

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

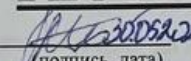
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

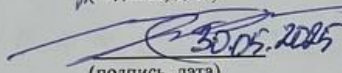
на тему

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ «ПОГОДА» НА
FLUTTER**

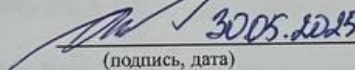
Студент

 Кривенкова Валерия Сергеевна
(подпись, дата) (ФИО полностью)

Руководитель

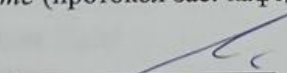
 Зинкин С.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 20.05.2025 г. № 16)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

6 июня 2025 г. № 9)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » 05 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кривенкова Валерия Сергеевна гр. 21ВВП1 факультета вычислительной техники
2. Тема работы Мобильное приложение «Погода» на Flutter.

Тема утверждена приказом ПГУ № 801/о _____ от "6" _____ мая _____ 2025 г.

3. Руководитель бакалаврской работы д.т.н., профессор кафедры ВТ ПГУ Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

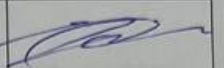
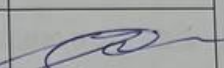
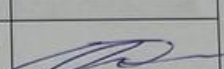
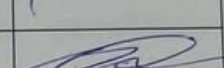
Разработать android-приложение. Язык программирования Flutter, среда разработки Android studio. Программа должна обладать интуитивно понятным интерфейсом в приложении и использовать следующие технологии:

1. Язык программирования Flutter
2. JSON и файловая система
3. Асинхронное программирование

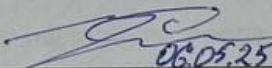
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения. Выбрать язык программирования, среду разработки и базу данных.

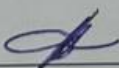
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	45	19.04.2025– 25.04.2025	
2	Анализ требований	35	26.04.2025 – 30.04.2025	
3	Выбор технологий, применяемых при проектировании и разработке	55	01.05.2025– 06.05.2025	
4	Создание графического интерфейса	65	06.05.2025 – 14.05.2025	
5	Разработка мобильного приложения	84	25.05.2025 – 27.05.2025	
6	Оформление пояснительной записки	40	28.05.2025 – 30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы _____  06.05.25 Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент _____  06.05.25 Кривенкова В.С.
(подпись, дата)

Работу к защите допустить
Декан факультета _____  05.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 99 листов, 33 рисунка, 0 таблиц, 6 источников и 3 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ПОГОДА, АСТРОНОМИЧЕСКИЕ ДАННЫЕ
ФАЙЛОВАЯ СИСТЕМА, HTTP API ИНТЕГРАЦИЯ, FLUTTER-ПРИЛОЖЕНИЕ,
ANDROID STUDIO, ANDROID

Цель работы – разработать мобильное приложение прогноза погоды. Приложение позволяет пользователям получать актуальную информацию о погоде, включая текущий прогноз, почасовые изменения, графическое представление динамики температур и астрономические данные.

В рамках проекта были реализованы следующие задачи: разработка клиентской части, интеграция с внешними API, формирование модульной архитектуры, управление состоянием и локальное хранение данных, экспорт данных.

В результате была создана мобильная программа на языке Flutter, полностью соответствующая заданным требованиям. Приложение предназначено для устройств на базе Android и разрабатывалось в среде Android Studio, что обеспечило удобство и эффективность процесса разработки. Оно успешно выполняет все поставленные задачи и обладает стабильной работой на целевых устройствах.

					ПГУ1.090301.12.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Мобильное приложение «Погода» на Flutter Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Кривенкова В.С.		30.05.23					
Провер.		Зинкин С.А.		30.05.23			2	93	
Н. Контр		Бычков А.С.		30.05.23		ФВТ, каф. ВТ гр. 21ВВП1			
Утверд.		Митрохин М.А.		30.05.23					

Содержание

Перечень сокращений и обозначений	7
Введение	8
1 Мобильное приложение «Погода»	9
1.1 Анализ предметной темы.....	9
1.2 Анализ существующих аналогов.....	10
1.2.1 Мобильное приложение «Яндекс.Погода»:	10
1.2.2 Мобильное приложение «Gismeteo»	12
1.2.3 Мобильное приложение «Погода»	13
1.2.4 Вывод	14
1.3 Постановка задачи.....	14
1.4 Выбор инструментов разработки	15
1.4.1 Выбор языка программирования	15
1.4.2 Выбор среды разработки.....	17
1.4.3 Обмен данными JSON.....	17
1.5 Установка и настройка инструментов разработки	17
1.5.1 Установка Android Studio	17
1.5.2 Создание проекта	18
1.5.3 Настройка проекта	19
1.5.4 Добавление зависимостей.....	21
1.6 Архитектура приложения.....	22
1.6.1 Архитектурный паттерн	22
1.6.2 Архитектура.....	23
1.7 Описание приложения.....	30
1.7.1 Описание функционала.....	30
1.7.2 Описание работы приложения.....	32
1.7.3 Основные компоненты приложения	33
1.7.4 Описание интерфейса приложения	34
2 Безопасность жизнедеятельности	40
2.1 Правила и техника безопасности при работе с ПК.....	40
2.2 Правила и техника безопасности при работе со смартфоном.....	41
Заключение	42

Список использованных источников.....	43
Приложение А– UML-диаграммы.....	44
Приложение Б– Листинг программы	50
Приложение В– Презентация	95

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

Перечень сокращений и обозначений

ОС – операционная система, управляющая аппаратными и программными ресурсами устройства

ПК – персональный компьютер, предназначенный для индивидуального использования

ПО – программное обеспечение

API – (Application Programming Interface) — интерфейс программирования приложений, который позволяет взаимодействовать между различными программными компонентами

IDE – (Integrated Development Environment) – интегрированная среда разработки

UI – (User Interface) – пользовательский интерфейс

UML – (Unified Modeling Language) – унифицированный язык моделирования

UX – (User Experience) – это совокупность всех аспектов взаимодействия пользователя с продуктом, системой или сервисом

					ПГУ1.090301.12.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В наше время информация о погоде играет значимую роль в повседневной жизни, помогая планировать свои дела, выбирать одежду и организовывать поездки. На сегодняшний день быстрый доступ к актуальным прогнозам стал необходимостью, мобильные приложения для отслеживания погоды стали очень популярны.

Разработка приложения для погоды является интересной задачей и позволяет значительно упростить жизнь пользователям. Благодаря современным приложениям и информации об изменении погоды люди могут заранее подготовиться к любым условиям.

Цель данной работы – разработка приложения для прогноза погоды. В рамках проекта необходимо:

- изучить особенности работы мобильных сервисов погоды;
- определить ключевые функции и создать удобный интерфейс для взаимодействия с приложением;
- разработать современное приложение, которое сможет предоставить точные прогнозы;
- протестировать систему, чтобы повысить ее надёжность.

Таким образом, данное мобильное приложение для прогноза погоды будет отвечать актуальным запросам пользователей и представлять собой интересный проект в области мобильных технологий.

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

1 Мобильное приложение «Погода»

1.1 Анализ предметной темы

В современном мире информация о погоде играет ключевую роль в жизни людей. Правильное понимание условий погоды позволяет планировать дела, организовывать поездки и выбирать подходящую одежду. Большинство людей сталкиваются с проблемой поиска точного прогноза, что часто требует загрузки нескольких приложений.

Приложение для прогноза погоды становится хорошим решением этой проблемы. Оно позволяет пользователям получать актуальную информацию о погодных условиях в реальном времени и прогнозы на несколько дней вперед. Приложение предложит широкий набор функций, таких как данные о температуре, ветре, шанса дождя и снега, видимость. Это делает планирование более простым.

Одним из главных преимуществ приложения является его доступность. Пользователям не нужно тратить время на поиск информации в интернете, вся необходимая информация доступна в одном месте. Более того, простота интерфейса позволяет любой категории пользователей легко разобраться с функционалом приложения, даже тем, кто не является технически подкованным.

Дополнительной возможностью является смена места и сохранение данных о погоде в файл. Также приложение может позволять сохранять местоположения, чтобы быстро получать необходимую информацию о погоде в разных городах и странах.

Автоматические обновления приложения избавляют пользователей от необходимости вручную загружать новые версии, что гарантирует доступ к актуальной информации без каких-либо затрат времени. Таким образом, пользователи могут сосредоточиться на своих делах, не отвлекаясь на обновление приложений.

					ПГУ1.090301.12.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Анализ существующих аналогов

На сегодняшний день существует множество приложений для прогнозирования погоды, каждое из которых имеет свои особенности. Рассмотрим 3 популярных сервиса:

- Яндекс.Погода;
- Gismeteo;
- Приложение «Погода».

Проведен анализ и выявлены слабые и сильные стороны каждого приложения.

1.2.1 Мобильное приложение «Яндекс.Погода»:

Функционал:

- актуальные данные о погоде на 10 дней;
- карта осадков и температур с возможностью настройки уведомлений;
- визуализация параметров погоды, включая скорость ветра, влажность и атмосферное давление.

Достоинства:

- удобный и понятный интерфейс;
- высокая точность прогнозов;
- возможность добавления нескольких локаций в избранное для быстрого доступа.

Недостатки:

- иногда возникают проблемы с обновлением данных в некоторых регионах;
- ограниченные функции в сравнении с некоторыми специализированными метеосервисами.

Интерфейс приложения представлен на рисунках 1 и 2.



Рисунок 1 – Главная страница приложения

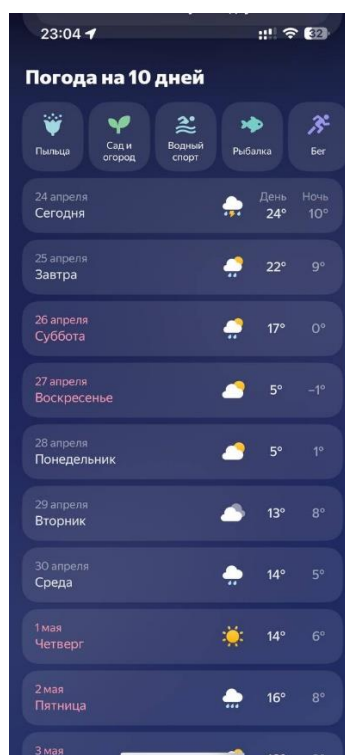


Рисунок 2 – Прогноз погоды

1.2.2 Мобильное приложение «Gismeteo»

Функционал:

- прогнозы на 10 дней с разбивкой по часам;
- данные о ветре, давлении, осадках и влажности;
- метеокарты с показателями погоды;
- мобильная версия и возможность использования на различных устройствах.

Достоинства:

- подробный и развернутый прогноз, включая погодные явления;
- наличие исторических данных о погоде;
- гибкость в выборе параметров отображения.

Недостатки:

- реклама в бесплатной версии может отвлекать от основного контента;
- некоторые пользователи отмечают недостаточную точность для отдаленных регионов.

Интерфейс приложения представлен на рисунке 3.



Рисунок 3 – Главная страница приложения

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

1.2.3 Мобильное приложение «Погода»

Функционал:

- прогнозы на 10 дней с разбивкой по часам;
- данные о ветре, давлении, осадках и влажности;
- метеокарты с показателями погоды.

Достоинства:

- удобный интерфейс и доступность на мобильных устройствах;
- наличие ярких и информативных графиков.

Недостатки:

- иногда прогнозы могут быть менее точными, чем у конкурентов;
- ограниченные возможности в плане пользовательских настроек.

Интерфейс приложения представлен на рисунке 4 и 5.

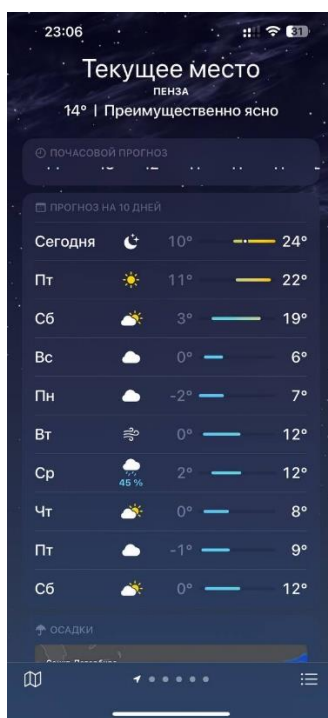


Рисунок 4 – Главная страница приложения

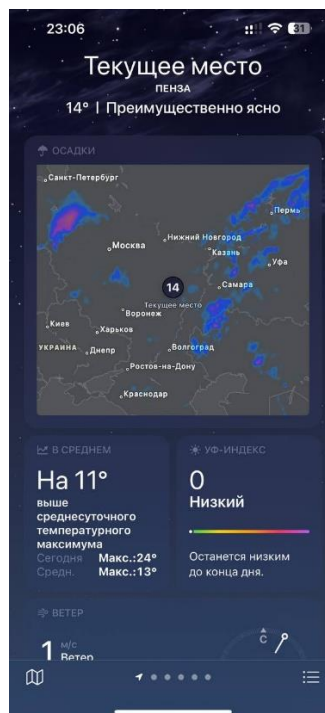


Рисунок 5 – Карта погоды

1.2.4 Вывод

Каждое из этих приложений имеет свои особенности и недостатки. Яндекс.Погода выделяется своей простотой и удобством, Gismeteo предоставляет более детализированные метеоданные и долгосрочные прогнозы, в то время как «Погода» предоставляет свои уникальные решения. Выбор подходящего варианта зависит от того, что именно важно для пользователя: точность прогноза, дополнительные сведения о погодных условиях или удобство приложения.

1.3 Постановка задачи

Важно создать приложение для прогноза погоды, которое предоставляет пользователям актуальные данные о погоде. Это приложение должно определять погоду в зависимости от местоположения пользователя.

