых – в 1995 году – компанией Sun Microsystems, позднее приобретен одной из крупнейших IT-корпораций Oracle. Язык является универсальным, так как используется для написания разнообразных клиентских и серверных приложений. Но основным направлением применения многие специалисты считают создание ПО для Android. До 2017 года был единственным языком, официально поддерживаемым Google и сервисом компании Android Studio.

Что такое Kotlin?

Kotlin также относится к типизированным и объектно-ориентированным языкам программирования. В этом нет ничего удивительного, так как он работает поверх JVM (Java Virtual Machine), которая является основной частью исполнительной системы Java. Поэтому с некоторой долей условности Kotlin можно назвать надстройкой над Java.

Язык появился в 2011 году, а в 2017 получил официальный статус от компании Google в качестве инструмента для работы на Android Studio. Что стало основанием для длительных разбирательств между Google и Oracle, чьи интересы, как владельца бренда Java, были затронуты таким решением.

## Различия языков Java и Kotlin.

И Kotlin, и Java используются в разработке Android приложений в данный момент.

Чтобы разобраться, чем Kotlin отличается от Java, сравним их по нескольким критериям: возраст языка, количество кода, проверяемые исключения, безопасность, среда разработки и цели разработки.

В таблице 1 представлено сравнение данных языков по разным критериям. Благодаря таблице можно наглядно продемонстрировать различия языков.

Таблица 1 – Различия языков Kotlin и Java

Критерий	Kotlin	Java
Возраст	Примерно 13 лет. Заметно уступает конкуренту по этому параметру	Около 29 лет. Java намного старше, что является одним из ключевых аргументов в его пользу
Количество кода	Краткий и лаконичный. Один из самых простых в написании	Сравнительно большой, заметно более громоздкий, чем у Kotlin
Безопасность	Относительно невысокая	Высокая, предусматривающая встроенную опцию null-безопасности
Среда разработки	Одинаковая для обоих языков разработки ПО – Android Studio	
Проверяемые исключения	В языке не предусмотрены условия для проверяемых исключений,	Проверяемые исключения - головная боль Java-программистов.

	а значит нет необходимости объявлять какие-либо исключения. Код сокращается – безопасность повышается	Компилятор заставляет каждый привлекающий внимание метод обозначать как исключение. Разработчик вынужден обрабатывать его, а это сказывается на скорости разработки
Цели разработки	Разработка нового и сложного программного обеспечения с длительным сроком использования	Преимущественно поддержка старых продуктов, написание простых приложений

[1, 18]

Почему Kotlin следует использовать для разработки приложения погоды?

Kotlin - это современный язык программирования, полностью совместимый с Java и завоевавший популярность среди разработчиков Android. Он предлагает множество преимуществ по сравнению с Java, таких как лаконичный синтаксис, защита от нулевых значений и улучшенная читаемость. Kotlin также имеет отличную поддержку асинхронного программирования, что важно при отправке запросов API и обработке сетевых ответов в приложении погоды. Кроме того, совместимость Kotlin с Java позволяет разработчикам использовать существующие библиотеки и фреймворки Java, что делает его идеальным выбором для создания приложений погоды [16].

## 2.2 Выбор интеграционной среды разработки

Интегрированная среда разработки (Integrated Development Environment (IDE) – это программа, в которой разработчики пишут, проверяют, тестируют

и запускают код, а также ведут большие проекты. Она включает в себя сразу несколько инструментов: редактор для написания кода, сервисы для его проверки и запуска, расширения для решения дополнительных задач разработки. Можно сказать, что это как Photoshop для дизайнера – общее пространство для большинства рабочих процессов [1, 5].

Существуют десятки разных IDE. Их можно делить на группы по разным критериям.

Для разработки Android приложений существует замечательная IDE – Android Studio.

Android Studio – программа, являющаяся средой разработки приложений для мобильной платформы Android. Прямой конкурент самой популярной утилиты для создания софта под Android – Eclipse.

Android Studio превосходит конкурента по многим параметрам, к которым можно отнести [4, 10, 15, 17]:

- гибкость среды разработки;
- большой набор функций;
- процесс разработки, который подстраивается под разработчика.

Во время создания приложений и утилит для операционной системы Android, пользователь программного обеспечения может наблюдать за изменениями в проекте, в режиме реального времени.

#### Особенности Android Studio

В программу встроен эмулятор, позволяющий проверить корректную работу приложения на устройствах с разными экранами, с различными соотношениями сторон.

Отличительная особенность эмулятора – просмотр приблизительных показателей производительности при запуске приложения на самых популярных устройствах.

Локализация приложений становится существенно проще с функцией SDK, которая также входит в перечень достоинств Android Studio.

Достоинства утилиты:

- среда разработки поддерживает работу с несколькими языками программирования, к которым относятся самые популярные – C/C++, Java.
- редактор кода, с которым удобно работать;
- позволяет разрабатывать приложения не только для смартфонов/планшетов, а и для портативных ПК, приставок для телевизоров Android TV, устройств Android Wear, новомодных мобильных устройств с необычным соотношением сторон экрана;
- тестирование корректности работы новых игр, утилит, их производительности на той или иной системе, происходит непосредственно в эмуляторе;
- рефакторинг уже готового кода;
- достаточно большая библиотека с готовыми шаблонами и компонентами для разработки ПО;
- разработка приложения для Android N – самой последней версии операционной системы;
- предварительная проверка уже созданного приложения на предмет ошибок в нем;
- большой набор средств инструментов для тестирования каждого элемента приложения, игры;
- для неопытных/начинающих разработчиков специально создано руководство по использованию Android Studio, размещенное на официальном сайте утилиты.

#### Недостатки/спорные моменты

Несмотря на наличие встроенного Android-эмулятора в самой среде разработки, с тестированием новоразработанного приложения могут возникнуть трудности. Так, для его запуска необходима достаточно внушительная по производительности аппаратная основа ПК, на котором планируется тестирование.

Еще один недостаток - это невозможность написать серверные проекты на языке Java для ПК, Android устройств.

Ниже приведены системные требования для Android Studio в Windows:

- 64-разрядная версия Microsoft® Windows® 8/10/11;
- архитектура процессора x86\_64; Intel Core 2-го поколения или новее, или процессор AMD с поддержкой гипервизора Windows;
- 8 ГБ оперативной памяти или более;
- минимум 8 ГБ свободного места на диске (IDE + Android SDK + эмулятор Android);
- минимальное разрешение экрана 1280 x 800.

Вывод

Android Studio, как программное обеспечение для разработки приложений на Android производит приятное впечатление, как опытным разработчикам, так и новичкам.

Богатый набор инструментов, гибкость в разработке, возможности тестирования, поддержка нескольких языков программирования и встроенный эмулятор делают эту утилиту одной из лучших в своей нише.

Поэтому Android Studio – отличный выбор для разработки данного мобильного приложения [4, 10, 15, 17].

## **2.3 Материально-техническое обеспечение**

Ноутбук:

- 64-разрядная версия Microsoft® Windows® 10;
- процессор: Intel(R) Core(TM) i7-8650U CPU @ 1,90 GHz 2,11 GHz;
- оперативная память: 16 ГБ;
- жесткий диск: 1 ТБ.

Java Development Kit: 22.

Integrated Development Environment: Android Studio Iguana | 2023.2.1 Patch 1, Build #AI-232.10300.40.2321.11567975.

Язык программирования: Kotlin.

Смартфон:

- модель: Samsung Galaxy S22 8+256GB;
- операционная система: Android 14;
- модель процессора: Exynos 2200;
- количество ядер: 8;
- объем оперативной памяти: 8 ГБ;
- объем встроенной памяти: 256 ГБ;
- диагональ экрана: 6.1";
- разрешение экрана: 400x1080 Пикс.

## **2.4 Последовательность разработки проекта**

1. Создание проекта и подготовка.
2. В Манифест обеспечить доступ в интернет – android.permission.INTERNET.
3. Разработка дизайна погодного приложения:
  - интерфейс;
  - типографика;
  - доступ к данным в реальном времени.
4. Получить ключ API.
5. Получать данные из OpenWeatherMap.
6. Android Studio: Сохранить ключ и город в настройках.
7. Получение данных о погоде в Android Studio:
  - выполнение запросов к API;
  - анализ ответов в формате JSON;
  - обработка ошибок;
8. Отображение информации о погоде.
9. Запуск. Отладка.
10. Запуск на смартфоне.



## 11. Выгрузка в Google Play.

В следующей главе будет последовательное описание реализации этих этапов в Android Studio.

## 3 Реализация мобильного Android-приложения “Прогноз погоды”

### 3.1 Подготовительный этап

#### 3.1.1 Создать новый проект

Запустить Android Studio, во вкладке File раскрыть ветвь New – New Project (Рисунок 7).

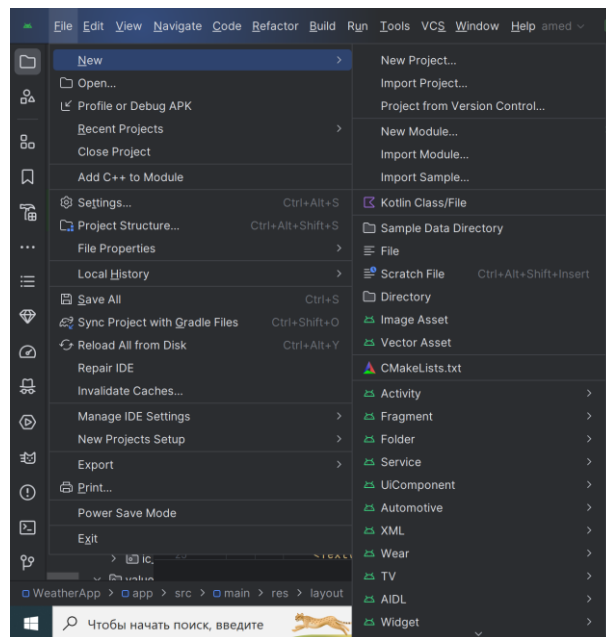


Рисунок 7 – Создание нового проекта

Выбрать шаблон Phone and Tablet, Empty Views Activity и нажать Next (Рисунок 8).

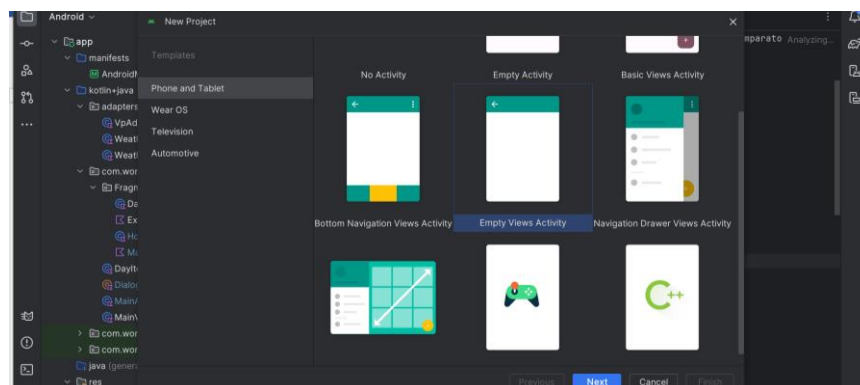


Рисунок 8 – Выбор шаблона

Выбрать и написать название приложению WeatherForecast. Имя пакету. Выбрать каталог (папку), где будет сохранен проект. Выбрать язык программирования Kotlin. Выбрать SDK (комплект разработки), который поддерживает максимальное количество Android версий. Выбрать языковые конфигурации сборщика. Нажать на Finish (Рисунок 9) [22, 24, 25].

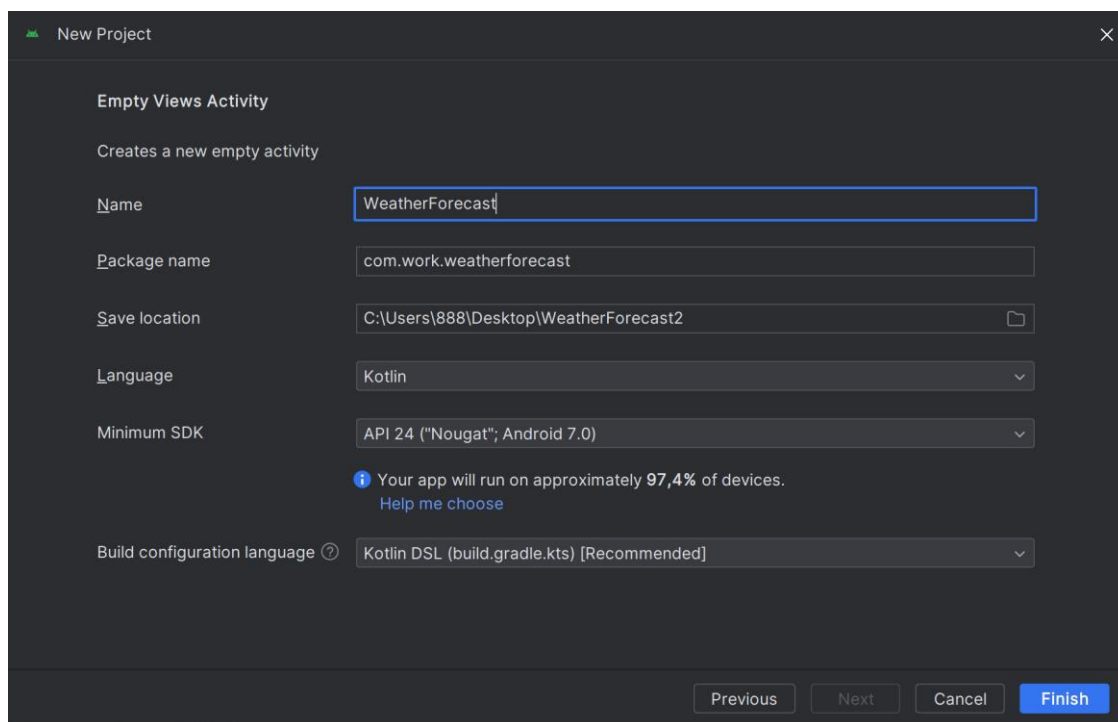


Рисунок 9 – Настройки Empty Views Activity

### 3.1.2 Подключение библиотек и настройки

В приложении используется одно Activity и несколько фрагментов. Поэтому следует подключить View binding (Рисунок 10) и создать переход в основной фрагмент MainFragment (Рисунок 11).