Основная цель приложения - обеспечить пользователей легким доступом к актуальной информации о погоде, чтобы они могли планировать свои дела.

Кроме того, приложение должно обладать высокой производительностью и быть легко масштабируемым. Это позволит в будущем добавлять новые функции и улучшения, если возникнет такая необходимость.

					ПГУ1.090301.12.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

В приложении должно быть реализовано:

- отображение погоды на несколько дней;
- отображение текущей погоды;
- выбор местоположения для отображения погоды;
- отображение дополнительных данных: влажность, ветер.

При разработке автоматизированной системы необходимо учитывать ряд технических требований, которые обеспечат её стабильную работу и совместимость с современными устройствами:

- операционная система, поддержка Android 8.0 (Oreo) и выше;
- графический ускоритель: требуется для корректной эмуляции среды Android / iOS;
- не менее 8 Гб оперативной памяти;
- интернет: высокоскоростное подключение для загрузки зависимостей и взаимодействия с API.

1.4 Выбор инструментов разработки

1.4.1 Выбор языка программирования

Чтобы разработать качественное приложение и оптимизировать процесс разработки, необходимо правильно выбрать инструменты. Это поможет избежать возможных сложностей, сократить затраты времени и ресурсов, а также повысить удобство работы с продуктом. Выбор языка программирования для Android, одной из самых популярных мобильных операционных систем, имеет большое значение. Важно подобрать язык, который станет эффективным инструментом для разработки безопасного и удобного приложения. Ниже представлены языки, используемые для создания приложений на Android:

- Java;
- Kotlin;
- C++;
- Flutter.

Чтобы определить, какой язык подходит лучше всего, проведем анализ и рассмотрим особенности каждого из них.

Язык программирования «Java». Java один из официальных языков для разработки Android-приложений, который широко используется в среде Android Studio. Благодаря большому сообществу разработчиков, обширному набору библиотек и фреймворков Java остаётся надёжным и востребованным инструментом для создания мобильных приложений.

Язык программирования «Kotlin». Kotlin, также официальный язык для Android, который в последние годы стал более предпочтительным для разработки, чем Java. Он предлагает удобную типизацию, компактный синтаксис, поддержку корутин и функционального программирования, что делает процесс создания приложений более быстрым и эффективным. Сообщество Kotlin активно развивается, предоставляя разработчикам поддержку и готовые решения, что делает этот язык отличным выбором для мобильной разработки.

Язык программирования «C++». Благодаря высокой производительности активно применяется в мобильной разработке, но чаще используется для специфических задач, таких как разработка 3D-игр и взаимодействие с низкоуровневыми компонентами Android. Однако сложность языка, отсутствие встроенного сборщика мусора и работа с указателями делают его более трудным для написания мобильных приложений по сравнению с Java или Kotlin.

Язык программирования «Flutter». Flutter язык для разработки нативных приложений на iOS и Android из единого кода. Он использует язык программирования Dart. Flutter предоставляет богатый набор виджетов, интерфейсы и высокую производительность, что позволяет создавать красивые и многозадачные приложения.

Вывод: в результате анализа различных вариантов было принято решение использовать Flutter в качестве основного инструмента для разработки приложения погоды. Этот фреймворк позволяет создать безопасное, удобное и экономичное приложение. Благодаря Flutter разработка становится более эффективной, а конечный продукт — стабильным и доступным для пользователей.

					<i>ПГУ1.090301.12.001 ПЗ</i>	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

1.4.2 Выбор среды разработки

Одной из самых популярных и официально поддерживаемых сред разработки для Android является - Android Studio. Эта IDE была специально создана для программирования мобильных приложений и признана официальным инструментом для работы с платформой Android. Android Studio основана на IntelliJ IDEA, благодаря чему обладает удобными функциями для редактирования кода, управления файлами проекта и проверки синтаксиса.

Для разработки мобильных приложений можно использовать такие IDE как IntelliJ, Eclipse и Visual Studio, но для разработки приложения на базе операционной системы Android они подходят в меньшей степени, поэтому была выбрана Android Studio.

1.4.3 Обмен данными JSON

Для обмена информацией с удалёнными сервисами, которые предоставляют данные о погоде, в разработке приложения используется формат JSON (JavaScript Object Notation). JSON – это легковесный текстовый формат, обладающий простым и понятным синтаксисом, который применяется для обмена структурированными данными в широком спектре языков программирования. В рамках данного проекта для автоматизированного преобразования JSON-ответов в объекты Dart и обратно применяется библиотека `json_serializable`. Это позволяет значительно сэкономить время разработчика, минимизировать вероятность ошибок при создании моделей данных и обеспечить быстрый и эффективный обмен информацией между клиентской частью приложения и внешними API. Благодаря использованию данного подхода, интеграция с метеосервисами проходит без лишних сложностей, а добавление новых функций в приложение становится лёгким и удобным.

1.5 Установка и настройка инструментов разработки

1.5.1 Установка Android Studio

Для начала работы над проектом скачана и установлена интегрированная среда с официального сайта (рисунок 6). Для разработки ПО была выбрана версия Meerkat. После скачивания и установки .exe файла запускается сама программа.

					ПГУ1.090301.12.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

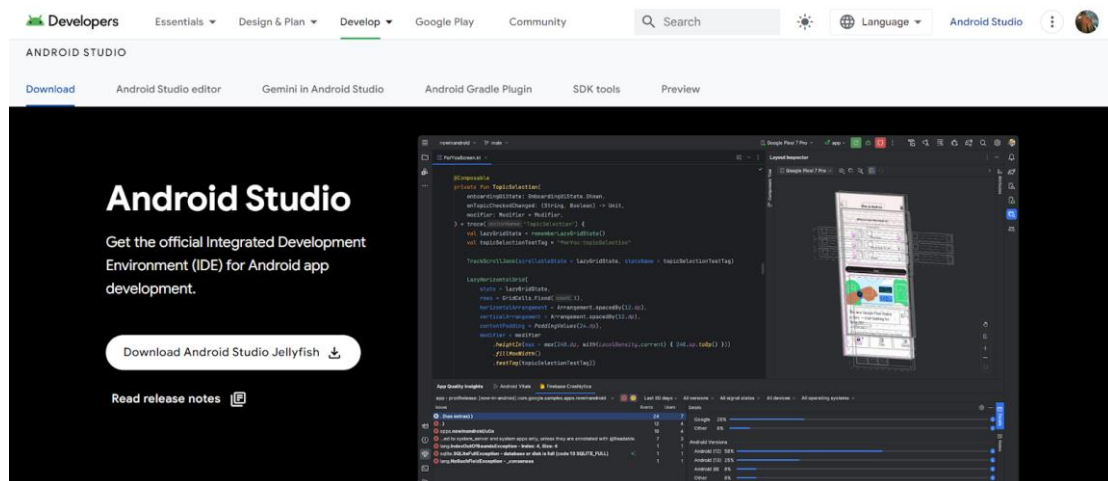


Рисунок 6 - Официальный сайт Android Studio

1.5.2 Создание проекта

При запуске программы открывается окно выбора шаблона. Выбран пустой шаблон “No Activity”.

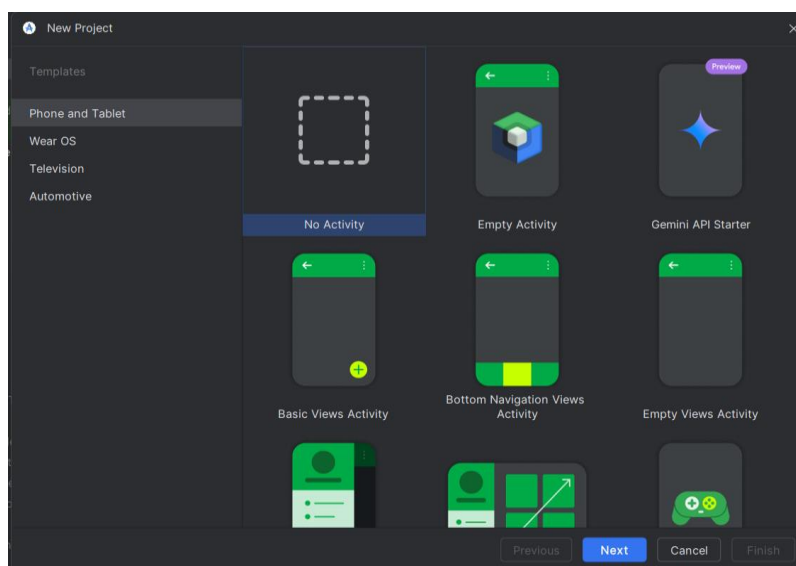


Рисунок 7 - Выбор шаблона проекта

После выбора шаблона устанавливаем название проекта, название пакета, где будут храниться все файлы, место сохранения, язык программирования и минимальную SDK.

Интерфейс этого окна представлен на рисунке 8.

Изм.	Лист	№ докум.	Подпись	Дата

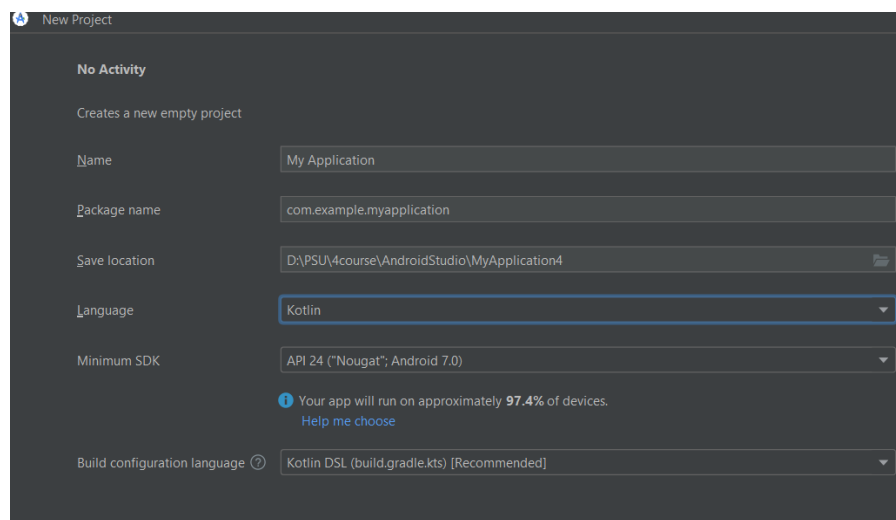


Рисунок 8 - Заполнение финальных свойств проекта

На рисунке 9 можно увидеть, как будет выглядеть уже созданный проект.

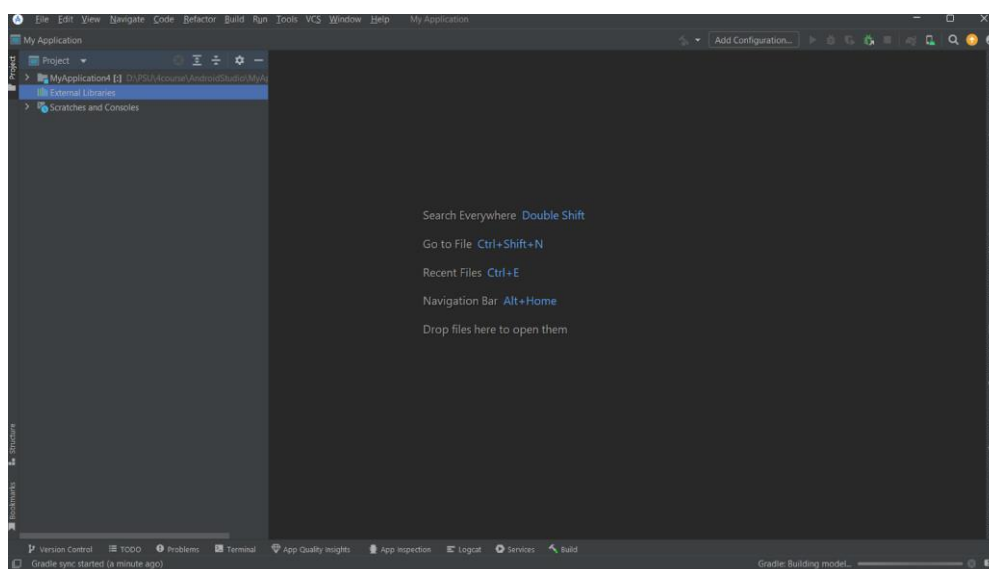


Рисунок 9 - Пустой проект

1.5.3 Настройка проекта

Изначально дерево проекта отображается как Android, для того чтобы поменять это требуется нажать на список представления и выбрать Project (рисунки 10 и 11).