Перейти во вкладку build.gradle (Module) в модуле android прописать:

```
buildFeatures{  
    viewBinding = true  
}
```

Синхронизировать проект.

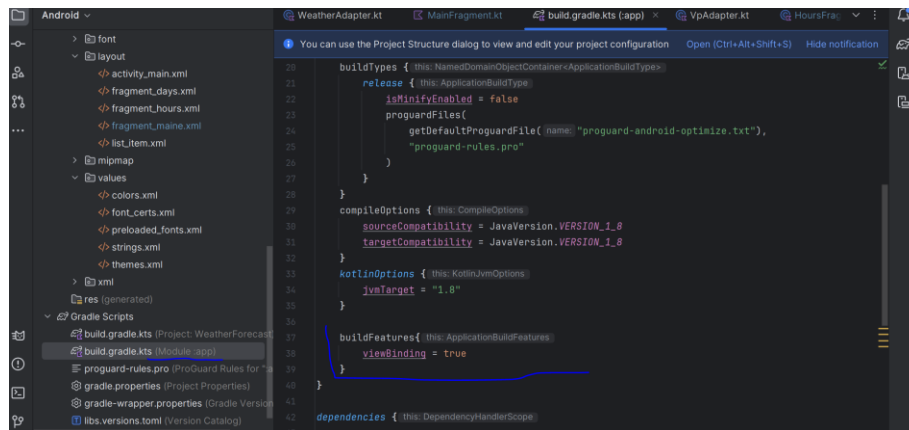


Рисунок 10 – Подключение View binding

На рисунке 11 представлен пустой фрагмент и фрагмент с пустой разметкой, который постепенно будет заполнен. Как только откроется MainActivity – сразу открывается MainFragment, где прописывается код, чтобы обновлять, выгружать с API сервера погодные данные, рассортировывать по другим фрагментам, которые показывают прогноз погоды по часам [22, 24, 25].

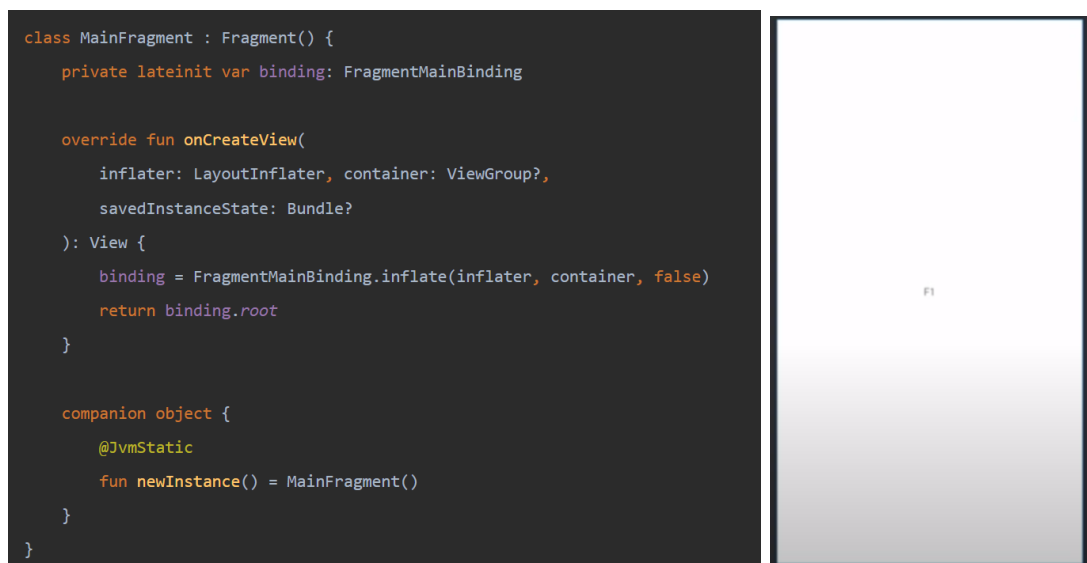


Рисунок 11 – Переход в основной фрагмент MainFragment

В MainActivity (Рисунок 12) будем запускать supportFragmentManager, чтобы открыть фрагмент.

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        supportFragmentManager
            .beginTransaction()
            .replace(R.id.placeholder, MainFragment.newInstance())
            .commit()
    }
}

```

Рисунок 12 – Запуск фрагмента

Чтобы приложение работало, происходило соединение с интернетом, а также происходила выгрузка через библиотеки, необходимо прописать разрешения в AndroidManifest и активировать зависимости в build.gradle.

Разрешение на передачу и прием данных из интернета, и учет местоположения:

```

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

android:usesCleartextTraffic="true"

```

Применяется android:usesCleartextTraffic="true", чтобы не было проблем, когда обращаемся через протокол http к данной web-странице.

Подгрузка библиотек, учитывающих работу с фрагментами, выгрузку иконок, геолокация в модуле build.gradle (Module):

```

implementation("com.android.volley:volley:1.2.1")
implementation("com.squareup.picasso:picasso:2.71828")
implementation("androidx.fragment:fragment-ktx:1.7.1")
implementation("com.google.android.gms:play-services-location:21.2.0")

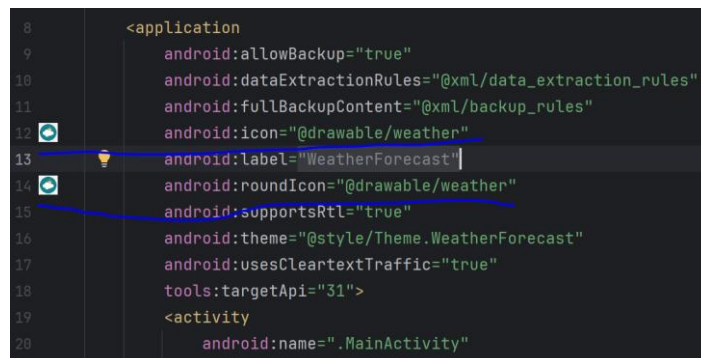
```

Синхронизировать проект.

### 3.1.3 Добавить логотип (иконку) приложения

В качестве иконки приложения, нажимая которое на экране смартфона пользователь будет заходить приложение погода, была выбрана картинка weather.png (Рисунок 7). Картинка загружена в папку drawable и теперь

файле `AndroidManifest.xml` обозначить: `android:icon="@drawable/weather"` – становится иконкой приложения, `android:label="@string/app_name"` – имя приложения под иконкой “WeatherForecast”, `android:roundIcon="@drawable/weather"` – скругление иконки (Рисунок 13 а). На рисунке 13 б представлено отображение иконки на экране смартфона [22, 24, 25].

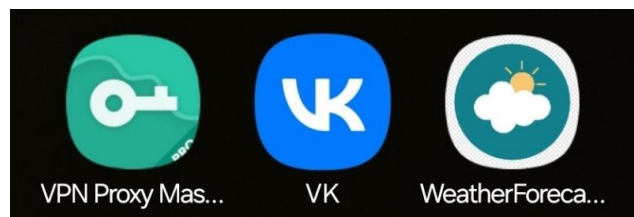


```

8      <application
9          android:allowBackup="true"
10         android:dataExtractionRules="@xml/data_extraction_rules"
11         android:fullBackupContent="@xml/backup_rules"
12         android:icon="@drawable/weather"
13         android:label="WeatherForecast"
14         android:roundIcon="@drawable/weather"
15         android:supportsRtl="true"
16         android:theme="@style/Theme.WeatherForecast"
17         android:usesCleartextTraffic="true"
18         tools:targetApi="31">
19         <activity
20             android:name=".MainActivity"

```

а



б

Рисунок 13 – Создание иконки приложения погода, где:  
код отображения (а), отображение на экране смартфона (б)

### 3.2 ViewModel

Класс `MainViewModel` он наследуется от `ViewModel`. Класс отвечает за обновление данных, полученных с сервера.

```

class MainViewModel: ViewModel() { //Данные
    val liveDataCurrent = MutableLiveData<WeatherModel>() //Погода сего-
дня/Сейчас
    val liveDataList = MutableLiveData<List<WeatherModel>>() //Список погоды
на ближайшие дни.
}

```

### 3.3 DataClass

`DataClass` – это группа переменных, в нем созданы все необходимые переменные.

```

data class DayItem(
    val city: String,
    val time: String,
    val condition: String,
    val imageUrl: String,
    val currentTemp: String,
    val maxTemp: String,
    val minTemp: String,
    val hours: String
)

```

### 3.4 Проверка разрешения

Приложению необходимо спрашивать разрешение, например, на использование доступа к местоположению пользователя. Чтобы пользователь дал разрешение можно использовать или нет, иначе приложение не получит доступ.

#### Extencion function:

```

fun Fragment.isPermissionGranted(p: String): Boolean {
    return ContextCompat.checkSelfPermission(
        activity as AppCompatActivity, p) == PackageManager.PERMISSION_GRANTED
}

```

В MainFragment выполняется проверка, при запуске основного фрагмента появляется вся нужная информация о прогнозе погоды и функция:

```

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    checkPermission()
    init()
    updateCurrentCurd()
}

```

Производится проверка, запускаемая private lateinit var pLauncher: ActivityResultLauncher<String>

```

private fun permissionListener(){
    pLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()){
        Toast.makeText(activity, "Permission is $it",
            Toast.LENGTH_LONG).show()
    }
}

```

Что запускает пользователю диалог – дает ли пользователь разрешение о доступе к местоположения.

Функция проверяет есть ли разрешение или нет:

```
private fun checkPermission() {  
    if (!isPermissionGranted(Manifest.permission.ACCESS_FINE_LOCATION)) {  
        permissionListener()  
        pLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION)  
    }  
}
```

## 3.5 Создание основного экрана

### 3.5.1 Разметка основного экрана

В fragment\_main.xml создать карточку, фон, отображения элементов.

Фоном будет изображение sky.jpg, которое было скачено из интернета, помещена в папку drawable и через ImageView выведена на экран (Рисунок 14).

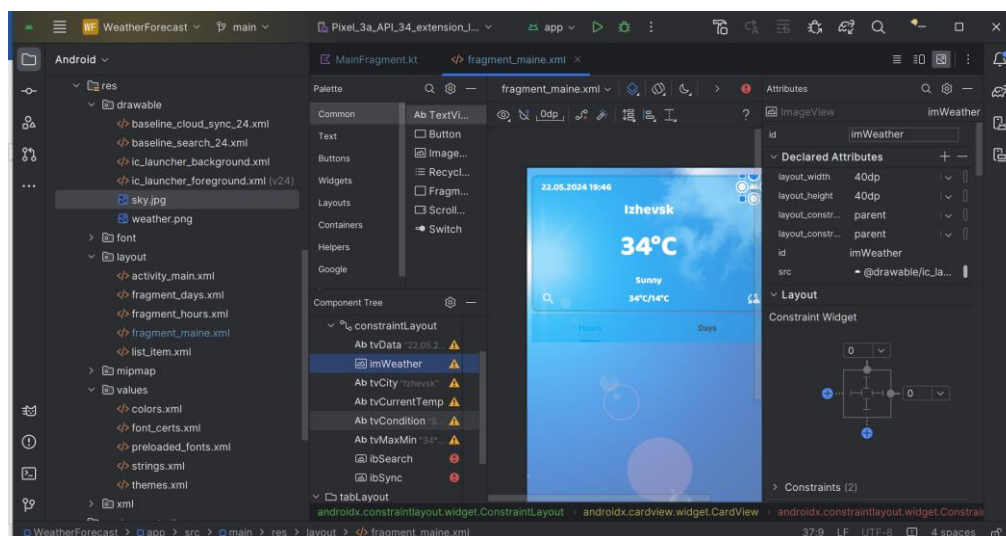


Рисунок 14 – Создание фона главного экрана

Через `scalType fitXY` зафиксировать, чтобы `ImageView` адаптировала картинку под экран. Атрибутом `alfa 0,5` сделать картинку тусклой.

Создание `CardView`, в которой будут расположены элементы, показывающие текущую погоду, время, температуру, осадки. Последовательным заполнением элементов (несколько текстов, `ImageView`, два `ImageButton`, пять `TextView`) (Рисунок 15) и описанием их свойств, значений, выравниванием, созданием закруглений, цветовой гаммы идентификаторов элементов



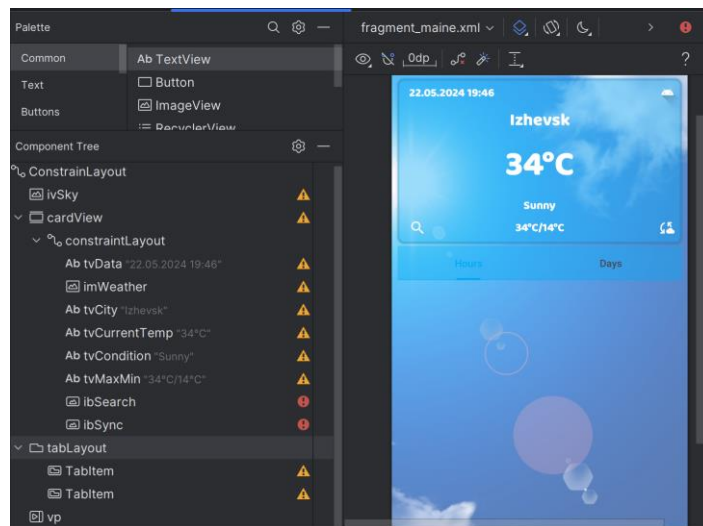


Рисунок 15 – Дерево построения разметки главного экрана

Стандартные иконки для ImageButton можно импортировать из папки drawable: drawable – New – Vector Asset, где найти нужную иконку (Рисунок 16), отключить background и добавить отступы [22, 24, 25].

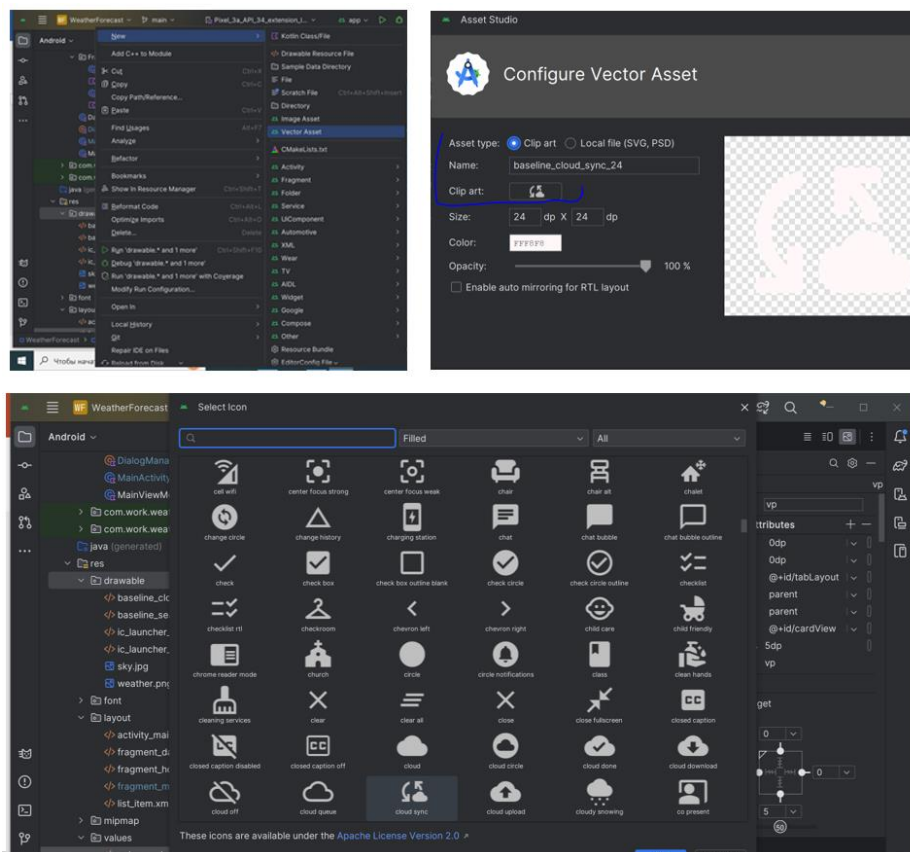


Рисунок 16 – Импорт стандартных иконок

### 3.5.2 Создание HoursFragment и DaysFragment

Необходимо разделить и активировать списки, показывающие погоду по часам и дням. Разделить на два разных фрагмента для часов и дней, настроить переключение между ними. В каждом фрагменте брать соответствующую информацию.

Создать HoursFragment удалить лишнее и перейти в fragment\_hours.xml.

Создать DaysFragment удалить лишнее и перейти в fragment\_days.xml.

Эти фрагменты будут отвечать за свою разметку, когда будет переключаться viewPager2.

Для переключения между фрагментами создан специальный адаптер (Рисунок 17) [22, 24, 25].

```
1 package adapters
2
3 import androidx.fragment.app.Fragment
4 import androidx.fragment.app.FragmentActivity
5 import androidx.viewpager2.adapter.FragmentStateAdapter
6
7 class VpAdapter(fa: FragmentActivity, private val list: List<Fragment>) : FragmentStateAdapter(fa) {
8     override fun getItemCount(): Int {
9         return list.size
10    }
11
12    override fun createFragment(position: Int): Fragment {
13        return list[position]
14    }
15 }
```

Рисунок 17 - viewPagerAdapter

Теперь к viewPager2 нужно подключить созданный адаптер. В MainFragment.kt создать функцию, которая будет инициализировать. Создан адаптер, передан FragmentActivity, который передан через конструктор в родительский класс. Передать список fList с фрагментами, между которыми надо переключаться. Указать порядо очередности переключения:

```
private fun init() = with(binding) {
    fLocationClient = LocationServices.getFusedLocationProviderClient(requireContext())
    val adapter = VpAdapter(activity as FragmentActivity, fList)
    vp.adapter = adapter
    TabLayoutMediator(tabLayout, vp) {
        tab, pos -> tab.text = tList[pos]
    }
```

```

    }.attach()
    ibSync.setOnClickListener {
        tabLayout.selectTab(tabLayout.getTabAt(0))
        checkLocation()
    }
    ibSearch.setOnClickListener {
        DialogManager.searchByNameDialog(requireContext(), object: DialogManager.Listener {
            override fun onClick(name: String?) {
                if (name != null) {
                    requestWeatherData(name)
                }
            }
        })
    }
}
}
}

```

Чтобы запустить эту функцию и иметь возможность переключаться:

```

private fun init() = with(binding) {
    val adapter = VpAdapter(activity as FragmentActivity, fList)
    vp.adapter = adapter
    TabLayoutMediator(tabLayout, vp) {
        tab, pos -> tab.text = tList[pos]
    }.attach()
}

```

Связать TabLayout с viewPager2, чтобы переключались фрагменты и создавалась анимация перелистывания элементов.

```

TabLayoutMediator(tabLayout, vp) {
    tab, pos -> tab.text = tList[pos]
}.attach()

```

Создать список, который при переключении показывает разные названия.

```

private val tList = listOf(
    "Hours",

```

```
"Days"
```

```
)
```

### 3.6 Создание data class WeatherModel

Data class (Рисунок 18), в котором будут храниться переменные (группа переменных) и передаваться в адаптер. Переменные: город, время, текущее состояние погоды, текущая температура, максимальная и минимальная температура [22, 24, 25].

```
1 package adapters
2
3 @Sergey
4 data class WeatherModel(
5     val city: String,
6     val time: String,
7     val condition: String,
8     val currentTemp: String,
9     val maxTemp: String,
10    val minTemp: String,
11    val imageUrl: String,
12    val hours: String
13 )
```

Рисунок 18 – Data class

### 3.7 Создание разметки

Создать файл `list_item.xml`. Разметка как будет выглядеть один элемент из списка (Рисунок 19). Установить элемент `CardView` внутри него `ConstraintLayout`, в котором прикрепить все что нужно. `TextView`, который показывает погоду по часам. `TextView Condition`, который показывает состояние погоды – солнечно/пасмурно. Вставить `TextView` для указания температуры. Произвести выравнивание, отступы, закругления. Присвоить индексы. Задать фон `CardView` командой `backgroundTint="@color/card_blue"` [22, 24, 25].

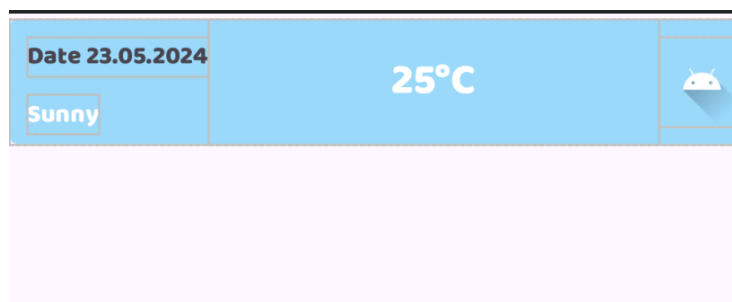


Рисунок 19 – Разметка элемента из списка

### 3.8 Создание RecyclerViewAdapter

Чтобы показать список нужен контейнер, который будет показывать элементы. Был создан шаблон – разметка для одного элемента, её будут копировать – клонировать и показывать в списке, заполняя разными данными.

Для этого добавить RecyclerView во фрагмент, в котором необходимо показывать данный список (HoursFragment и DaysFragment).

Во фрагменте `fragment_hours.xml` прописать `<androidx.constraintlayout.widget.ConstraintLayout`.

Если по каким-то причинам придет пустой список, то вывести `android:text="Empty"`. В дизайне прикрепить TextView. Дать идентификатор.

Добавить и прикрепить RecyclerView, а также дать ему идентификатор.

Получен контейнер, где будет отображаться список.

А как это работает? Много элементов участвует в этом процессе. Следует понимать, например, у человека есть стопка пригласительных открыток. Некий шаблон с полями где нужно заполнить ФИО. Таких открыток много и все они одинаковые. Все поля пустые. Есть список с людьми, у которых есть фамилия, имя и отчество. Человек является адаптером, стол, где будут складывать заполненные открытки – RecyclerView, т.е. контейнер, куда добавляют элементы. Человек смотрит на список, где есть первое имя, на первой строчке - заполняет эту открытку и кладет на стол. Берет следующую, смотрит, заполняет, кладёт, таким образом – человек является RecyclerViewAdapter, тот элемент, который собирает все вместе [22, 24, 25].

Создать класс WeatherAdapter (Рисунок 20) и связать его с ListAdapter, передать данные с WeatherModel.

```
import android.view.View
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.meter_alc_rgb.weatherappcursey.databinding.ListItemBinding

class WeatherAdapter : ListAdapter<WeatherModel, WeatherAdapter.Holder>() {

    class Holder(view: View) : RecyclerView.ViewHolder(view){
        val binding = ListItemBinding.bind(view)

        fun bind(item: WeatherModel) = with(binding){
            tvDate.text = item.time
            tvCondition.text = item.condition
            tvTemp.text = item.currentTemp
        }
    }
}
```

Рисунок 20 – Создать класс WeatherAdapter

Класс ViewHolder хранит ссылки на View, шаблон (Рисунок 19) состоит из View, когда создаются копии – список из ста элементов – это все копии одного элемента и это разные элементы, которые сохранены в разных местах памяти, поэтому у каждого элемента своя ссылка. И это хранится и заполняется в ViewHolder. RecyclerView выводит и перерисовывает элементы.

В class WeatherAdapter : ListAdapter<WeatherModel, WeatherAdapter.Holder>(Comparator()) необходимо создать и передать Comparator. Когда не было ListAdapter использовался RecyclerViewAdapter, где приходилось прописывать самостоятельно, когда заменяем старый список на новый, чтобы все оптимально происходило – приходилось прописывать, нужно проверить какие элементы будут одинаковы, а с помощью ListAdapter и Compara-

тор все будет происходить автоматически. Comparator (Рисунок 21) будет проходить по списку и сравнивать элементы, если они одинаковые, то его трогать не будет, если разные, то заменит на новый [22, 24, 25].