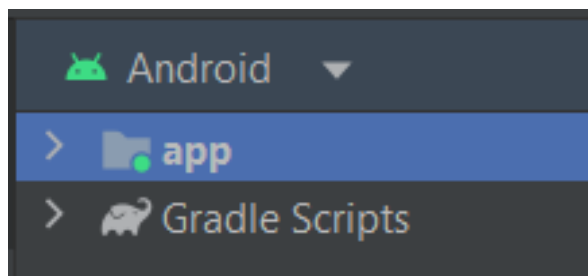


Рисунок 10 - Смена типа дерева проекта

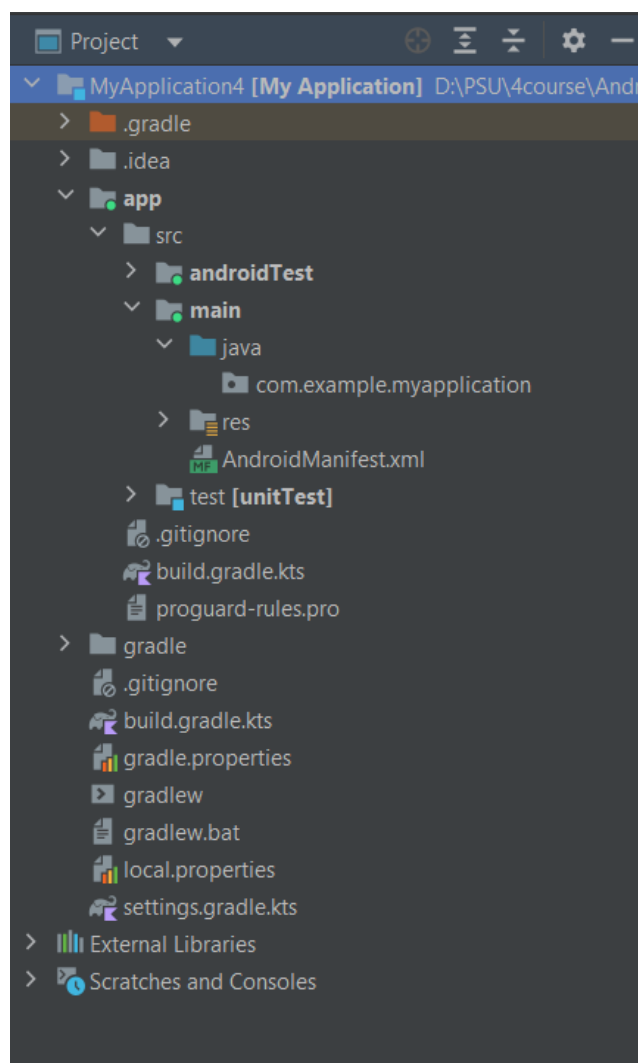


Рисунок 11 - Вид дерева проекта как Project.

Теперь в проект нужно добавить необходимые папки как: menu, layout, font для работы с ресурсами Android, raw. Для этого нужно нажать правой кнопкой мыши по папке res, выбрать пункт меню «New» и «Directory» (рисунки 12 и 13).

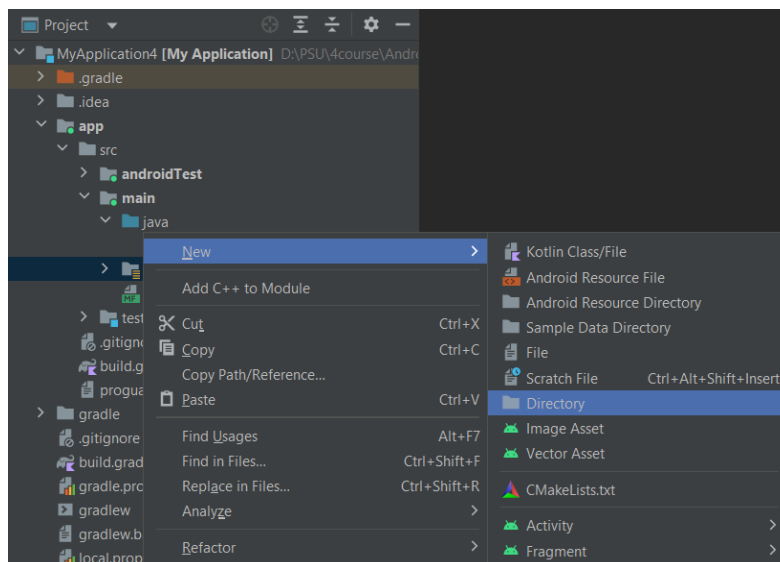


Рисунок 12 - Подготовка к созданию дополнительных папок

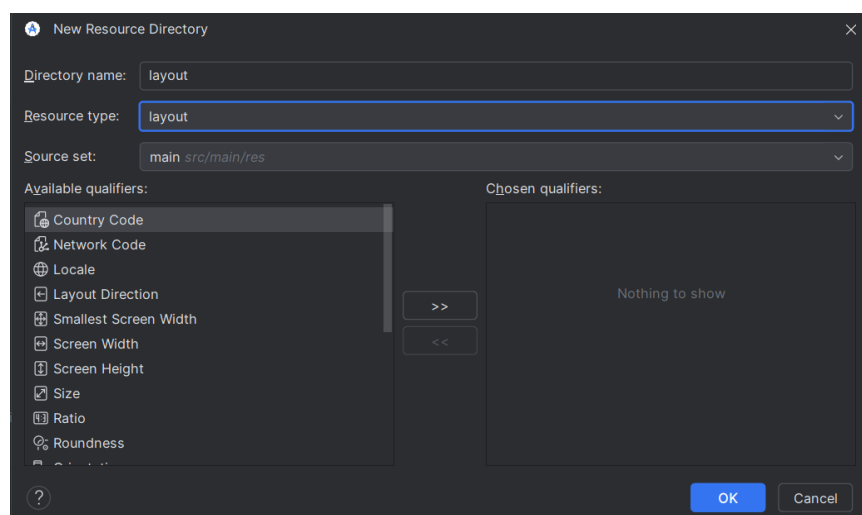


Рисунок 13 - Создание папки layout.

1.5.4 Добавление зависимостей

Последним шагом в настройке IDE является добавление зависимостей в проект. Это необходимо для подключения сторонних библиотек, что позволяет разработчику использовать их функциональность. Чтобы добавить зависимости, нужно открыть файл «build.gradle.kts» в корневой директории проекта (рисунки 14 и 15).

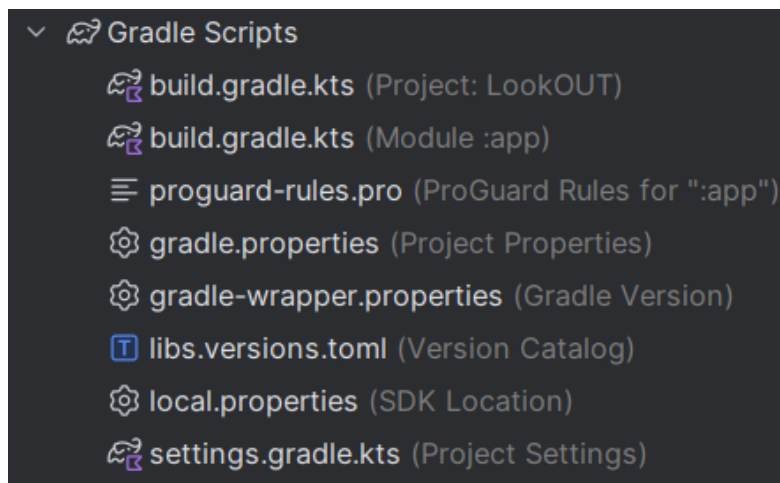


Рисунок 14 – Файлы для настройки зависимостей

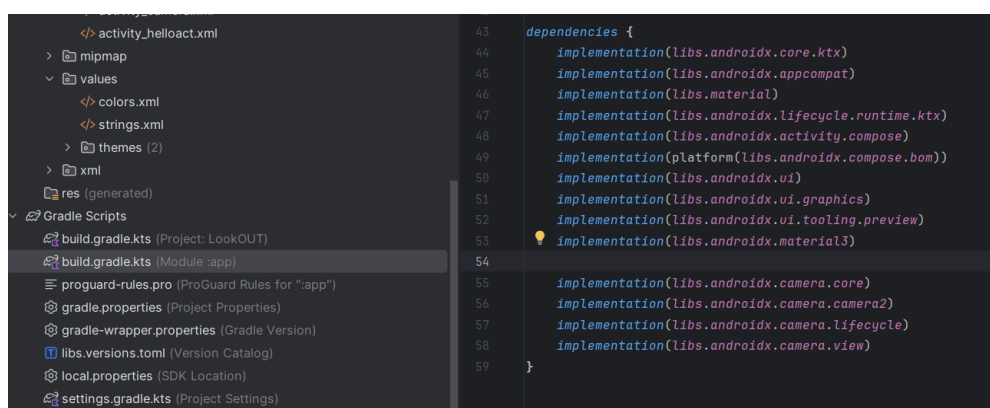


Рисунок 15 - Добавление новых зависимостей.

После этого проект полностью настроен и готов к работе с ним.

1.6 Архитектура приложения

1.6.1 Архитектурный паттерн

При разработке приложения основным паттерном разработки является MVVM. Паттерн Model View ViewModel (MVVM) помогает разделить логику представления, обработку данных и бизнес-процессы. Этот подход значительно упрощает тестирование, поддержку и расширение функционала приложения, а также делает код более структурированным и удобным для повторного использования.

Паттерн состоит из трёх отдельных компонентов:

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		22

- представление (View), отвечает за отображение интерфейса и взаимодействие с пользователем;
- модель представления (ViewModel), связывает пользовательский интерфейс с данными, управляет логикой представления и состоянием приложения;
- модель (Model), содержит основную информацию и отвечает за работу с данными, управляет получением информации из внешних источников, взаимодействует с базами данных и выполняет бизнес-логику..

На рисунке 16 показаны компоненты, которые используются в паттерне MVVM.

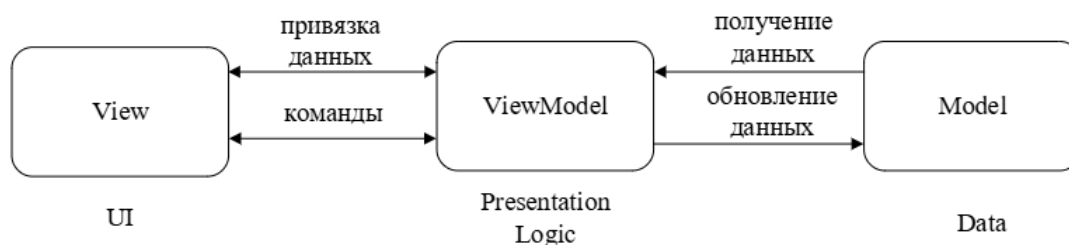


Рисунок 16 – Интерфейс для фактической нагрузки

1.6.2 Архитектура

Архитектура разработанного погодного приложения включает несколько ключевых слоёв, каждый из которых выполняет свою роль и обеспечивает стабильную работу системы.

Пользовательский интерфейс реализован на платформе Flutter, что гарантирует удобное взаимодействие с пользователем. В приложении можно легко просматривать текущую погоду, почасовой прогноз и дополнительные параметры, такие как влажность, скорость ветра и атмосферное давление.

Для лучшего понимания логики работы приложения и способов взаимодействия пользователя с его функционалом можно ознакомиться с UML-диаграммой вариантов использования (рисунок 17), где подробно показаны ключевые процессы и связи между компонентами системы.

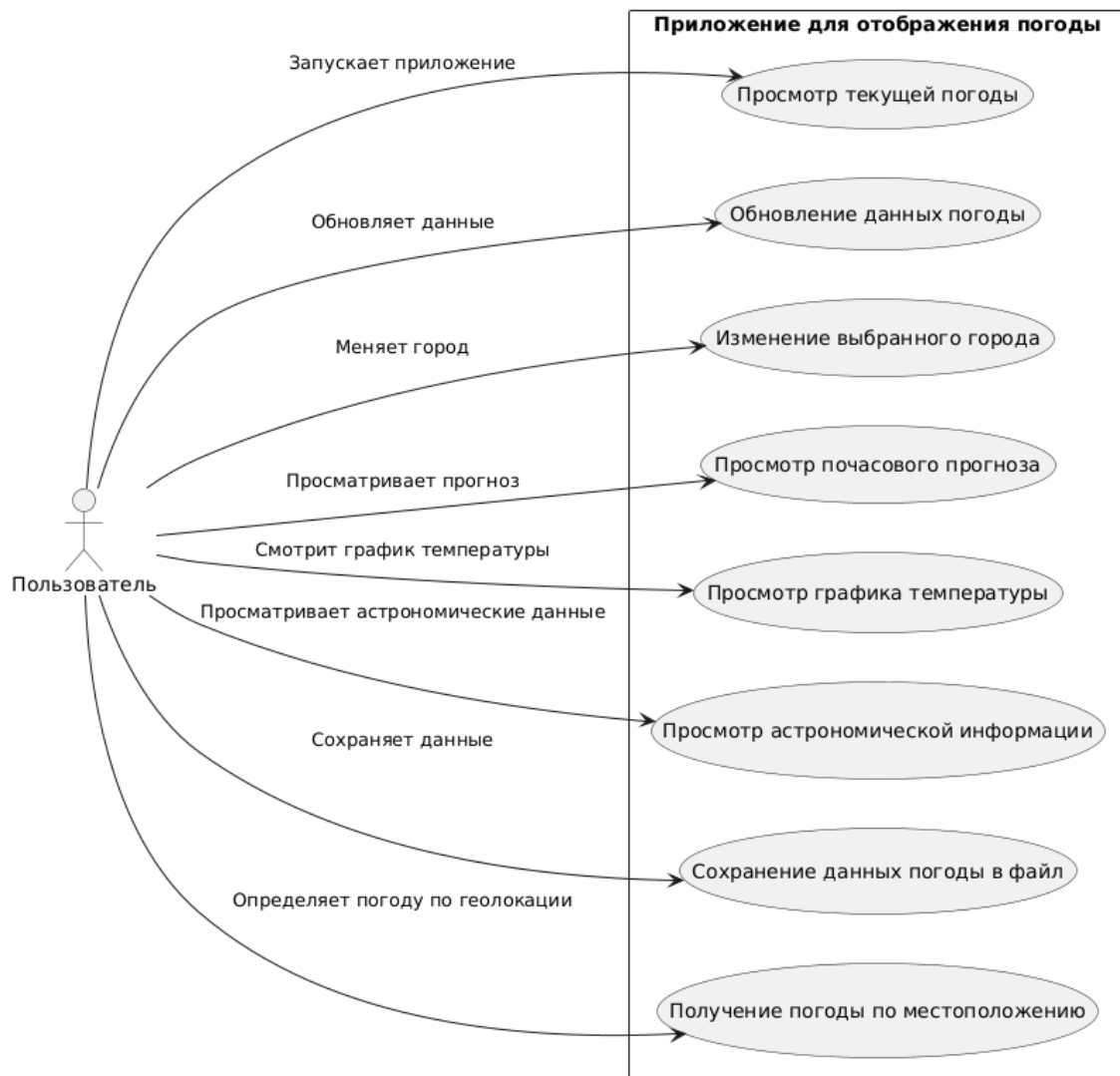


Рисунок 17 – UML-диаграмма вариантов использования

Для более точного понимания взаимодействия пользователя с приложением создана UML-диаграмма последовательности (рисунок 18).