```
class Comparator : DiffUtil.ItemCallback<WeatherModel>(){
    override fun areItemsTheSame(oldItem: WeatherModel, newItem: WeatherModel): Boolean
    {
        return oldItem == newItem
    }

    override fun areContentsTheSame(oldItem: WeatherModel, newItem: WeatherModel):
    Boolean {
        return oldItem == newItem
    }

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
    val view = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent,
false)
    return Holder(view)
}

override fun onBindViewHolder(holder: Holder, position: Int) {
    holder.bind(getItem(position))
}
}
```

Рисунок 21 – Создание класса Comparator

Выгрузить разметку:

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
Holder {
    val view = LayoutInflater.from(parent.context).inflate(R.layout.
list_item, parent, false)
    return Holder(view)
}
```

Когда только создается и отправляется в память Holder(view) – сразу запускается onBindViewHolder. Регламентирует, как заполнять. Выдает позицию, на которой создался Holder.

```

override fun onBindViewHolder(holder: Holder, position: Int) {
    holder.bind(getItem(position))
}

```

Дополним класс HoursFragment и создадим временный список, чтобы вывести пробный список (Рисунок 22).

```

class HoursFragment : Fragment() {
    private lateinit var binding: FragmentHoursBinding
    private lateinit var adapter: WeatherAdapter

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentHoursBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initRcView()
    }

    private fun initRcView() = with(binding) {
        rcView.layoutManager = LinearLayoutManager(activity)
        adapter = WeatherAdapter()
    }
}

```



```

rcView.adapter = adapter

val list = listOf(
    WeatherModel(
        "", "12:00",
        "Sunny", "25°C",
        "", "", "", ""),
    WeatherModel(
        "", "13:00",
        "Sunny", "25°C",
        "", "", "", ""),
    WeatherModel(
        "", "14:00",
        "Sunny", "35°C",
        "", "", "", ""))

adapter.submitList(list)
}

companion object {
    @JvmStatic
    fun newInstance() = HoursFragment()
}
}

```



Рисунок 22 – HoursFragment вывод пробного списка

## 3. 9 Получение данных через API

### 3.9.1 Функция requestWeatherData

Создать функцию `private fun requestWeatherData(city: String) {`, в которую передать название города. Будет выдавать прогноз погоды в данном городе.

### 3.9.2. Создать url для запроса

В электронном ресурсе `weather api` во вкладке `Interactive API Explorer` формируем ссылку для запроса (Рисунок 23). Данными для запроса является ключ пользователя, формат протокола ссылки `https`, формат вывода `JSON`, ключ города. Во вкладке `Forecast` указать требуемое количество дней, например, 3 [25, 26].

Parameter	Value	Type	Location	Description
q	Izhevsk	string	query	Pass US Zipcode, UK Postcode, Canada Postalcode, IP address, Latitude/Longitude (decimal degree) or city name. Visit <a href="#">request parameter</a> section to learn more.

Parameter	Value	Type	Location	Description
days	3	integer	query	Number of days of weather forecast. Value ranges from 1 to 10
aqi	no	string	query	Get air quality data
alerts	no	string	query	Get weather alert data

Рисунок 23 – Создаем url для запроса

Получим ссылку (<https://api.weatherapi.com/v1/forecast.json?key=30de988cefc545bfa33162049242005&q=Izhevsk&days=3&aqi=no&alerts=no>) и Response Body (Рисунок 24) код с выводом всех интересующих параметров погоды в .

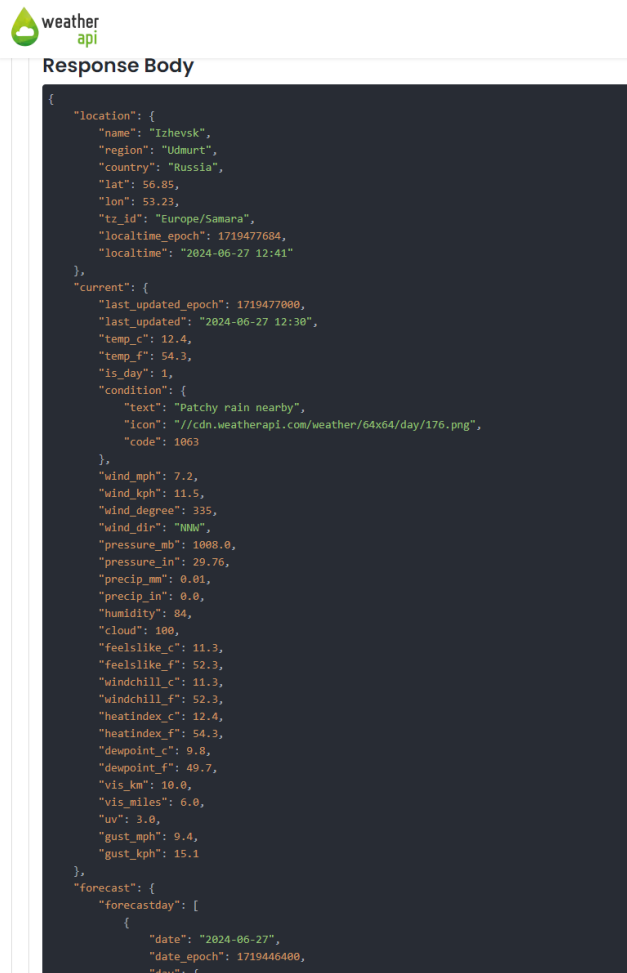


Рисунок 24 - Response Body код в формате JSON с интересующими параметрами погоды

### 3.9.3 Volley.newRequestQueue(context)

Создать специальный класс (очередь) queue:

```
val queue = Volley.newRequestQueue(context)
```

Создали – инициализировали очередь и теперь в эту очередь надо передавать запросы queue.add(request).

### 3.9.4 Создать запросы StringRequest()

Указать метод с помощью которого получать данные Request.Method.GET. Передать ссылку url, сформировать ожидание ответа с выводом результата result -> parseWeatherData(result), которым будет JSON формате.

Слушатель ошибок error -> Log.d( "MyLog", "Error: \$error") [22, 24, 25].

### 3.10 Парсинг данных полученных с API

#### 3.10.1 Парсинг данных полученных с API

Данные с API в JSON формате, как температура, погодные условия – заполнить в карточку, в список по часам и дня.

Создать функцию `private fun parseWeatherData(fromServer: String)`.

Указываем `val mainObject = JSONObject(result)`, что позволяет читать JSON формат.

Рисунке 25 представлены данные в формате JSON (а) название города, картинка с погодными условиями в данный момент, погодные условия описание, минимальная и максимальная температура, время, когда произвели замеры будут выведены на экран (б) [22, 24, 25].

```
{
  "location": {
    "name": "Izhevsk",
    "region": "Udmurt",
    "country": "Russia",
    "lat": 56.85,
    "lon": 53.23,
    "tz_id": "Europe/Samara",
    "localtime_epoch": 1719486659,
    "localtime": "2024-06-27 15:10"
  },
  "current": {
    "last_updated_epoch": 1719486000,
    "last_updated": "2024-06-27 15:00",
    "temp_c": 14.6,
    "temp_f": 58.2,
    "is_day": 1,
    "condition": {
      "text": "Overcast",
      "icon": "///cdn.weatherapi.com/weather/64x64/day/122.png",
      "code": 1009
    }
  }
}
```

а



б

Рисунке 25 – Данные в формате JSON (а), они же выведены на экран после обработки (б)

Последовательно подготовим класс с выводом этих значений.

```
private fun parseCurrentData(mainObject: JSONObject, weatherItem:
WeatherModel) {

    val item = WeatherModel(

        mainObject.getJSONObject("location").getString("name"),

        mainObject.getJSONObject("current").getString("last_up-
dated"),

        mainObject.getJSONObject("current")
```

```

        .getJSONObject("condition").getString("text"),
        mainObject.getJSONObject("current").getString("temp_c"),
        weatherItem.maxTemp,
        weatherItem.minTemp,
        mainObject.getJSONObject("current")
            .getJSONObject("condition").getString("icon"),
        weatherItem.hours
    )

    Log.d("MyLog", "City: ${item.maxTemp}")
    Log.d("MyLog", "Time: ${item.minTemp}")
    Log.d("MyLog", "Time: ${item.hours}")
}

```

### 3.10. 2 Прасинг массива, полученного с API

В прошлом разделе были выгружены текущие данные после последнего обновления. Когда на сервере обновились данные, отправили запрос и получили сведения о погоде – температура, которая есть сейчас.

Есть еще прогноз погоды, где нужно выгрузить список для каждого дня, какая будет погода на завтра – через час, два или три. Нужно получить этот список и взять из него первый элемент из списка – это будет сегодняшний день, достать минимальную и максимальную температуру – это и будет готовый прогноз погоды.

Объект `forecast` (Рисунок 26) этот объект содержит в себе всю информацию о прогнозе погоды именно в списке – массив с информацией и каждый объект содержит массив прогноз погоды по часам.

```

    },
    "forecast": {
      "forecastday": [
        {
          "date": "2024-06-27",
          "date_epoch": 1719446400,
          "day": {
            "maxtemp_c": 16.7,
            "maxtemp_f": 62.1,
            "mintemp_c": 9.5,
            "mintemp_f": 49.2,
            "avgtemp_c": 12.8,
            "avgtemp_f": 55.1,
            "maxwind_mph": 18.1,
            "maxwind_kph": 16.2,
            "totalprecip_mm": 0.17,
            "totalprecip_in": 0.01,
            "totalsnow_cm": 0.0,
            "avgvis_km": 10.0,
            "avgvis_miles": 6.0,
            "avghumidity": 80,
            "daily_will_it_rain": 1,
            "daily_chance_of_rain": 75,
            "daily_will_it_snow": 0,
            "daily_chance_of_snow": 0,
            "condition": {
              "text": "Patchy rain nearby",
              "icon": "https://cdn.weatherapi.com/weather/64x64/day/176.png",
              "code": 1063
            },
            "uv": 3.0
          },
          "astro": {
            "sunrise": "03:36 AM",
            "sunset": "09:24 PM",
            "moonrise": "No moonrise",
            "moonset": "10:16 AM",
            "moon_phase": "Waning Gibbous",
            "moon_illumination": 71,
            "is_moon_up": 1,
            "is_sun_up": 1
          },
          "hour": [
            {
              "time_epoch": 1719412800,
              "hour": {
                "time_epoch": 1719518400,
                "time": "2024-06-28 00:00",
                "temp_c": 10.0,
                "temp_f": 50.0,
                "is_day": 0,
                "condition": {
                  "text": "Overcast ",
                  "icon": "https://cdn.weatherapi.com/weather/64x64/night/122.png",
                  "code": 1009
                }
              }
            }
          ]
        }
      ]
    }
  }
}

```

Рисунок 26 – Массив данных в формате JSON, прогноз погоды по часам для этого дня

Необходимо выгрузить массив, в котором хранится информация по дням.

Создадим функцию, которая выводит forecastday, функция будет выдавать лист – список объектов из WeatherModel. Список пустой, заполняется прохождением цикла. Достать массив в виде JSONObject - val daysArray = mainObject.getJSONObject("forecast")

.getJSONArray("forecastday"), каждый объект день и нужно достать информацию [22, 24, 25]:

```

private fun parseDays(mainObject: JSONObject): List<WeatherModel>{
    val list = ArrayList<WeatherModel>()

    val daysArray = mainObject.getJSONObject("forecast")

    .getJSONArray("forecastday")

```

```

        val name = mainObject.getJSONObject("location").getString("name")

        for (i in 0 until daysArray.length()) {
            val day = daysArray[i] as JSONObject
            val item = WeatherModel(
                name,
                day.getString("date"),
                day.getJSONObject("day").getJSONObject("condition")
                    .getString("text"),
                "",
                day.getJSONObject("day").getString("maxtemp_c"),
                day.getJSONObject("day").getString("mintemp_c"),
                day.getJSONObject("day").getJSONObject("condition")
                    .getString("icon"),
                day.getJSONArray("hour").toString()
            )
            list.add(item)
        }

        return list
    }
}

```

Запустить данную функцию через `parseCurrentData(mainObject, list[0])`:

```

private fun parseWeatherData(result: String) {
    val mainObject = JSONObject(result)
    val list = parseDays(mainObject)
    parseCurrentData(mainObject, list[0])
}

```

### 3.11 Показ прогноза на экране

#### 3.11.1 Перенос кода в новую функцию `parseCurrentData()`

В этом разделе продолжим заполнение погодной карточки (Рисунок 27). В прошлых разделах получили вывод основной информации, теперь необходимо обновлять все на экране. Вывод картинки состояния погоды, температуры.



Рисунок 27 – Погодная карточка

В `MainFragment` инициализировать `MainViewModel` класс, т.к. через него будем передавать `WeatherModel`:

```
val liveDataCurrent = MutableLiveData<WeatherModel>()
```

В функции `private fun parseCurrentData(mainObject: JSONObject, weatherItem: WeatherModel) {`

```
    val item = WeatherModel( - item нужно передавать в MainView-  
    Model, передавать и заполнять.
```

`liveDataCurrent` сюда будет передавать специальным `observe`, который будет следить, когда все `View` доступны, тогда и будут обновлять. Следит за циклом жизни фрагмента, позволяет избежать ошибок.

Активация `MainViewModel` происходит: `private val model: MainViewModel by activityViewModels()` – инициализировал класс, который будет обновлять.

Создадим `observe`, который будет выдавать полученный класс в `WeatherModel` [22, 24, 25]:

```
private fun updateCurrentCurd() = with(binding) {  
    model.liveDataCurrent.observe(viewLifecycleOwner) {  
        val maxMinTemp = "${it.maxTemp}°C/${it.minTemp}°C"  
        tvData.text = it.time  
        tvCity.text = it.city  
        tvCurrentTemp.text = it.currentTemp.ifEmpty { "${it.maxTemp}°C /  
        ${it.minTemp}°C"  
        tvCondition.text = it.condition
```



```

        tvMaxMin.text = if(it.currentTemp.isEmpty()) "" else maxMinTemp
        Picasso.get().load("https:" + it.imageUrl).into(imWeather)
    }
}

```

Перейдем во фрагмент `fragment_main.xml` (Рисунок 28). На основном экране есть различные `TextView`: дата обновления, температура в данный момент, максимальная и минимальная температура данного дня, название локации, картинка — все это нужно заполнять.

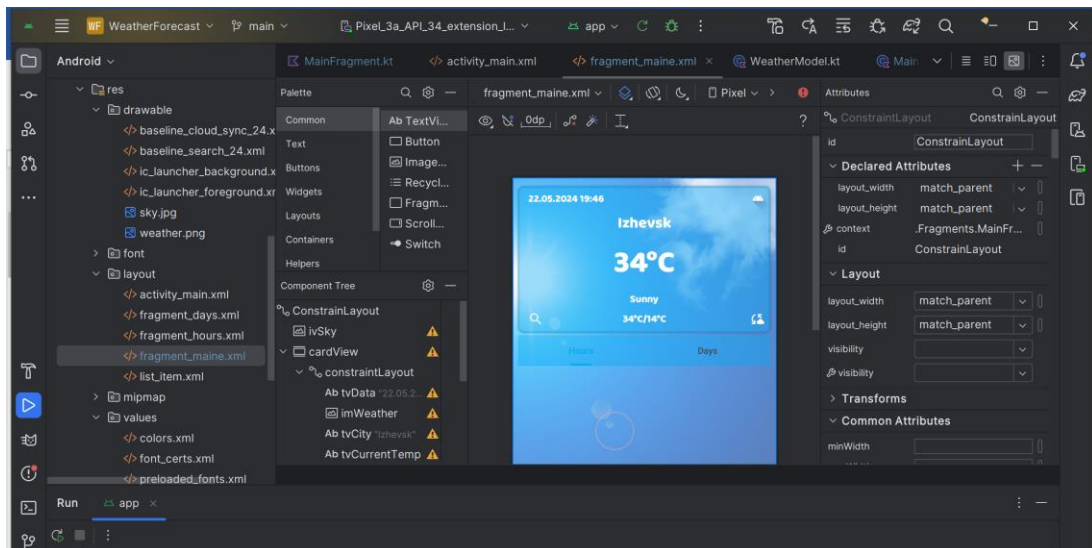


Рисунок 28 – Основной экран приложения фрагмент `fragment_main.xml`

### 3.11.2 Функция `parseCurrentData()` для получения списка

```

private fun parseCurrentData(mainObject: JSONObject, weatherItem: Weather-
Model) {
    val item = WeatherModel(
        mainObject.getJSONObject("location").getString("name"),
        mainObject.getJSONObject("current").getString("last_updated"),
        mainObject.getJSONObject("current")
            .getJSONObject("condition").getString("text"),
        mainObject.getJSONObject("current")
            .getString("temp_c").toFloat().toInt().toString()+"°C",
        weatherItem.maxTemp,
        weatherItem.minTemp,
        mainObject.getJSONObject("current")
            .getJSONObject("condition").getString("icon"),
        weatherItem.hours
    )
    model.liveDataCurrent.value = item
}

```

## 3.12 Показ прогноза погоды по часам

### 3.12.1 Передача WeatherModel на HoursFragment

На текущий момент вывод погоды по часам выводит информацию, которую мы завели ранее. Сами создали WeatherModel с заполненным списком, чтобы проверить RecyclerViewAdapter.

Необходимо доставать реальную информацию, заполнять WeatherModel и создавать списки, тогда появится прогноз погоды по часам.

В HoursFragment нужно передать WeatherModel используя в MainViewModel.kt liveDataCurrent.

В HoursFragment добавить private val model: MainViewModel by activityViewModels().

И функцией:

```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {
    super.onCreateView(view, savedInstanceState)
    initRecyclerView()
    model.liveDataCurrent.observe(viewLifecycleOwner){
        adapter.submitList(getHoursList(it))
    }
}
```

### 3.12.2 Функция getHoursList

Функция getHoursList формирует список WeatherModel и выгружает список по часа, полученный список преобразовать в JSON формат val hoursArray = JSONArray(wItem.hours), а затем преобразовать и вывести строкой (hoursArray[i] as JSONObject).getString("time"), пройти по количеству объектов в массиве [22, 24, 25].

```
private fun getHoursList(wItem: WeatherModel): List<WeatherModel> {
    val hoursArray = JSONArray(wItem.hours)
    val list = ArrayList<WeatherModel>()
    for(i in 0 until hoursArray.length()) {
        val item = WeatherModel(
            wItem.city,
            (hoursArray[i] as JSONObject).getString("time"),
            (hoursArray[i] as JSONObject).getJSONObject("condition").getString("text"),
            (hoursArray[i] as JSONObject).getInt("temp_c").toString()+"°C",
            "",
            "",
            (hoursArray[i] as JSONObject).getJSONObject("condition").getString("icon"),
            ""
        )
        list.add(item)
    }
}
```

### 3.13 Список прогноза по дням

В этом разделе создадим список прогноза по дням. Принцип схож, как и по часам, но здесь нужно выводить максимальную и минимальную температуру.

В классе MainViewModel передаем liveDataCurrent и в liveDataList = MutableLiveData<List<WeatherModel>>() передать список, который получаем в MainFragment из списка функции parseDays, где первый элемент сегодняшний день, из этого списка нужно извлекать остальные элементы и перемещать в DaysFragment [22, 24, 25].

```
class DaysFragment : Fragment() {  
    private lateinit var adapter: WeatherAdapter  
    private lateinit var binding: FragmentDaysBinding  
    private val model: MainViewModel by activityViewModels()  
  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        binding = FragmentDaysBinding.inflate(inflater, container,  
false)  
        return binding.root  
    }  
  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
        init()  
        model.liveDataList.observe(viewLifecycleOwner) {  
            adapter.submitList(it.subList(1, it.size))  
        }  
    }  
}
```

```

private fun init() = with(binding){

    adapter = WeatherAdapter()

    rcView.layoutManager = LinearLayoutManager(activity)

    rcView.adapter = adapter

}

companion object {

    @JvmStatic

    fun newInstance() = DaysFragment()

}
}

```

В WeatherAdapter выполнить дополнение для вывода максимальной и минимальной температуры во вкладке по дням передать из Item максимальную и минимальную температуру:

```

fun bind(item: WeatherModel) = with(binding){
    itemTemp = item
    tvDate.text = item.time
    tvCondition.text = item.condition
    tvTemp.text = item.currentTemp.ifEmpty { "${item.maxTemp}°C /
    ${item.minTemp}°C" }
    Picasso.get().load("https:" + item.imageUrl).into(im)

}
}

```

### 3.14 Обработка нажатий на список дней

Активировать прослушивание нажатий на элементов из списка Days.

В WeatherAdapter добавить интерфейс [22, 24, 25]:

```

interface Listener{

    fun onClick(item: WeatherModel)

}

```

Активировать в DaysFragment:

```
class DaysFragment : Fragment(), WeatherAdapter.Listener {
    private lateinit var adapter: WeatherAdapter
    private lateinit var binding: FragmentDaysBinding
    private val model: MainViewModel by activityViewModels()
```

Передать этот слушатель:

```
private fun init() = with(binding){
    adapter = WeatherAdapter(this@DaysFragment)
    recyclerView.layoutManager = LinearLayoutManager(activity)
    recyclerView.adapter = adapter
}
```

В HoursFragment listener добавить null, т.к. здесь ничего прослушивать не надо.

```
private fun initRecyclerView() = with(binding){
    recyclerView.layoutManager = LinearLayoutManager(activity)
    adapter = WeatherAdapter(null)
    recyclerView.adapter = adapter
}
```

В WeatherAdapter в класс Holder передать этот listener.

```
class Holder(view: View, val listener: Listener?) : RecyclerView.ViewHolder(view){
    val binding = ListItemBinding.bind(view)
    var itemTemp: WeatherModel? = null
    init{
        itemView.setOnClickListener {
            itemTemp?.let { it1 -> listener?.onClick(it1) }
        }
    }
}
```

Если itemTemp = 0, то функция не запустится и ничего не произойдет.

В DaysFragment передаем слушателя.

При нажатии на любой элемент из списка Days получим WeatherModel нажатого элемента (Рисунок 29). Выводит его:

```
override fun onClick(item: WeatherModel) {
    model.liveDataCurrent.value = item
}
```

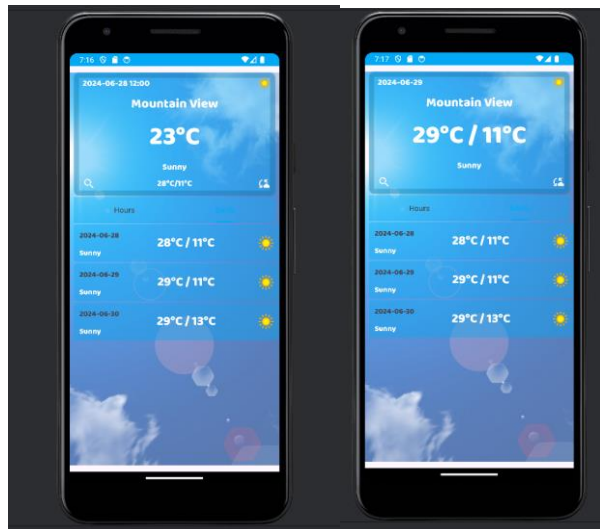


Рисунок 29 – Выведение погоды из списка дня

Однако, в этом случае нет температуры в данный момент, требуется максимальная и минимальная, которую нужно вывести.

В MainFragment прописав функцию с такой зависимостью – выведем значения максимальной и минимальной температуры на экран:

```
private fun updateCurrentCard() = with(binding) {
    model.liveDataCurrent.observe(viewLifecycleOwner) {
        val maxMinTemp = "${it.maxTemp}°C / ${it.minTemp}°C"
        tvData.text = it.time
        tvCity.text = it.city
        tvCurrentTemp.text = it.currentTemp.ifEmpty { maxMinTemp }
        tvCondition.text = it.condition
        tvMaxMin.text = if(it.currentTemp.isEmpty()) "" else
maxMinTemp
        Picasso.get().load("https:" + it.imageUrl).into(imWeather)
    }
}
```

### 3.15 Получение прогноза погоды по местоположению

Чтобы получить доступ к локации потребуется функция по получению местоположения и `fLocationClient`, с помощью которого будем получать местоположение [22, 24, 25].

```
private lateinit var fLocationClient: FusedLocationProviderClient
```

Его нужно инициализировать:

```
private fun init() = with(binding){  
    fLocationClient = LocationServices.getFusedLocationProvider-  
Client(requireContext())
```

Теперь нужна функция, с помощью которой будем получать сведения о местоположении, указать требуемую точность, `CancellationToken`, подтвердить проверку, что пользователь дал разрешение на использование местоположения:

```
private fun getLocation(){  
    val ct = CancellationTokenSource()  
    if (ActivityCompat.checkSelfPermission(  
        requireContext(),  
        Manifest.permission.ACCESS_FINE_LOCATION  
    ) != PackageManager.PERMISSION_GRANTED && Activity-  
Compat.checkSelfPermission(  
        requireContext(),  
        Manifest.permission.ACCESS_COARSE_LOCATION  
    ) != PackageManager.PERMISSION_GRANTED  
    ) {  
        return  
    }  
    fLocationClient  
        .getCurrentLocation(LocationRequest.PRIORITY_HIGH_ACCU-  
RACY, ct.token)
```

```

        .addOnCompleteListener{
            requestWeatherData("${it.result.latitude},${it.re-
sult.longitude}")
        }
    }
}

```

Вывести полученную локацию:

```

fLocationClient
    .getCurrentLocation(LocationRequest.PRIORITY_HIGH_ACCU-
RACY, ct.token)
    .addOnCompleteListener{
        requestWeatherData("${it.result.latitude},${it.re-
sult.longitude}")
    }
}

```

Теперь при входе в приложение будут считывать координаты пользователя:

```

    override fun onViewCreated(view: View, savedInstanceState: Bun-
dle?) {
        super.onViewCreated(view, savedInstanceState)
        checkPermission()
        init()
        updateCurrentCard()
        getLocation()
    }

    private fun init() = with(binding){
        fLocationClient = LocationServices.getFusedLocationProvider-
Client(requireContext())

        val adapter = VpAdapter(activity as FragmentActivity, fList)
    }
}

```



```

vp.adapter = adapter

TabLayoutMediator(tabLayout, vp) {
    tab, pos -> tab.text = tList[pos]
}.attach()

ibSync.setOnClickListener {
    tabLayout.selectTab(tabLayout.getTabAt(0))

    getLocation()
}
}

```

### 3.16 Проверка включения GPS

#### 3.16.1 Функция sLocationEnabled()

Создать проверку – включен ли у пользователя GPS и включить его.  
Для этого создать отдельную функцию [24, 25]:

```

private fun isLocationEnabled(): Boolean{
    val lm = activity?.getSystemService(Context.LOCATION_SERVICE)
as LocationManager

    return lm.isProviderEnabled(LocationManager.GPS_PROVIDER)
}

```

В функции private fun getLocation выполнить проверку и выдаст предупреждение (Рисунок 30):

```

private fun getLocation(){
    if (!isLocationEnabled()){
        Toast.makeText(requireContext(), text="Location disabled!", Toast.LENGTH_SHORT).show()
        return
    }
    val ct = CancellationTokenSource()
    if (ActivityCompat.checkSelfPermission(
        requireContext(),
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
        requireContext(),
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
}

```

Рисунок 30 – Функция isLocationEnabled()

### 3.16.2 AlertDialog

Хорошим решением является создание диалога, который подскажет пользователю, что нужно включить GPS и направит в настройки.