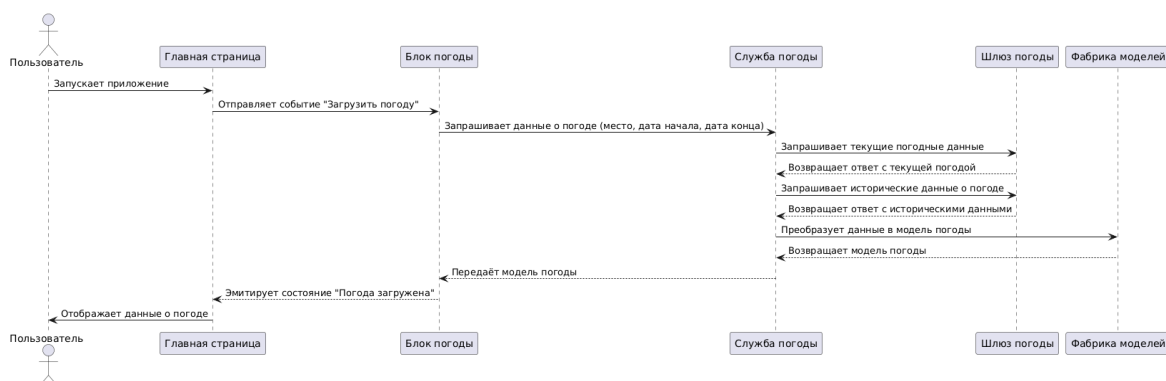


Рисунок 18 – UML-диаграмма последовательности

Так как при создании приложения для отображения метеоданных использовался фреймворк Flutter, архитектура системы построена с учётом современных принципов модульного проектирования. Такой подход позволяет разделить систему на независимые слои, что упрощает поддержку, тестирование и дальнейшее развитие функционала.

Структура приложения включает в себя несколько слоев:

– Presentation Layer (слой представлений) - в этом слое реализован весь пользовательский интерфейс приложения. Здесь используются стандартные Flutter-виджеты, которые обеспечивают удобное взаимодействие с пользователем. Экран с текущей погодой, а также специальные виджеты для отображения почасового прогноза, графика температуры и астрономических данных, предоставляют пользователю всю необходимую информацию в наглядном виде. Визуальные эффекты, такие как размытие через BlurryContainer, улучшают восприятие интерфейса и делают его более современным;

– Business Logic Layer (слой бизнес-логики) - логика приложения реализована с применением паттерна Bloc при помощи библиотеки flutter_bloc. Этот подход позволяет отделить обработку событий от отображения данных. WeatherBloc принимает события и эмитирует соответствующие состояния, такие как загрузка, получение данных или сообщение об ошибке. Это делает работу приложения структурированной;

– Service Layer (слой сервисов) - для работы с данными о погоде выделен отдельный слой сервисов. WeatherService, реализованный в виде WeatherServiceImpl, отвечает за получение информации с внешнего API. Он взаимодействует с WeatherGateway, который выполняет HTTP-запросы через специализированный HttpClient и получает данные в формате JSON. Результаты запросов преобразуются в удобные модели через WeatherFactory, что позволяет использовать их напрямую в бизнес-логике и презентационном слое;

– Data Layer (слой данных) - слой данных включает в себя все компоненты, отвечающие за обмен информацией с удалёнными сервисами и работу с локальным хранилищем. HttpClient выполняет HTTP-запросы к внешнему API погоды, а полученные JSON-ответы преобразуются с помощью библиотеки json_serializable в строго типизированные модели. Такие модели, как WeatherResponse и WeatherHistoryResponse, содержат всю необходимую информацию для формирования прогноза. Помимо этого, для сохранения локальных настроек используется SharedPreferences;

– Dependency Injection & Utility Layer (слой зависимостей и утилит) - вся система зависит от правильно настроенного механизма внедрения зависимостей. С помощью библиотеки GetIt реализована регистрация и предоставление нужных экземпляров сервисов, gateway-ов, фабрик и других классов. Дополнительные утилиты для работы с форматированием дат, логированием и управлением файлами обеспечивают удобство разработки и стабильную работу приложения.

Таким образом, выбранная архитектура, разделённая на независимые слои, позволяет создать гибкое, масштабируемое и легко поддерживаемое приложение для отображения актуальных погодных данных. Такой подход способствует повышению тестируемости кода, упрощает внесение изменений и дальнейшее расширение функционала, что является важным аспектом при разработке современных мобильных приложений.

Структура погодных данных описана в моделях, таких как CurrentWeatherModel, WeatherItemModel, WeatherDayModel, HourModel и Astro. CurrentWeatherModel содержит общую информацию о погоде в конкретном городе, включая название города, региона, текущую температуру и список прогнозов. WeatherItemModel объединяет дневной прогноз, почасовые данные и астрономическую информацию для конкретного дня, что помогает гибко организовать отображение информации в UI. Дополнительные модели, такие как WeatherDayModel и HourModel, структурируют детали, например, максимальные и минимальные температуры, иконки состояния погоды, а Astro хранит данные о времени восхода и заката солнца и луны.

					ПГУ1.090301.12.001 ПЗ	Лист
						26
Изм.	Лист	№ докум.	Подпись	Дата		

Визуальные компоненты: на уровне представления также используются специализированные виджеты, такие как WeatherGraphWidget, который визуализирует почасовые данные о погоде в виде графика, и WeatherAstroWidget, отображающий астрономические данные. Функция showSnackBar отвечает за уведомление пользователя о результатах операций, например, при сохранении погодных данных в локальный файл.

Рассмотрим классы, которые я использовала. Класс WeatherService отвечает за получение информации о погоде. Его реализация, WeatherServiceImpl, объединяет два важных компонента: WeatherGateway – занимается отправкой HTTP-запросов к API и получением исходных данных о погодных условиях, и WeatherFactory – компонент, преобразующий полученные данные в структурированную модель CurrentWeatherModel. Класс WeatherBloc служит для управления состояниями пользовательского интерфейса. Он реагирует на действия пользователя, такие как ввод города или определение местоположения, и передаёт соответствующее состояние, например, загрузку данных, успешное получение информации или сообщение об ошибке. Это позволяет обеспечить динамичное обновление UI и разделение бизнес-логики от представления.

Модели данных описывают структуру погодной информации в приложении. CurrentWeatherModel хранит сводные данные о погоде в выбранном городе, включая название города, региона, текущую температуру и список прогнозов. WeatherItemModel представляет данные о погоде на конкретный день, объединяя дневной прогноз, почасовые данные и астрономическую информацию. Дополнительные модели, такие как WeatherDayModel и HourModel, структурируют информацию о температуре, погодных условиях и других параметрах, а Astro содержит данные о времени восхода и заката солнца и луны.

Пользовательский интерфейс строится на основе моделей и реализован в виде набора виджетов. MyApp и CurrentWeatherPage создают базовую структуру приложения и отвечают за настройку темы, ориентации экрана и первичное отображение данных. Виджеты, такие как WeatherAstroWidget, WeatherGraphWidget и WeatherHourWidget, используют данные моделей для

построения информативного интерфейса с графиками, иконками и текстовыми описаниями.

Архитектура приложения основана на принципе четкого разделения ответственности между его компонентами. Данные о погоде сначала запрашиваются через API-слой, затем проходят обработку в сервисах и фабриках, после чего передаются в BLoC для управления состоянием. В конечном итоге информация отображается на экране с помощью виджетов. Такая структура не только упрощает сопровождение и расширение функционала приложения, но и обеспечивает его масштабируемость и устойчивость к изменениям внешних API.

На рисунке 19 представлена UML-диаграмма классов.

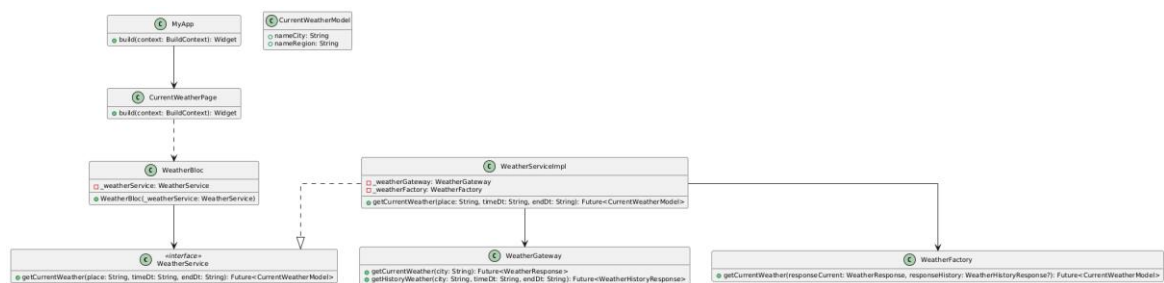


Рисунок 19 – UML-диаграмма классов

На рисунке 20 представлена UML-диаграмма деятельности.

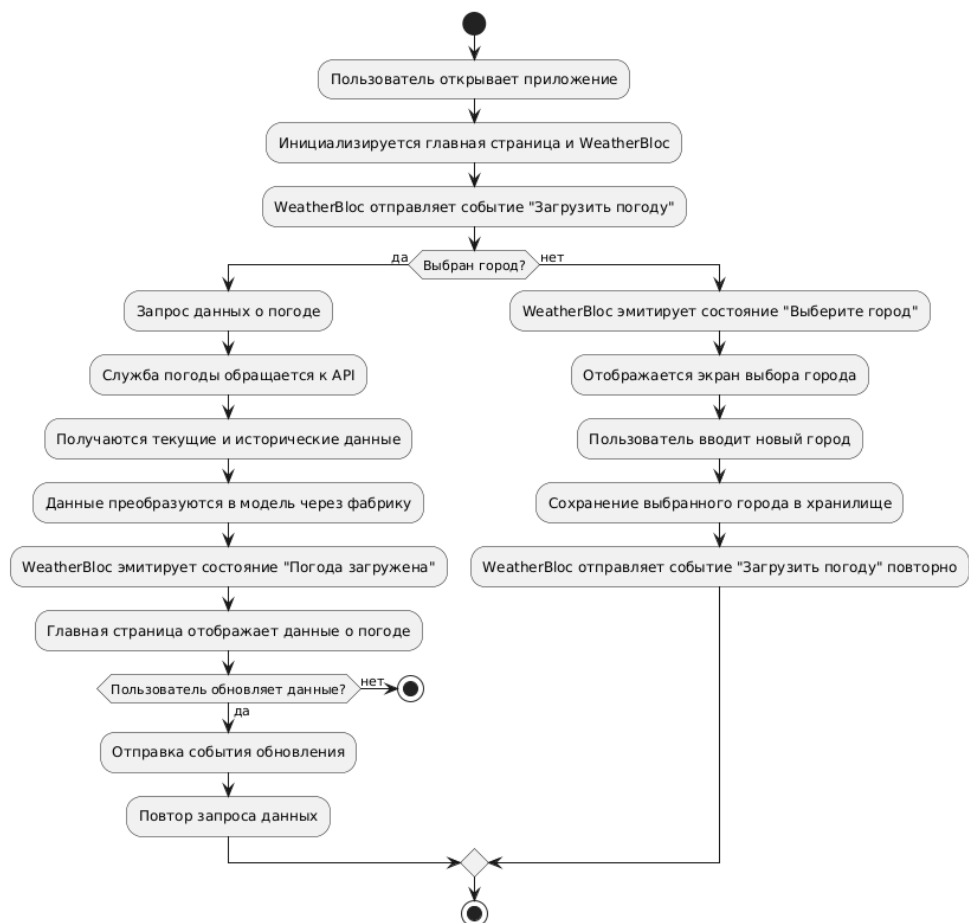


Рисунок 20 – UML-диаграмма деятельности

На рисунке 21 представлена UML-диаграмма развертывания.

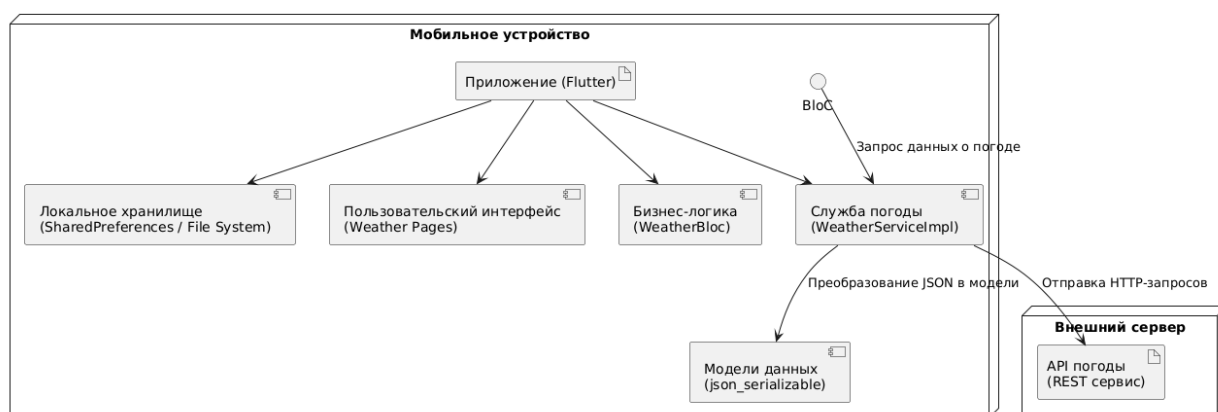


Рисунок 21 – UML-диаграмма развертывания

1.7 Описание приложения

1.7.1 Описание функционала

В разработанном приложении реализована система выбора и сохранения городов, что позволяет пользователям получать персонализированные прогнозы погоды. Пользователь имеет возможность самостоятельно указать интересующий его город или воспользоваться ранее сохранёнными локациями, что значительно упрощает получение актуальной информации. Сохранение настроек осуществляется с помощью локального хранилища (Shared Preferences), что обеспечивает мгновенную загрузку данных при последующих запусках приложения.