Создать класс DialogManager, в котором будет функция [25]:

```
import android.app.AlertDialog
import android.content.Context

object DialogManager {

    fun locationSettingsDialog(context: Context, listener: Listener){
        val builder = AlertDialog.Builder(context)
        val dialog = builder.create()
        dialog.setTitle("Enable location?")
        dialog.setMessage("Location disabled, do you want enable location?")

        dialog.setButton(AlertDialog.BUTTON_POSITIVE, "OK") { _, _ ->
            listener.onClick()
            dialog.dismiss()
        }

        dialog.setButton(AlertDialog.BUTTON_NEGATIVE, "Cancel") { _, _ ->
            dialog.dismiss()
        }

        dialog.show()
    }

    interface Listener{
        fun onClick()
    }
}
```

### 3.16.3 Функция checkLocation()

В MainFragment создать функцию checkLocation():

```
private fun checkLocation() {  
    if (isLocationEnabled()) {  
        getLocation()  
    } else {  
        DialogManager.locationSettingsDialog(requireContext(), ob-  
ject : DialogManager.Listener {  
            override fun onClick() {  
                startActivity(Intent(Settings.ACTION_LOCA-  
TION_SOURCE_SETTINGS))  
            }  
        })  
    }  
}
```

### 3.16.4 Использование функций checkLocation() и onResume()

Возникают проблемы с появлением диалога и автоматического обновления данных при включении GPS, код больше не включается – он запускается только один раз, когда создаются View, мы возвращаемся и запускается функция onResume(), возвращаемся – все это берется из памяти, поэтому мы не видим никакого обновления, нужно перезайти в приложение, тогда все появится.

Нам нужно запускать не только при нажатии кнопки checkLocation(), но и onResume(), когда мы возвращаемся, то запускается функция из жизни фрагмента [25, 25]:

```
override fun onResume() {  
    super.onResume()  
    checkLocation()  
}
```

Тогда мы возвращаемся с экрана, когда пользователь включил, делаем checkLocation, и если пользователь включил, то происходит проверка и выдает getLocation(), если не включил, то запускается диалог.

### 3.17 AlertDialog выбор города

В этом разделе активизируем кнопку search, выведем возможность диалога, в котором будет поле поиска города, когда он будет введен – при нажатии Ок, нам сразу же покажет прогноз погоды в городе, который мы ввели (Рисунок 31) [25].

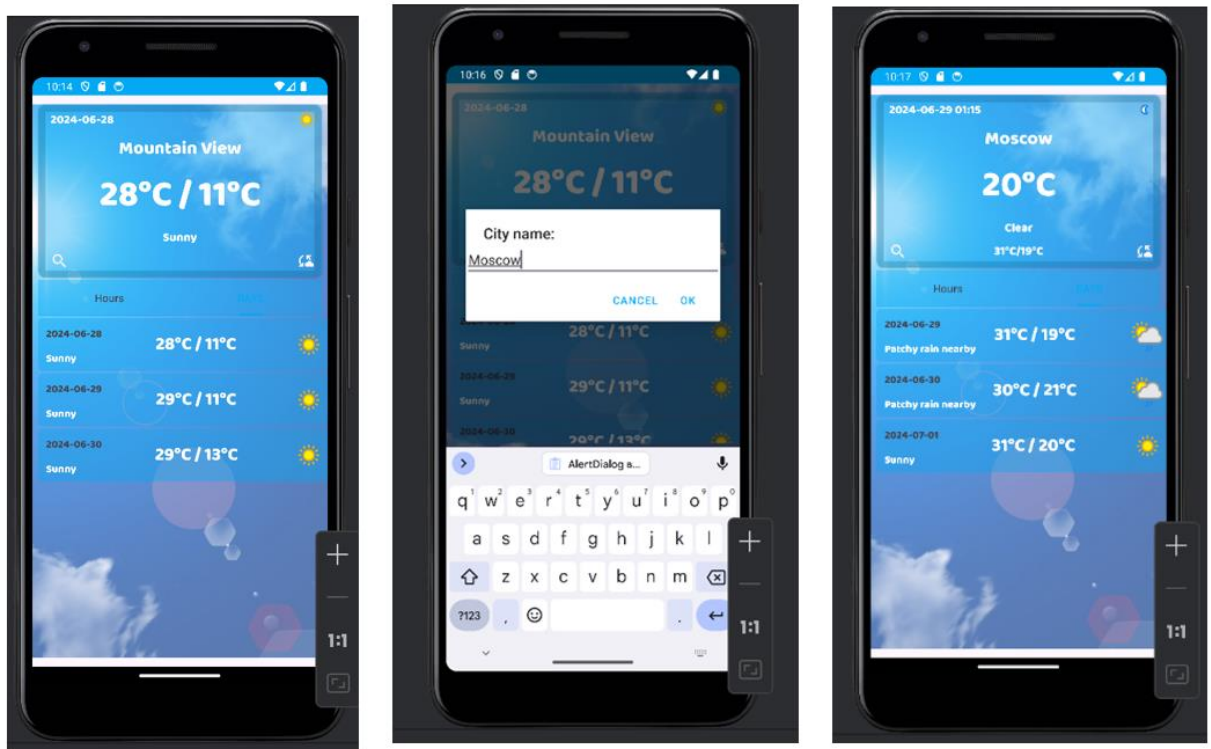


Рисунок 31 – Работа поискового запроса погоды по городам

Перейти в DialogManager, создать функцию (Рисунок 32):

```
fun searchByNameDialog(context: Context, listener: Listener)
```

Задать заголовок: "City name:", возможность набирать и редактировать текст и его активировать:

```
val edName = EditText(context)  
builder.setView(edName)
```

В слушателе выполнить возврат:

```
interface Listener{  
    fun onClick(name: String?)  
}
```

```

fun searchByNameDialog(context: Context, listener: Listener){
    val builder = AlertDialog.Builder(context)
    val edName = EditText(context)
    builder.setView(edName)
    val dialog = builder.create()
    dialog.setTitle("City name:")
    dialog.setButton(AlertDialog.BUTTON_POSITIVE, text: "Ok"){_, _ ->
        listener.onClick(edName.text.toString())
        dialog.dismiss()
    }
    dialog.setButton(AlertDialog.BUTTON_NEGATIVE, text: "Cancel") { _, _ ->
        dialog.dismiss()
    }
    dialog.show()
}

interface Listener{
    fun onClick(name: String?)
}

```

Рисунок 32 – Функция поискового запроса погоды по городам

Изменился интерфейс – он теперь передает name: String? в MainFragment дополнить этот фрагмент.

```

private fun checkLocation() {
    if(isLocationEnabled()) {
        getLocation()
    } else {
        DialogManager.locationSettingsDialog(requireContext(), object : DialogManager.Listener{
            override fun onClick(name: String?) {
                startActivity(Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS))
            }
        })
    }
}

```

Во втором случае отследить нажатие кнопки search, поэтому создать нового слушателя, что нажатие этой кнопки запускает диалог. Добавить интерфейс. Метод [25]:

```
ibSearch.setOnClickListener {
    DialogManager.searchByNameDialog(requireContext(), object: Dialog-
Manager.Listener{
        override fun onClick(name: String?) {
            if (name != null) {
                requestWeatherData(name)
            }
        }
    })
}
```

Таким образом приложение показывает прогноз погоды не только по местоположению пользователя, а также по поисковому запросу.

### 3.18 Запуск приложения

Проект готов, чтобы его запустить следует выполнить сборку проекта (Рисунок 33 позиция 1) , после выполнения сборки запустить (Рисунок 33 позиция 2) [22, 24, 25].

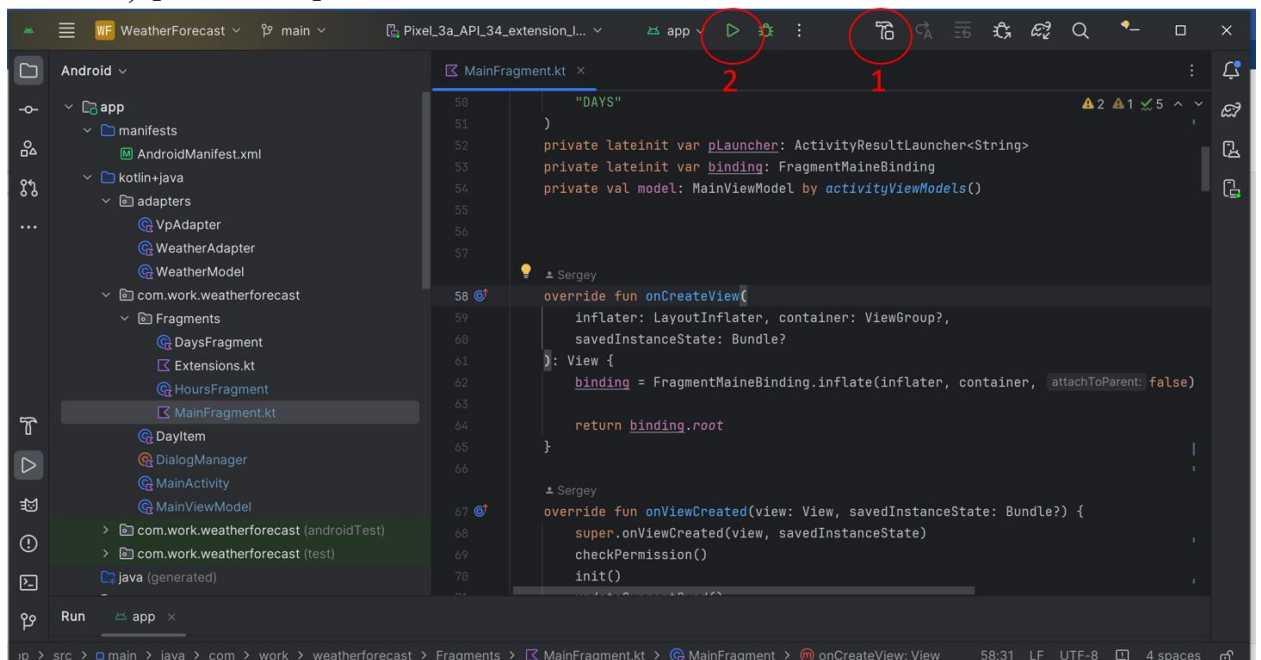


Рисунок 33 – Сборка и запуск приложения

На рисунке 34 представлено запущенное погодное приложение. На экране показано: время последнего обновления, локация, состояние, текущая температура, максимальная и минимальная температура. Выгружен и представлен список погоды по часам, рядом с ним соседняя вкладка - список погоды по дням. Кнопка search – для ввода локации, где пользователю интересно узнать погоду.

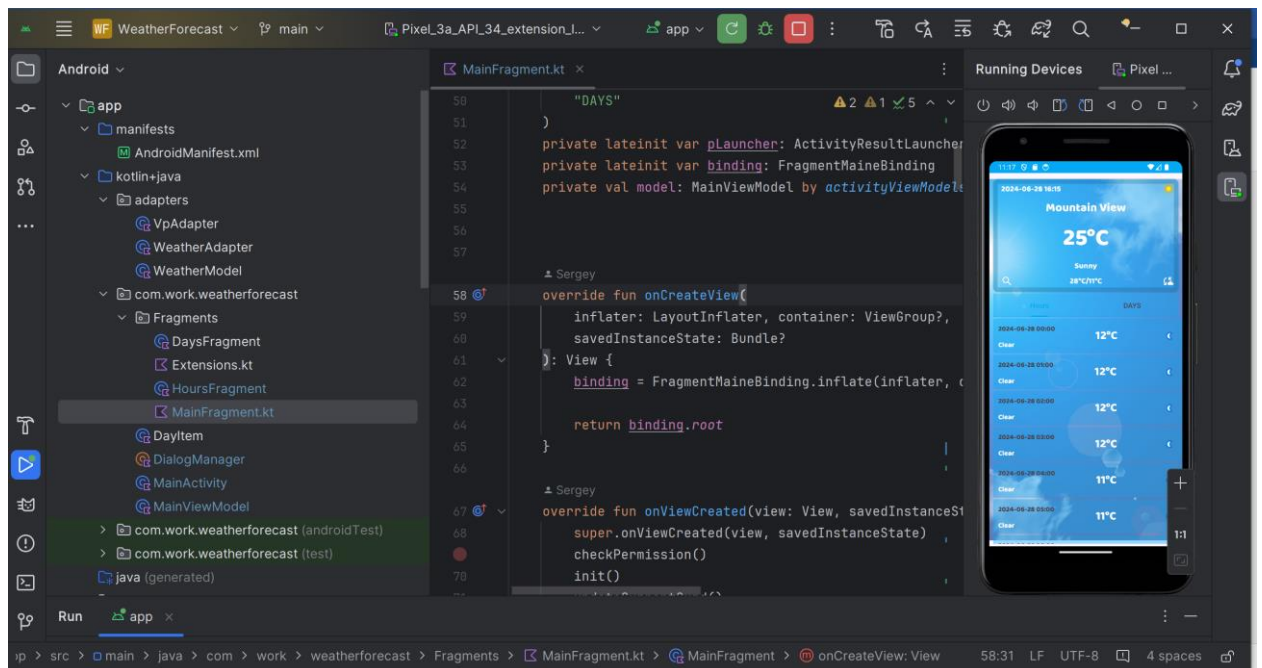


Рисунок 34 – Запущенное приложение

## 4 Тестирование и подготовка к релизу

### 4.1 Подключение смартфона к Android Studio

#### 4.1.1 Настройка параметров Android Studio

Приложение заработало на виртуальном устройстве, пришло время протестировать его на смартфоне.

Следует убедиться, что требуемые драйвера Android Studio активированы для этого перейти во вкладке Tools (Рисунок 35) зайти SDK Manager – Android SDK – SDK Tools. На рисунке 36 представлены параметры, которые должны быть активны, чтобы была корректная работа со смартфоном.

Убедиться в соответствии настроек, активировать Google USB Driver [7].

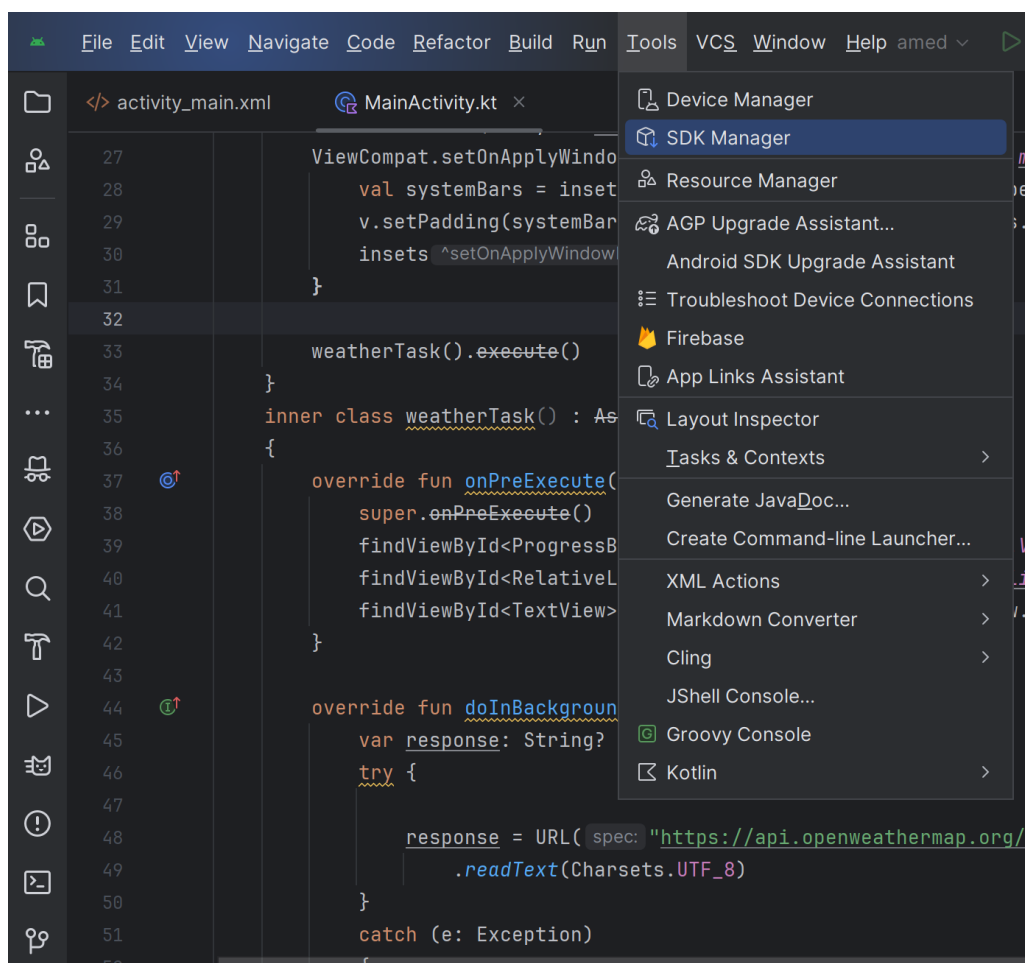


Рисунок 35 – Вкладка Tools



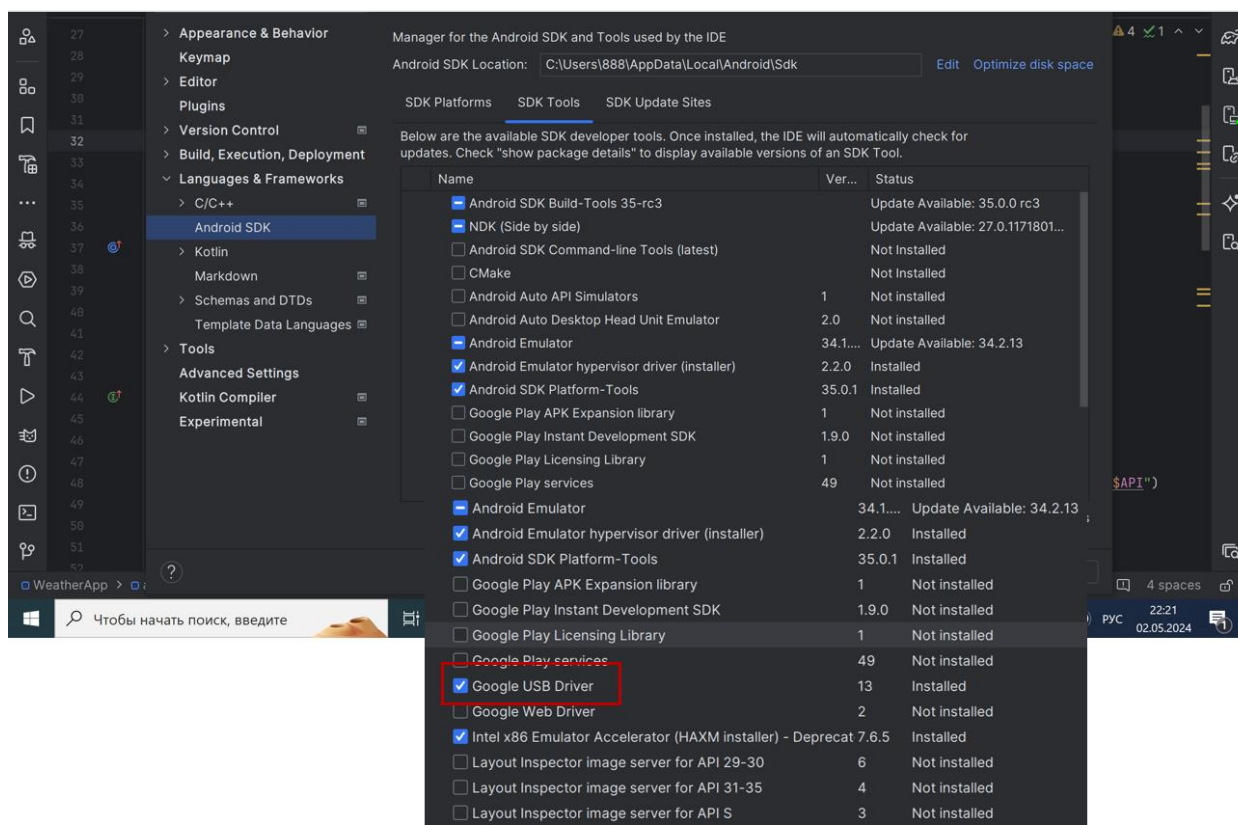


Рисунок 36 – Требуемые драйвера

#### 4.1.2 Настройки параметров смартфона

В настройках смартфона (Рисунок 37): Настройки – Сведения о телефоне (1) – Сведения о ПО – найти “Номер сборки” (2) кликнуть по нему 7 раз, после чего откроются параметры разработчика (3), где разрешить отладку по USB (4-5).

Подключить смартфон к компьютеру.

Перейти в Android Studio выбрать подключение девайса.

Следовать диалоговому окну, после чего смартфон отобразится на верхней панели и можно запускать.

Рисунок 38 смартфон подключен к Android Studio и отображает разработанное приложение [7].

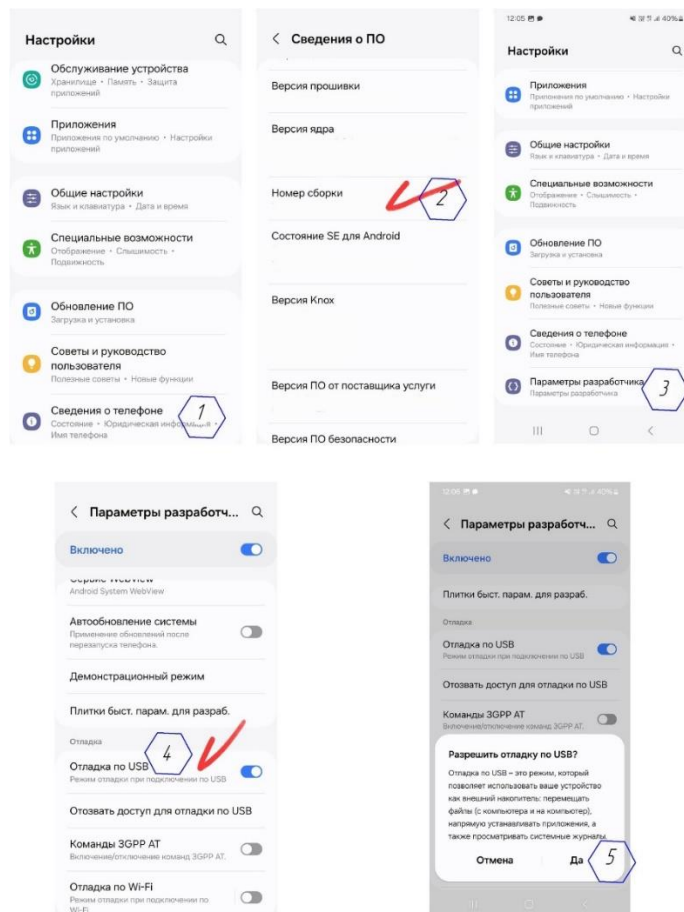


Рисунок 37 – Настройки подключения смартфона по USB к Android Studio

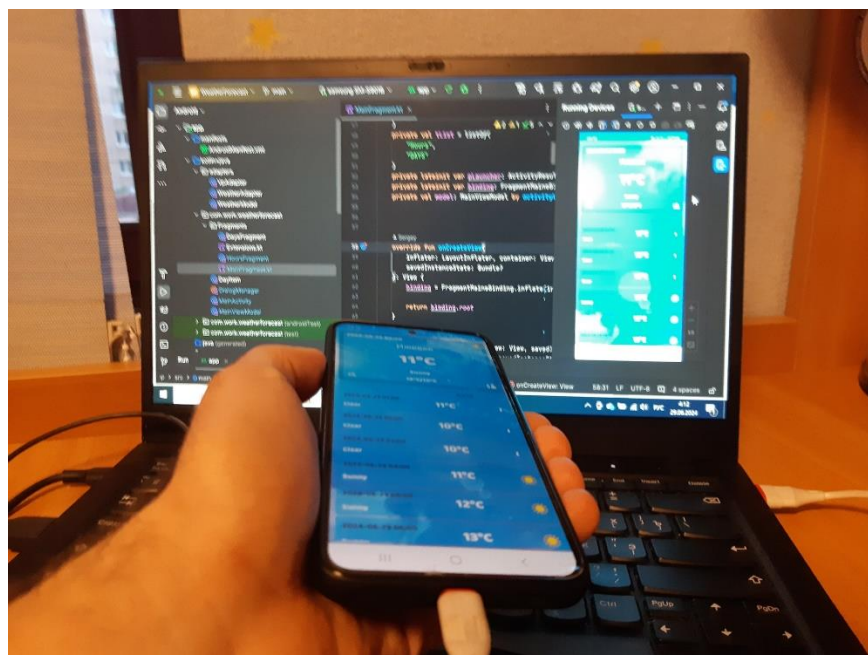


Рисунок 38 – Смартфон подключен приложение работает

## 4.2 Сохранить установочный файл приложения – арк файл

Чтобы сохранить установочный файл, как арк файл и взаимодействовать им: в Android Studio перейти во вкладку Build – Build Bundle(s) / APK(s) и выбрать Build APK(s) (Рисунок 39) [8].