После выбора города пользователю доступны следующие возможности:

- просмотр текущего прогноза: отображаются данные о текущей температуре, ощущаемом значении, погодное описание и другие ключевые показатели, обновляющиеся в реальном времени;

- просмотр текущего прогноза: отображаются данные о текущей температуре, ощущаемом значении, погодное описание и другие ключевые показатели, обновляющиеся в реальном времени;

- детализированный почасовой прогноз: приложение предоставляет информацию о погодных условиях на каждый час, включая температуру, направление ветра, облачность, вероятность осадков и другие параметры;

- графическая визуализация: для наглядного представления динамики изменений используется диаграмма, отображающая колебания температуры на протяжении дня;

- астрономические данные: информация о времени восхода/заката солнца и луны помогает пользователю планировать свой день;

- сохранение погодных данных: реализована функция экспорта информации в локальный файл, что позволяет сохранить прогноз для последующего анализа.

Главный экран приложения оформлен в современном и удобном стиле, что обеспечивает интуитивно понятный интерфейс для пользователя. При запуске

					ПГУ1.090301.12.001 ПЗ	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

отображается заставка с логотипом и слоганом, после чего пользователь должен выбрать город для прогноза погоды. На рисунке 22 представлено окно загрузки приложения.

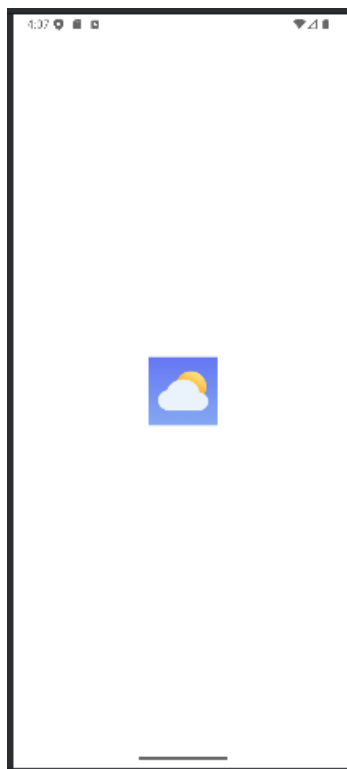


Рисунок 22 - Окно загрузки приложения

После входа в приложение пользователь может: просмотреть актуальный прогноз погоды, почасовой прогноз, изучить подробные данные о погоде, такие как температурные показатели, данные об уровне влажности, скорости и направлении ветра. Кроме того, приложение предоставляет удобную визуализацию данных в виде графиков и диаграмм, что помогает лучше понять изменения погодных условий. Также доступны астрономические сведения, такие как время восхода и захода солнца и луны.

Пользователь может в любой момент времени выбрать другой город или прекратить работу приложения. На рисунке 23 представлен главный экран приложения.



Рисунок 23 - Главный экран приложения

1.7.2 Описание работы приложения

Пользователь работает только с клиентской частью приложения. Чтобы начать пользоваться программой, необходимо запустить её на поддерживаемой версии Android, затем войти в свою учётную запись или пройти регистрацию. После авторизации на экране появится главный интерфейс приложения, предоставляя доступ ко всем основным функциям.

Все выполняемые действия пользователя изображены на диаграмме последовательности на рисунке 24.

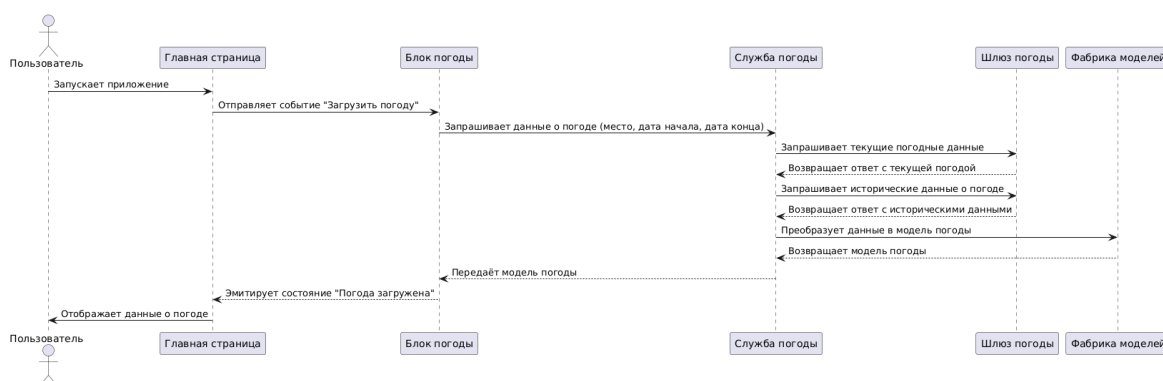


Рисунок 24 – UML-диаграмма последовательности

Изм.	Лист	№ докум.	Подпись	Дата

А физическое расположение каждого из компонентов данной системы представлено на рисунке 25.

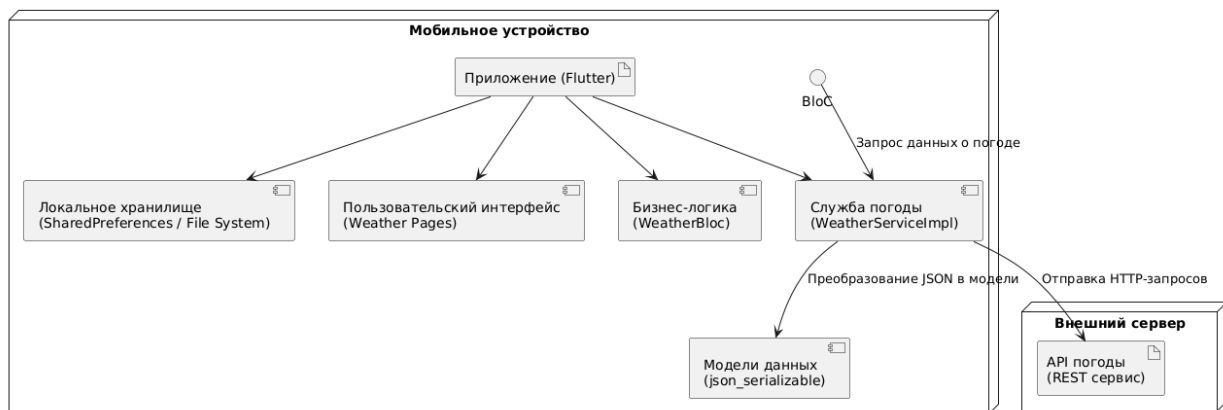


Рисунок 25 – UML-диаграмма развертывания

1.7.3 Основные компоненты приложения

Приложение для отображения погодных данных построено с использованием фреймворка Flutter и реализует продуманную архитектуру с использованием паттерна BLoC для управления состоянием. Главный экран приложения, представленный виджетом CurrentWeatherPage, служит центральным узлом, через который пользователь получает доступ к полной информации о погоде. После запуска приложения инициируется запрос к погодному API, после чего на экране отображаются не только текущие условия, но и подробные прогнозы на несколько дней.

Основной экран включает в себя реализацию горизонтального пролистывания с помощью PageView, что позволяет интуитивно просматривать прогнозы по дням. При этом каждая карточка дня отображает сводные данные, такие как средняя температура, погодное описание и иконку, демонстрирующую погодные условия. При выборе конкретного дня пользователь может перейти к более детальному просмотру, где данные разбиты по часам.

Для более наглядного отображения погодных показателей используются отдельные виджеты. Виджет WeatherHourWidget выводит почасовые данные в виде сетки, где каждая ячейка информирует о температуре, облачности, скорости ветра и других характеристиках для определённого часа. Аналогично, WeatherGraphWidget реализует графическое представление изменений

температуры в течение дня, а WeatherAstroWidget предоставляет ключевую астрономическую информацию, такую как время восхода и захода Солнца и Луны.

Дополнительный функционал приложения включает возможность сохранения выбранного города и данных о погоде посредством локального хранилища (SharedPreferences), что позволяет пользователю не вводить данные повторно при каждом запуске. Также предусмотрена функция экспорта прогноза в виде текстового файла, что может быть полезно для анализа или архивирования данных.

Таким образом, приложение объединяет все необходимые инструменты для получения, анализа и визуального представления погодных условий. Это делает его удобным решением для ежедневного использования.

1.7.4 Описание интерфейса приложения

При открытии приложения пользователем ему необходимо ввести город. При успешном вводе открывается главный экран приложения, на котором отображается актуальный прогноз погоды. На рисунке 26 представлено окно выбора города.

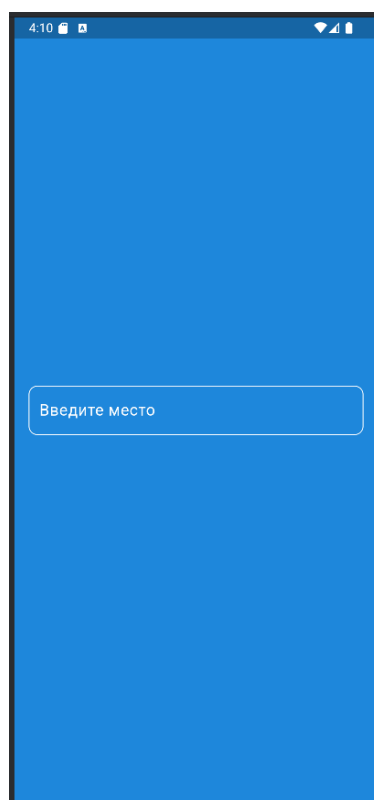


Рисунок 26 - Окно выбора города

На главном экране пользователь может посмотреть актуальную погоду, почасовую погоду, дополнительные параметры погоды, такие как: время, температура, облачность, ветер, шанс дождя, шанс снега, видимость, условия. Так же при просмотре погоды почасово можно увидеть диаграмму изменения температуры. На рисунке 27 представлен главный экран приложения.



Рисунок 27 – Главный экран приложения

При перемещении станицы вниз можно посмотреть астрономические данные: восход солнца, закат, восход луны и заход луны. В нижней части страницы находятся кнопки: «Сменить место», «Погода по текущему местоположению», «Сохранить погодные данные в файл». На рисунке 28 представлен главный экран приложения.



Рисунок 28 – Главный экран приложения

При нажатии на кнопку «Сменить место», открывается окно со списком добавленных городов. Сюда можно добавить новый город, погода сменится автоматически. На рисунке 29 показано окно смены места.



Рисунок 29 – Окно смены места

После смены места отображается погода в другом городе. На рисунке 30 представлен главный экран приложения.



Рисунок 30 – Главный экран приложения

При нажатии на кнопку «Смена места» будут показаны сохраненные города, их можно добавлять или удалять. На рисунке 31 показано окно смены места.

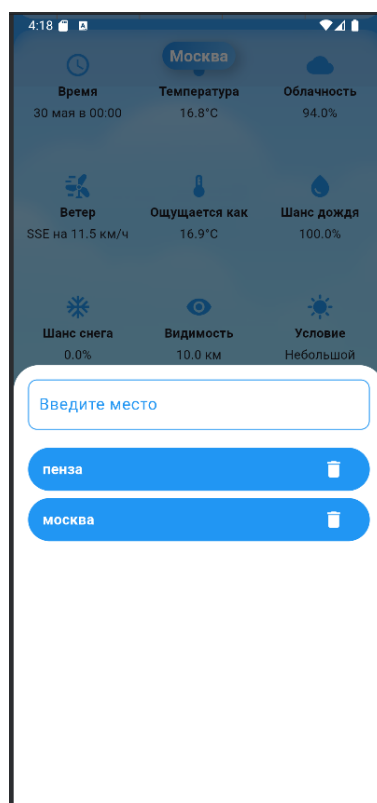


Рисунок 31 – Окно смены места

Пользователю доступна погода на 7 дней. Можно посмотреть погоду за предыдущие 3 дня, на текущий день и на последующие 3 дня. На эти дни будут так же доступны функции: почасовая погода, диаграмму изменения температуры, дополнительные параметры погоды, такие как: время, температура, облачность, ветер, шанс дождя, шанс снега, видимость, условия диаграмму изменения температуры и астрономические данные: восход солнца, закат, восход луны и заход луны. Кнопки: «Сменить место», «Погода по текущему местоположению», «Сохранить погодные данные в файл» так же будут отображаться. На рисунке 31 представлена погода предыдущего дня.



Рисунок 32 – Погода предыдущего дня

При нажатии на кнопку «Сохранить погодные данные в файл» данные будут сохранены в файл. На рисунке 33 показана работа кнопки.