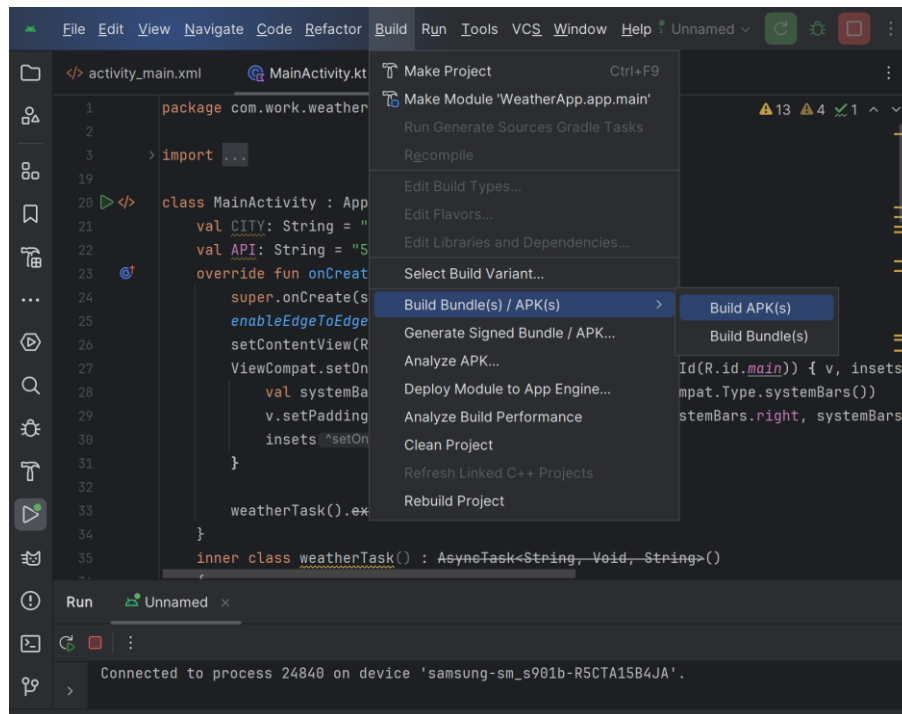


Рисунок 39 – Формирование установочного файла

После формирования установочного файла система выдаст сообщение, что он готов, кликнув по locate будет открыт каталог с файлом (Рисунок 40). Это файл можно загрузить и запустить на смартфоне или отправить через мессенджер [8].

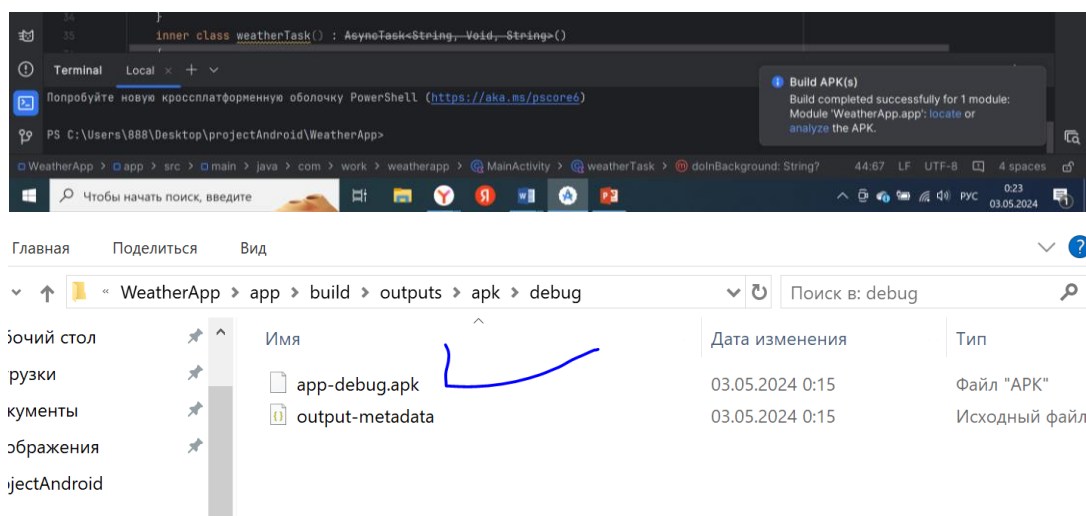


Рисунок 40 – Создание и размещение установочного файла

### 4.3 Опубликовать приложение в Google Play

Для публикации приложения в Google Play необходимо зарегистрироваться в Google Play Console, что из-за современных проблем простым способом сделать не получилось (Рисунок 41).

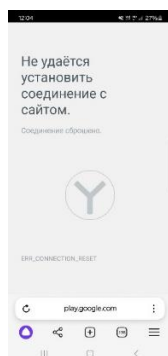


Рисунок 41 – Проблемы с регистрацией Google Play Console

Для публикации приложения следует подготовить: наименование приложения, иконку, картинку для описания, скриншоты с принципом работы и интерфейсом, видео работы приложения, краткое и полное описание, заполнить политику конфиденциальности. Aab файл, название среды разработки (Рисунок 42). Процесс публикации имеет свои особенности, возможности, ограничения, особенно, это касается приложений с финансами, для детей и информационные ресурсы, которые иногда следует отправить на тестирование [6].

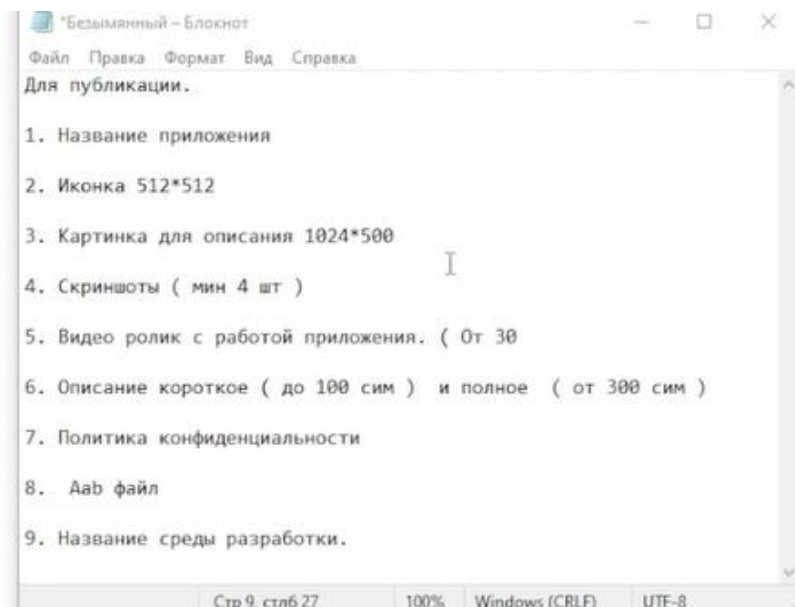


Рисунок 42 – Требования для публикации приложения

Поэтому если приложение пишется для работодателя, то будет достаточно установочного файла.

## **Заключение**

В результате выполнения работы, было разработано приложение на платформе Android для просмотра текущей погоды, погоды на день по часам и ближайшие три дня в локации пользователя. Кроме того используя поиск, пользователь может узнать погоду в интересующем его городе.

В ходе работы произошло ознакомление с большинством циклов и процессов разработки мобильного приложения, а также выполнены все поставленные задачи:

- формирование требований к мобильному приложению;
- изучить и подобрать технологии для реализации мобильного приложения;
- реализация данного мобильного приложения.

Мобильное приложение выполнено загружено на смартфон, где отображает погоду в локации пользователя и выполняет поисковый запрос.

## Список использованных источников

1. Гудзенко, А. А. Разработка мобильного Android-приложения для организации мероприятий и встреч [Текст] : выпускная квалификационная работа бакалавра . – Санкт-Петербург, 2023 . – 43 с.
2. Архитектура приложения погода RuTube . [Электронный ресурс] – режим доступа : <https://rutube.ru/video/168f0c8e7612eac57b01deecf4b5266/>
3. Битва титанов: Java vs Kotlin. [Электронный ресурс] – режим доступа : <https://tproger.ru/articles/bitva-titanov-java-vs-kotlin>
4. В чем писать код начинающему Android-разработчику: выбираем IDE . [Электронный ресурс] – режим доступа : <https://netology.ru/blog/03-2019-vybiraem-ide?ysclid=lvbaua78h4124613705>
5. Главный инструмент разработчика: что такое IDE, зачем она нужна и как её выбрать . [Электронный ресурс] – режим доступа : <https://practicum.yandex.ru/blog/integrirovannaya-sreda-razrabotki-ide/>
6. Как опубликовать приложение в Google Play console developer В 2024 году ? – YouTube . [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=Ca6xTu3Cnqs>
7. Как подключить телефон к Android Studio [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=j2OWTkQFtJ8>
8. Как сделать apk файл в Android Studio [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=18cij80c7eg>
9. Как создать погодное приложение, подобное Weather Underground или AccuWeather? [Электронный ресурс] – режим доступа : <https://appmaster.io/ru/blog/prilozhenie-pogoda-v-zdani>
10. ОСНОВНЫЕ ПРЕИМУЩЕСТВА ANDROID STUDIO . [Электронный ресурс] – режим доступа : <https://mobile.incredibleart.ru/blog/preimuschestva-android-studio/?ysclid=lvbb5bqkse759562065>

11. Создаём приложение “прогноз погоды” на Android [Электронный ресурс] – режим доступа : <https://code.tutsplus.com/ru/create-a-weather-app-on-android--cms-21587t>
12. Что такое архитектура приложения. Паттерны MVC, MVP, MVVM. – YouTube . [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=HC33Mggec3k>
13. Что такое API простыми словами? [Электронный ресурс] – режим доступа : [https://dzen.ru/video/watch/641764ebafcb9056fc5c59a7?f=d2d&utm\\_referrer=away.vk.com](https://dzen.ru/video/watch/641764ebafcb9056fc5c59a7?f=d2d&utm_referrer=away.vk.com)
14. Что такое MVC за 4 минуты – YouTube . [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=NDOPFWOId28>
15. Android Studio: среда разработки мобильных приложений . [Электронный ресурс] – режим доступа : <https://arduinoplus.ru/android-studio/?ysclid=lvbasakjem240703017>
16. Building a Weather App with Kotlin and OpenWeatherMap API . [Электронный ресурс] – режим доступа : <https://dopebase.com/building-weather-app-kotlin-openweathermap-api>
17. Download and install Android Studio . [Электронный ресурс] – режим доступа : <https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio#1>
18. Java vs Kotlin – большой обзор. [Электронный ресурс] – режим доступа : <https://www.sravni.ru/kursy/info/java-vs-kotlin/>
19. Kotlin за час. Теория и практика. – YouTube . [Электронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=30tchn0TjaM>
20. KOTLIN VS JAVA: ЩО КРАЩЕ ДЛЯ ANDROID-РОЗРОБКИ? . [Электронный ресурс] – режим доступа : <https://itvdn.com/ru/blog/article/kotlin-vs-java>



21. MVC, MVVM Архитектура. Наглядная теория и примеры – YouTube . [Элек-тронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=X85soC5evw0>
22. Make a Weather App for Android | Android Studio | Kotlin – YouTube . [Элек-тронный ресурс] – режим доступа : <https://www.youtube.com/watch?v=gj0g1a75Lmo>
23. OpenWeatherMap [Электронный ресурс] – режим доступа : <https://openweathermap.org/>
24. WEATHER APP part 1 retrofit setup. Android studio | Kotlin – YouTube. [Электронный ресурс] – режим доступа : [https://www.youtube.com/watch?v=W\\_oe5aTEJZ4](https://www.youtube.com/watch?v=W_oe5aTEJZ4)
25. Видеоуроки Android приложение Прогноз погоды на котлин – YouTube. [Электронный ресурс] – режим доступа: [https://youtube.com/playlist?list=PLmjT2NFTgg1f9uVjP5EBtdRp\\_VVvha6GH&si=u2K7ACTNNK7rzx1E](https://youtube.com/playlist?list=PLmjT2NFTgg1f9uVjP5EBtdRp_VVvha6GH&si=u2K7ACTNNK7rzx1E)
26. Free Weather API [Электронный ресурс] – режим доступа : <https://www.weatherapi.com/>