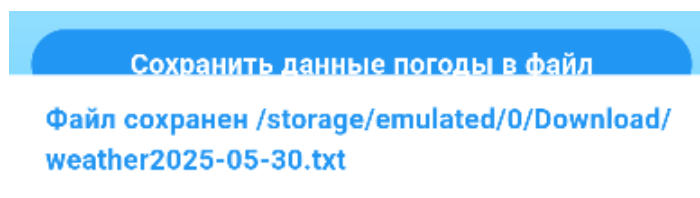


Рисунок 33 – Кнопка «Сохранить погодные данные в файл»

2 Безопасность жизнедеятельности

Использование персонального компьютера и смартфона может быть не таким безопасным, как кажется на первый взгляд. Это подтверждают данные стандарта ГОСТ 12.0.003-2015, в котором описаны основные риски, связанные с длительной работой за компьютером и использованием мобильных устройств. Для защиты пользователей и разработчиков были разработаны специальные рекомендации, несоблюдение которых может негативно сказаться на здоровье и безопасности.

2.1 Правила и техника безопасности при работе с ПК.

Согласно стандарту ГОСТ 12.0.003-2015 при работе с ПК могут возникать следующие факторы, влияющие на состояние пользователя:

- повышение температуры отдельных компонентов компьютера, что может привести к перегреву рабочей зоны;
- монотонность рабочего процесса, вызывающая усталость и снижение концентрации;
- значительная нагрузка на зрение из-за длительного использования монитора;
- риск поражения статическим электричеством при контакте с электронными компонентами устройства;
- недостаточное освещение, которое может привести к быстрой утомляемости глаз;

Высокая контрастность экрана, способная негативно сказаться на зрении при длительном использовании. Чтобы минимизировать вредное воздействие следует соблюдать несколько правил.

Необходимо организовать правильную эргономику рабочего места:

- правильная посадка. Необходимо сидеть с прямой спиной, расположить руки на уровне клавиатуры, а ступни должны полностью опираться на пол. Оптимальное расстояние до монитора — примерно на вытянутую руку, а его угол

наклона должен составлять около 30 градусов для снижения нагрузки на позвоночник и глаза;

– освещение. Рабочее место должно быть хорошо освещено, но при этом не должно быть ярким, чтобы избежать усталости глаз. Важно минимизировать появление бликов на экране.

В целях безопасности не следует использовать повреждённое оборудование. Рабочие места лучше размещать вдоль стен, чтобы снизить воздействие электромагнитного излучения.

2.2 Правила и техника безопасности при работе со смартфоном.

Так как приложение будет работать на мобильных устройствах, необходимо учитывать возможные риски при их использовании.

Основные меры предосторожности:

- не вскрывать и не модифицировать устройство самостоятельно, так как это может привести к поломке или потере гарантийного обслуживания;
- избегать механических повреждений: не ронять, не сгибать и не подвергать телефон сильным ударам;
- осторожно обращаться с водой и огнём: не допускать попадания жидкости на устройство и не оставлять его вблизи источников высокой температуры.

Соблюдение этих правил продлит срок службы устройства и сохранит здоровье пользователя, обеспечивая безопасное и комфортное использование смартфонов и компьютеров в повседневной жизни.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильное приложение «Погода».

Приложение предоставляет данные о текущей температуре, влажности, скорости ветра, а также почасовой прогноз. Кроме того, оно включает астрономические сведения, такие как время восхода и заката солнца и луны, что делает его функционал более полным и полезным.

Одним из главных преимуществ приложения является наличие удобного и интуитивно понятного для пользователя графического интерфейса. Простота использования приложения облегчит пользователям просмотр прогноза погоды.

В дальнейшем можно усовершенствовать разработанное приложение, улучшить алгоритмы прогнозирования и внедрить дополнительные визуальные элементы для более наглядного представления погодных условий. Такие доработки позволят сделать приложение ещё более востребованным и удобным.

					<i>ПГУ1.090301.12.001 ПЗ</i>	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Flutter. Официальная документация: [Электронный ресурс]. URL: <https://flutter.dev> (дата обращения: 20.04.2025).
2. Dart. Официальная документация по языку программирования: [Электронный ресурс]. URL: <https://dart.dev> (дата обращения: 25.04.2025).
3. OpenWeatherMap API. Официальная документация: [Электронный ресурс]. URL: <https://openweathermap.org/api> (дата обращения: 30.04.2025)
4. Ф.Заметти «Flutter на практике», 2020. – 300 с.
5. Р.Ричард «Flutter и Dart. Сборник рецептов: Разработка полнофункциональных облачных приложений», 2020. – 250 с.
6. Рап Пейн «Начало разработки приложений с Flutter», 2020. – 180 с.

Приложение А

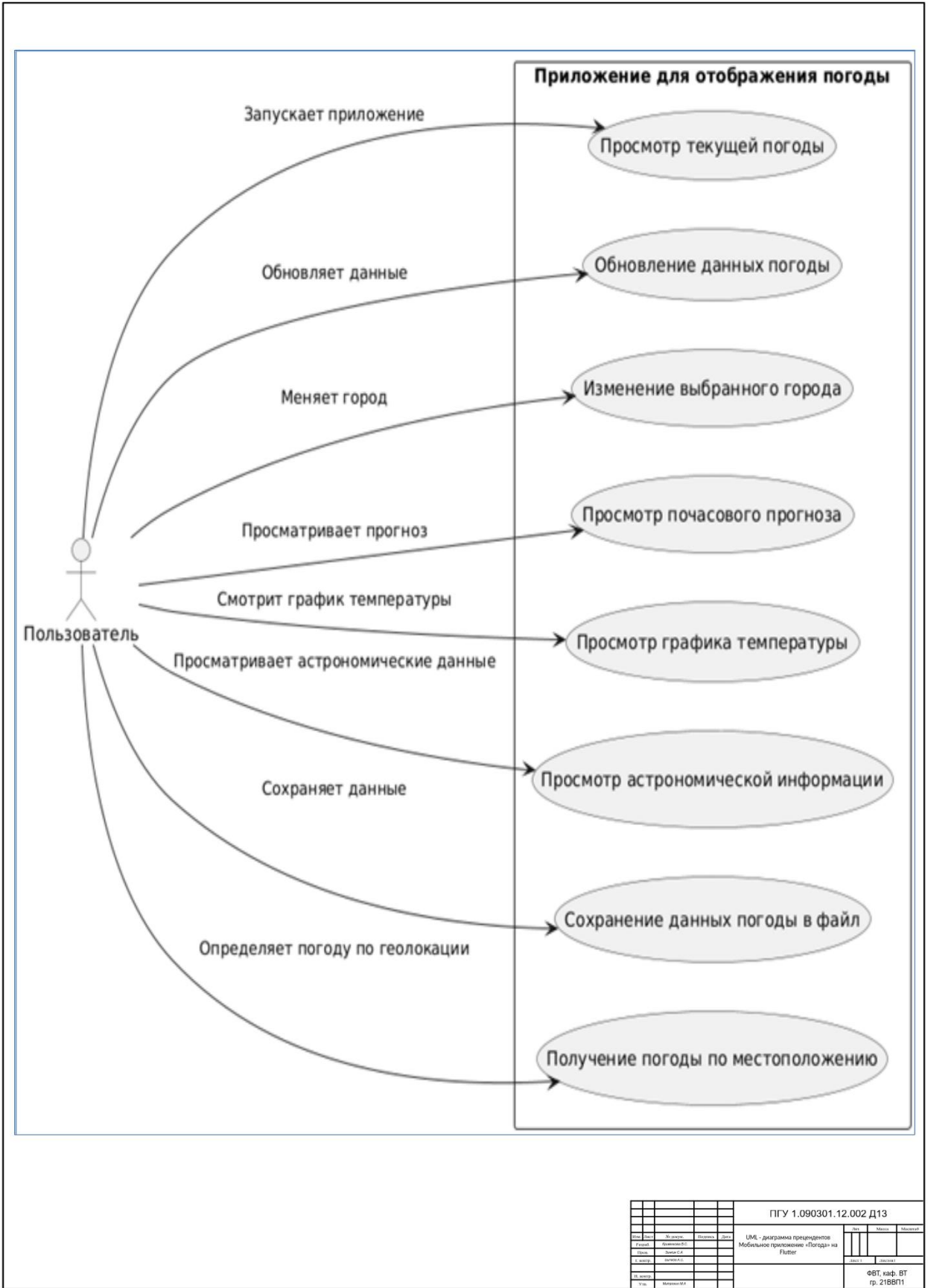
(Обязательное)

Мобильное приложение «Погода»

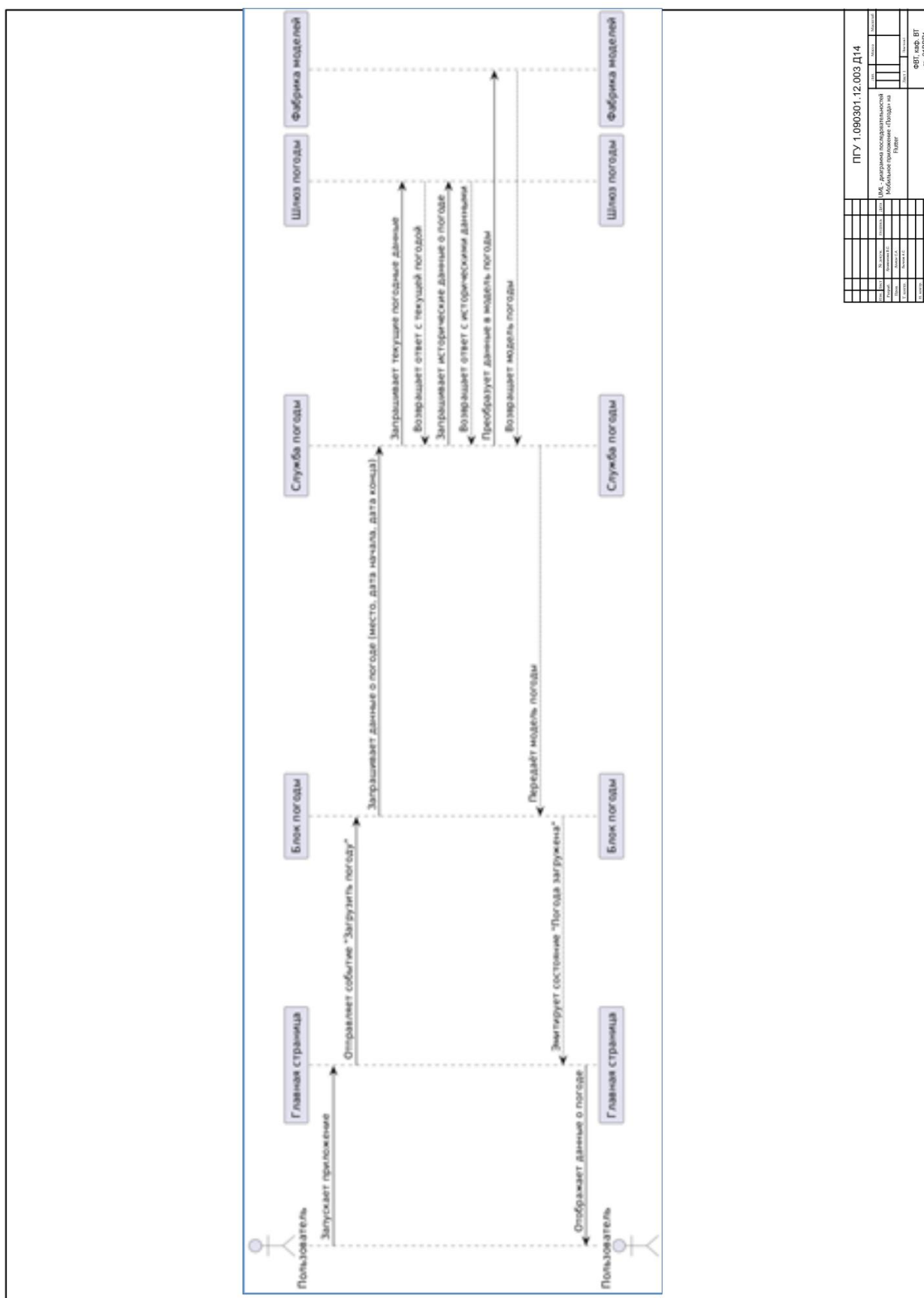
UML-диаграммы

					ПГУ1.090301.12.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML – диаграмма прецедентов



Приложение А.2 – UML – диаграмма последовательностей

[illegible]

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.12.001 ПЗ

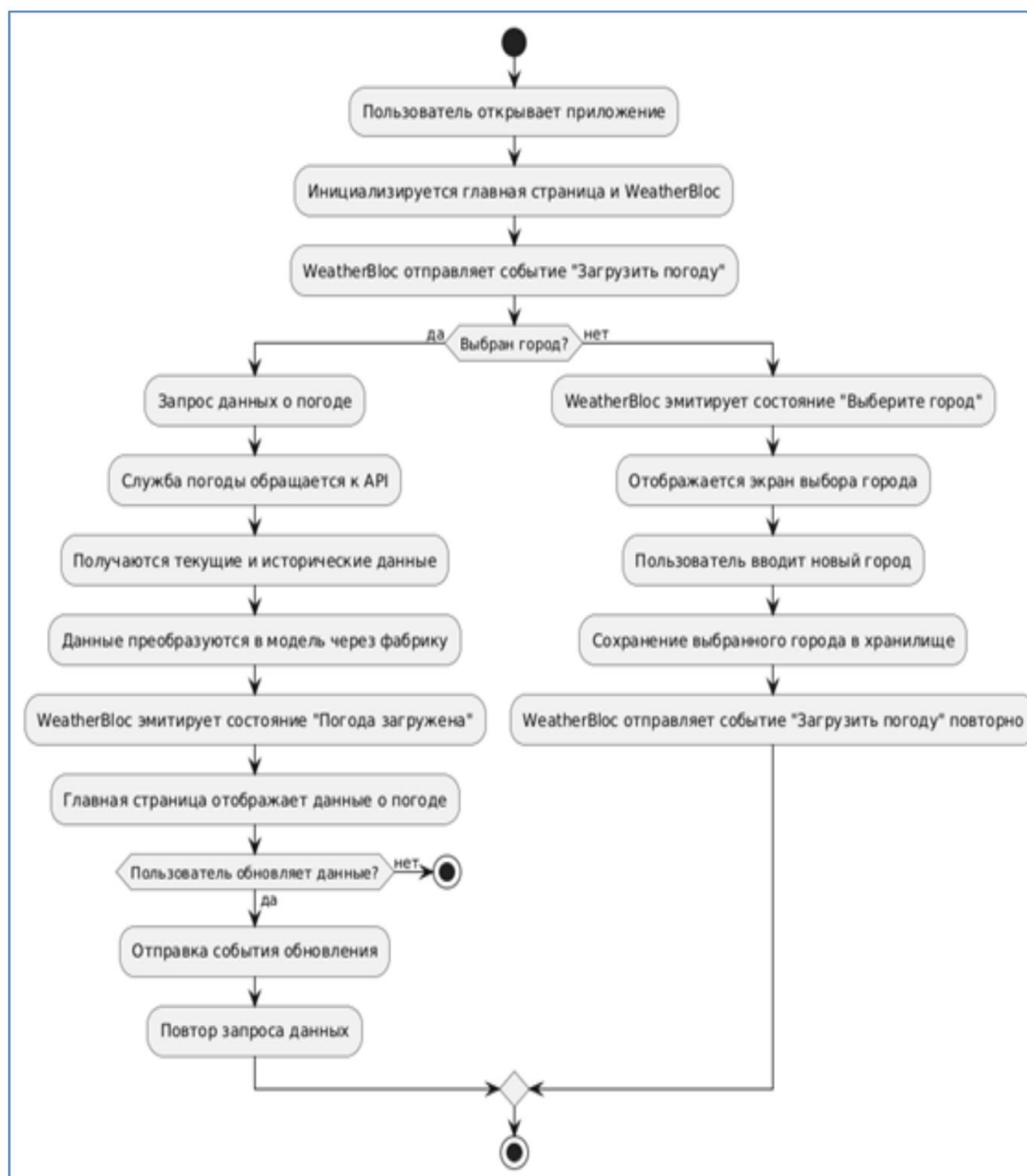
Лист

46

Приложение А.3 – UML – диаграмма классов

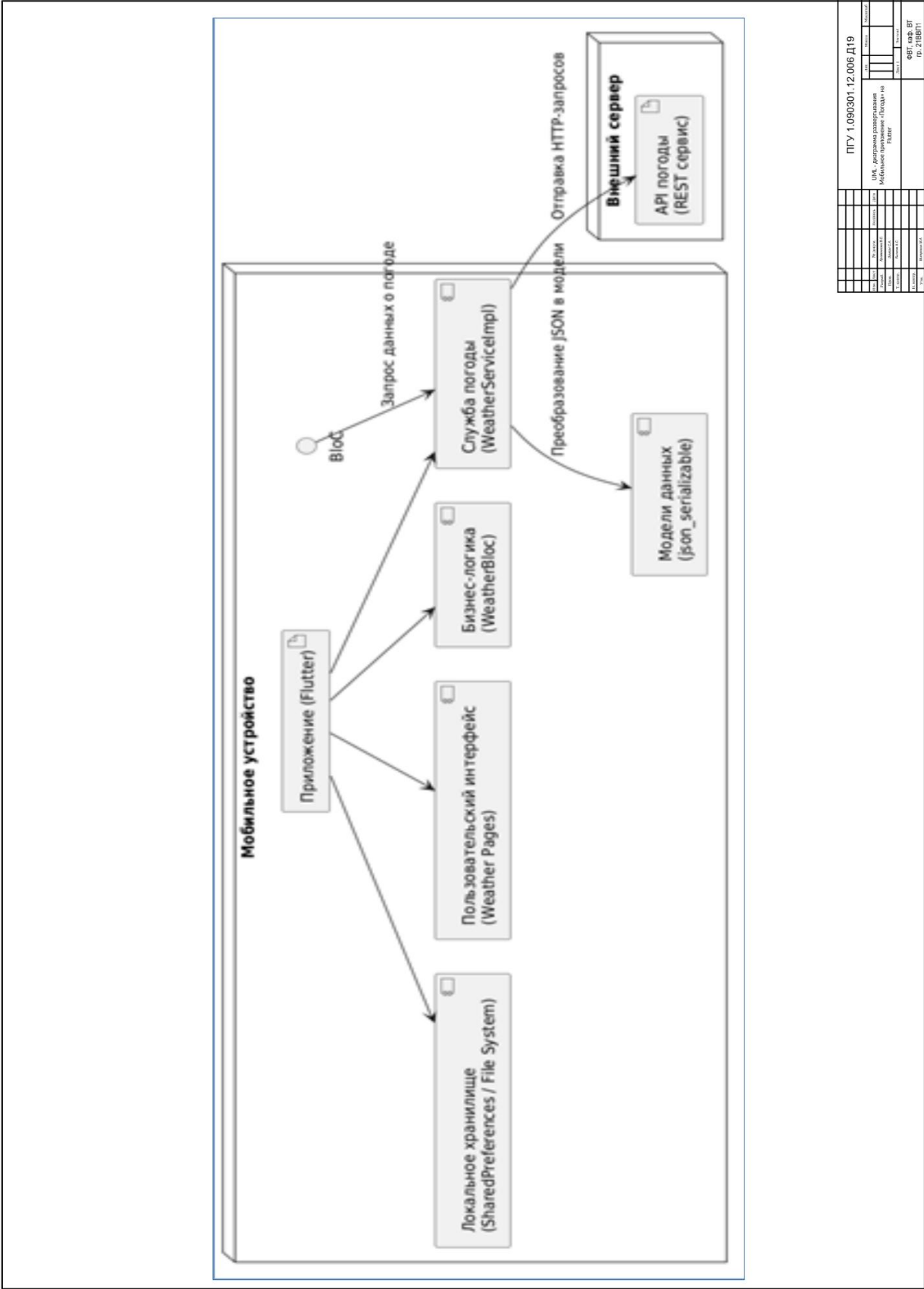
[illegible]

Приложение А.4 – UML – диаграмма деятельности



						ПГУ 1.090301.12.005 D17			
Имя	Фамилия	№ заявки	Описание	Дата	УМЛ - диаграмма деятельности Мобильное приложение «Погода» на Flutter				
Г. Иванов	Петров И.И.				Вит	Макс	Минус		
Оцен	Время С/А								
С. Иванов	Иванов И.И.				00021	0001047			
И. Иванов	Иванов И.И.				ФБТ, каф. БТ р. 218ПР1				

Приложение А.5 – UML – диаграмма развёртывания



--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Приложение Б

(Обязательное)

Мобильное приложение «Погода»

Листинг программы

					ПГУ1.090301.12.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «getIt.dart»

```
import 'dart:io';

import 'package:get_it/get_it.dart';
import
'package:weather_app/factory/impl/weather_factory_impl.dart';
import 'package:weather_app/factory/weather_factory.dart';
import
'package:weather_app/gateway/impl/weather_gateway_impl.dart';
import 'package:weather_app/gateway/weather_gateway.dart';
import 'package:weather_app/http_client.dart';
import
'package:weather_app/service/impl/weather_service_impl.dart';
import 'package:weather_app/service/weather_service.dart';
/// Функция `setupDependencies` настраивает зависимости
приложения с использованием библиотеки `GetIt`.
/// GetIt - это Service Locator, который позволяет
регистрировать и получать зависимости в любом месте приложения.
/// Это упрощает управление зависимостями и делает код более
модульным и тестируемым.
setupDependencies() async {
  // Получаем экземпляр GetIt
  final serviceLocator = GetIt.instance;

  // Регистрируем WeatherService как Lazy Singleton.
  // Lazy Singleton означает, что объект создается только при
первом обращении к нему.
  serviceLocator.registerLazySingleton<WeatherService>(
    () => WeatherServiceImpl(
      GetIt.instance<WeatherGateway>(), // Получаем зависимость
WeatherGateway
      GetIt.instance<WeatherFactory>(), // Получаем зависимость
WeatherFactory
    ),
  );

  // Регистрируем HttpClient как Lazy Singleton.
  // HttpClient будет использоваться для выполнения HTTP-
запросов.
  serviceLocator.registerLazySingleton<HttpClient>(
    () => HttpClient(),
  );

  // Регистрируем WeatherGateway как Factory.
  // Factory означает, что новый объект создается каждый раз при
запросе.
```

					ПГУ1.090301.12.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

```

    serviceLocator.registerFactory<WeatherGateway>(
        () => WeatherGatewayImpl(
            GetIt.instance<HttpClient>(), // Получаем зависимость
            HttpClient
        ),
    );

    // Регистрируем WeatherFactory как Factory.
    // WeatherFactory отвечает за создание моделей данных.
    serviceLocator.registerFactory<WeatherFactory>(
        () => WeatherFactoryImpl(),
    );
}

```

Файл «http_client.dart»

```

import 'dart:convert';

import 'package:http/http.dart' as http;
import 'package:weather_app/logger.dart';
// класс реализует рест запросы к серверу
/**
 *Здесь реализованы методы для отправки рест запросов, таких как
get post put
 * далее эти методы используются в gateway
 */
class HttpClient {
    static const String apiKey =
    'f79dbe0c3cmshffc3db0ccc081b0p1e7209jsne5380980e367';
    static const String apiKey2 =
    'f9a88eafb0msh355b62eb511cb0cple8a4djsnfa0e783e0041';

    static const String rapidApi = 'weatherapi-
com.p.rapidapi.com';
    static const Map<String, String> header = {
        'x-rapidapi-host': rapidApi,
        'x-rapidapi-key': apiKey
    };
    HttpClient();

    Future<dynamic> get(String endpoint, {Map<String, dynamic>?
params}) async {
        final uri = Uri.parse('https://weatherapi-
com.p.rapidapi.com' + endpoint).replace(queryParameters:
params,);
        logger.i('get ${uri.path}');

```

```

        final response = await http.get(uri, headers: header);
        return _handleResponse(response);
    }

    Future<dynamic> post(String endpoint, {Map<String, dynamic>?
data}) async {
        final uri = Uri.parse('https://weatherapi-
com.p.rapidapi.com' + endpoint);
        logger.i('post ${uri.path}');
        final response = await http.post(uri, body:
json.encode(data), headers: header);
        return _handleResponse(response);
    }

    Future<dynamic> put(String endpoint, {Map<String, dynamic>?
data}) async {
        final uri = Uri.parse('https://weatherapi-
com.p.rapidapi.com' + endpoint);
        logger.i('put ${uri.path}');
        final response = await http.put(uri, body:
json.encode(data), headers: header);
        return _handleResponse(response);
    }
    //обработчик ответа
    dynamic _handleResponse(http.Response response) {
        if (response.statusCode >= 200 && response.statusCode < 300)
        {
            logger.i('${response.statusCode}
${response.request?.url}');
            logger.i(jsonDecode(utf8.decode(response.bodyBytes)));
            return jsonDecode(utf8.decode(response.bodyBytes));
        } else {
            logger.e('${response.statusCode}
${response.request?.url}');
            throw Exception('Error: ${response.statusCode}
${response.reasonPhrase}');
        }
    }
}

```

Файл «logger.dart»

```

import 'package:logger/logger.dart';
/// Геттер `logger` предоставляет доступ к экземпляру логгера.
/// Это позволяет использовать логгер в любом месте приложения,
вызывая `logger`

```

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		53

```

Logger get logger => MyLogger.instance;
/// Класс `MyLogger` является кастомной реализацией логгера на
основе библиотеки `logger`.
/// Он настраивает формат и поведение логов.
class MyLogger extends Logger {
  MyLogger._() : super(
    printer: PrettyPrinter(
      colors: true,
      errorMethodCount: 8,
      printEmojis: true,
      printTime: true,
      methodCount: 0,
      noBoxingByDefault: true
    ));
  /// Статический экземпляр логгера, который создается один раз
  (Singleton).
  static final instance = MyLogger._();
}

```

Файл «main.dart»

```

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:weather_app/getIt.dart';
import 'package:weather_app/page/current_weather_page.dart';

/// Точка входа в приложение.
/// Инициализирует зависимости и запускает приложение.
Future<void> main() async {
  // Настройка зависимостей перед запуском приложения
  await setupDependencies();

  // Запуск приложения
  runApp(const MyApp());
}

/// Класс `MyApp` является корневым виджетом приложения.
/// Он настраивает тему, ориентацию экрана и определяет
начальную страницу.
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    // Устанавливаем предпочтительную ориентацию экрана (только
портретная)

```

```

SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);

return MaterialApp(
  // Убираем баннер "Debug" в углу экрана
  debugShowCheckedModeBanner: false,

  // Название приложения
  title: 'Погода',

  // Настройка темы приложения
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple), // Цветовая схема
    useMaterial3: true, // Использование Material 3
  ),

  // Начальная страница приложения
  home: CurrentWeatherPage(),
);
}
}

```

Файл «utils.dart»

```

import 'package:intl/intl.dart';

/// Преобразует число типа double в строку и обратно в double.
/// Это может быть полезно для устранения проблем с точностью
чисел с плавающей точкой.
double toDouble(double number) {
  return double.parse(number.toString());
}

/// Преобразует время из 12-часового формата (AM/PM) в 24-
часовой формат.
/// Например, "02:30 PM" -> "14:30".
String convertToRussianTimeFormat(String time) {
  // Регулярное выражение для поиска времени в формате "HH:MM
AM/PM"
  final format = RegExp(r'(\d{1,2}):(\d{2})\s([APM]{2})');
  final match = format.firstMatch(time);

  if (match != null) {
    final hour = int.parse(match.group(1)!); // Часы
    final minute = match.group(2)!; // Минуты
    final period = match.group(3)!; // Период (AM/PM)

```

					ПГУ1.090301.12.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

```

int convertedHour = hour;

if (period == 'AM') {
    // Если AM, то 12:00 AM это 00:00 в 24-часовом формате
    if (hour == 12) {
        convertedHour = 0;
    }
} else if (period == 'PM') {
    // Если PM, то 12:00 PM остается 12:00, остальные часы
увеличиваются на 12
    if (hour != 12) {
        convertedHour += 12;
    }
}

return '$convertedHour:$minute'; // Возвращаем время в 24-
часовом формате
}

return time; // Если формат не совпадает, возвращаем исходное
значение
}

/// Форматирует дату в формате "день месяц".
/// Например, "2023-10-05" -> "5 октября".
String formatMonthDay(String inputDate) {
    // Преобразуем строку в DateTime
    DateTime date = DateTime.parse(inputDate);

    // Массив с названиями месяцев на русском языке
    List<String> months = [
        'января', 'февраля', 'марта', 'апреля', 'мая', 'июня',
        'июля', 'августа', 'сентября', 'октября', 'ноября',
        'декабря'
    ];

    // Форматируем дату для отображения только месяца и дня
    String formattedDate = '${date.day} ${months[date.month -
1]}';

    return formattedDate;
}

/// Форматирует дату и время в формате "день месяц в HH:MM".
/// Например, "2023-10-05T14:30:00" -> "5 октября в 14:30".
String formatDateDateTime(String inputDate) {

```

```

// Преобразуем строку в DateTime
DateTime date = DateTime.parse(inputDate);

// Массив с названиями месяцев на русском языке
List<String> months = [
    'января', 'февраля', 'марта', 'апреля', 'мая', 'июня',
    'июля', 'августа', 'сентября', 'октября', 'ноября',
    'декабря'
];

// Форматируем дату вручную
String formattedDate = '${date.day} ${months[date.month - 1]}
в ${date.hour.toString().padLeft(2,
'0')}:${date.minute.toString().padLeft(2, '0')}';

return formattedDate;
}

/// Форматирует дату в зависимости от того, относится ли она к
сегодняшнему дню, вчерашнему или завтрашнему.
/// Например, "2023-10-05" -> "Сегодня", если сегодня 5 октября
2023 года.
String formatDate(String dateStr) {
    final date = DateTime.parse(dateStr); // Преобразуем строку в
DateTime
    final now = DateTime.now(); // Текущая дата и время
    final today = DateTime(now.year, now.month, now.day); //
Сегодняшняя дата без времени
    final yesterday = today.subtract(Duration(days: 1)); //
Вчерашняя дата
    final tomorrow = today.add(Duration(days: 1)); // Завтрашняя
дата

    // Форматируем дату в формате "дд.ММ.гггг"
    final formattedDate = DateFormat('dd.MM.yyyy').format(date);

    // Проверяем, относится ли дата к вчерашнему дню
    if (date.year == yesterday.year &&
        date.month == yesterday.month &&
        date.day == yesterday.day) {
        return 'Вчера';
    }
    // Проверяем, относится ли дата к сегодняшнему дню
    else if (date.year == today.year &&
        date.month == today.month &&
        date.day == today.day) {
        return 'Сегодня';
    }
}

```

```

    }
    // Проверяем, относится ли дата к завтрашнему дню
    else if (date.year == tomorrow.year &&
        date.month == tomorrow.month &&
        date.day == tomorrow.day) {
        return 'Завтра';
    }
    // Если дата не относится к вчера, сегодня или завтра,
    возвращаем отформатированную дату
    else {
        return formattedDate;
    }
}

```

Файл «weather_bloc.dart»

```

// Package imports:

import 'dart:io';

import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:collection/collection.dart';
import 'package:geolocator/geolocator.dart';
import 'package:intl/intl.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:weather_app/bloc/weather_event.dart';
import 'package:weather_app/bloc/weather_state.dart';
import 'package:weather_app/service/weather_service.dart';
/**
 * На флаттере в основном используют блок он помогает разделить
 * основную логику приложения от интерфейса,
 * в интерфейсе вызываем определенный эвент (событие) например
LoadCurrentWeatherEvent
 * это событие инициирует определенную логику в ходе которой мы
эмитим в интерфейс определенные состояния например
 * LoadingWeatherState, LoadingWeatherState это объект, он может
хранить в себе определенные данные
 * через объекты такие как LoadCurrentWeatherEvent мы передаем
данные от интерфейса к обратчику
 */

class WeatherBloc extends Bloc<WeatherEvent, WeatherState> {

    final WeatherService _weatherService;

```

```

WeatherBloc(this._weatherService)
    : super(InitialState()) {
    on<LoadCurrentWeatherEvent>((event, emit) =>
        _loadCurrentWeather(event, emit));
    on<LoadGeoWeatherEvent>((event, emit) =>
        _loadCurrentGeoWeather(event, emit));
}
late LoadedWeatherState _loadedWeatherState;

///обработчик события загрузки погоды
_loadCurrentWeather(LoadCurrentWeatherEvent event,
    Emitter<WeatherState> emit) async {
    try {
        //эмитим в интерфейс состояние загрузки
        emit>LoadingWeatherState());

        //проверяем сохраненный город
        final prefs = await SharedPreferences.getInstance();
        final selectedCity = prefs.getString('selectedCity');

        //если есть сохраненный город
        if(selectedCity!=null){
            DateTime currentDate = DateTime.now();

            // Вычитаем 3 дня из текущей даты
            // так как сервер не хочет давать историю погоды ранее
            чем 3 назад
            DateTime threeDaysAgo =
currentDate.subtract(Duration(days: 3));
            DateTime oneDaysAgo = currentDate.subtract(Duration(days:
1));

            // Форматируем дату в формат yyyy-MM-dd
            String dt = DateFormat('yyyy-MM-dd').format(threeDaysAgo);
            String endDt = DateFormat('yyyy-MM-
dd').format(oneDaysAgo);

            //получаем погоду
            final currentWeather = await
_weatherService.getCurrentWeather(place: selectedCity,timeDt:
dt,endDt: endDt);
            _loadedWeatherState = LoadedWeatherState(currentWeather:
currentWeather);

            //эмитим модель погоды в интерфейс
            emit(LoadedWeatherState(currentWeather: currentWeather));
        }else{

```

```

        //эмитим состояние выбора города, когда открыли
приложения первый раз например
        emit(SelectedPlaceState());
    }
} catch (e){
    emit(ErrorState(message: e.toString()));
}
}

// обработчик загрузки погоды через координаты, с помощью
геолокации
_loadCurrentGeoWeather(LoadGeoWeatherEvent event,
    Emitter<WeatherState> emit) async {
    try {
        emit>LoadingWeatherState());
        bool serviceEnabled;
        bool isError = false;

        LocationPermission permission;

        // Проверяем, включена ли служба местоположения
        serviceEnabled = await
Geolocator.isLocationServiceEnabled();
        if (!serviceEnabled) {
            emit(ErrorToastState(message: 'Разрешение не
представлено'));
            emit(_loadedWeatherState);
            isError = true;
        }

        // Запрашиваем разрешение на доступ к местоположению
        permission = await Geolocator.checkPermission();
        if (permission == LocationPermission.denied) {
            permission = await Geolocator.requestPermission();
            if (permission == LocationPermission.denied) {
                emit(ErrorToastState(message: 'Разрешение не
представлено'));
                emit(_loadedWeatherState);
                isError = true;
            }
        }

        if (permission == LocationPermission.deniedForever) {
            emit(ErrorToastState(message: 'Разрешение не
представлено'));
            emit(_loadedWeatherState);
        }
    }
}

```

```

        isError = true;
    }
    Position? position = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.high,
    );

    if(!isError){
        DateTime currentDate = DateTime.now();
        emit>LoadingWeatherState();
        // Вычитаем 3 дня из текущей даты
        DateTime threeDaysAgo = currentDate.subtract(Duration(days:
3));
        DateTime oneDaysAgo = currentDate.subtract(Duration(days:
1));

        // Форматируем дату в формат yyyy-MM-dd
        String dt = DateFormat('yyyy-MM-dd').format(threeDaysAgo);
        String endDt = DateFormat('yyyy-MM-dd').format(oneDaysAgo);
        final currentWeather = await
_weatherService.getCurrentWeather(place:
'${position.latitude},${position.longitude}',timeDt: dt,endDt:
endDt);
        emit(LoadedWeatherState(currentWeather: currentWeather));
    }else{
        emit(_loadedWeatherState);
    }
}catch (e){
    emit(ErrorState(message: e.toString()));
    emit(_loadedWeatherState);
}
}
}

```

Файл «weather_event.dart»

```

import 'package:equatable/equatable.dart';
//абстрактный класс для событий
abstract class WeatherEvent extends Equatable {}

//событие загрузки погоды
class LoadCurrentWeatherEvent extends WeatherEvent {

    LoadCurrentWeatherEvent();
    @override
    List<Object> get props => [];
}

```

```
//событие загрузки погоды через геолокацию
class LoadGeoWeatherEvent extends WeatherEvent {

  LoadGeoWeatherEvent();
  @override
  List<Object> get props => [];
}
```

Файл «weather_state.dart»

```
import 'package:equatable/equatable.dart';
import 'package:weather_app/model/current_weather_model.dart';

/// Абстрактный класс `WeatherState` представляет базовое
состояние для управления погодными данными.
/// Он расширяет `Equatable`, чтобы упростить сравнение объектов
состояний.
abstract class WeatherState extends Equatable {}

/// Класс `InitialState` представляет начальное состояние
приложения.
/// Это состояние используется, когда приложение только
запускается и данные еще не загружены.
class InitialState extends WeatherState {
  InitialState();

  @override
  List<Object> get props => [];
}

/// Класс `LoadingWeatherState` представляет состояние загрузки
данных о погоде.
/// Это состояние используется, когда данные запрашиваются, но
еще не получены.
class LoadingWeatherState extends WeatherState {
  LoadingWeatherState();

  @override
  List<Object> get props => [];
}

/// Класс `LoadedWeatherState` представляет состояние, когда
данные о погоде успешно загружены.
/// Он содержит объект `CurrentWeatherModel`, который хранит
информацию о текущей погоде.
class LoadedWeatherState extends WeatherState {
```

					ПГУ1.090301.12.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

```

    final CurrentWeatherModel currentWeather; // Модель данных о
текущей погоде
    LoadedWeatherState({required this.currentWeather});

    @override
    List<Object> get props => [currentWeather];
}

/// Класс `SelectedPlaceState` представляет состояние, когда
пользователь должен выбрать место.
/// Это состояние используется, если город для отображения
погоды не выбран.
class SelectedPlaceState extends WeatherState {
    SelectedPlaceState();

    @override
    List<Object> get props => [];
}

/// Класс `ErrorState` представляет состояние ошибки.
/// Он содержит сообщение об ошибке, которое может быть
отображено пользователю.
class ErrorState extends WeatherState {
    final String message; // Сообщение об ошибке
    ErrorState({required this.message});

    @override
    List<Object> get props => [message];
}

/// Класс `ErrorToastState` представляет состояние ошибки,
которое отображается в виде всплывающего уведомления (Toast).
/// Он также содержит сообщение об ошибке.
class ErrorToastState extends WeatherState {
    final String message; // Сообщение об ошибке
    ErrorToastState({required this.message});

    @override
    List<Object> get props => [message];
}

```

Файл «weather_factory.dart»

```

import 'package:weather_app/model/current_weather_model.dart';
import
'package:weather_app/model/response_model/weather_response_model

```

					ПГУ1.090301.12.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

```
.dart';
/// Абстрактный класс `WeatherFactory` определяет контракт для
фабрики, которая создает объект `CurrentWeatherModel`.
/// Этот класс используется для преобразования данных,
полученных из API, в модель, которая используется в приложении.
abstract class WeatherFactory {
  /// Метод `getCurrentWeather` принимает два параметра:
  /// - `responseCurrent`: Данные о текущей погоде, полученные
из API.
  /// - `responseHistory`: Исторические данные о погоде.
  /// Возвращает объект `CurrentWeatherModel`, который содержит
объединенные данные о текущей и исторической погоде.
  Future<CurrentWeatherModel> getCurrentWeather(
    WeatherResponse responseCurrent, WeatherHistoryResponse?
responseHistory);
}
```

Файл «weather_factory_impl.dart»

```
import 'package:weather_app/factory/weather_factory.dart';
import 'package:weather_app/model/current_weather_model.dart';
import
'package:weather_app/model/response_model/weather_response_model
.dart';
import 'package:collection/collection.dart';

/// Метод `getCurrentWeather` в классе `WeatherFactoryImpl`
отвечает за преобразование данных, полученных из API, в модель
`CurrentWeatherModel`, которая используется в приложении.
/// Он принимает два параметра: `responseCurrent` (данные о
текущей погоде) и `responseHistory`.
/// Метод парсит данные из API, создает список
`WeatherItemModel`, который включает прогнозы на несколько дней,
почасовые данные и астрономическую информацию.
/// Для исторических данных (`responseHistory`) и текущего
прогноза (`responseCurrent`) формируются отдельные элементы
списка, после чего они объединяются.
/// В результате создается объект `CurrentWeatherModel`,
содержащий всю необходимую информацию о погоде, включая текущую
температуру, ощущаемую температуру, индекс текущего дня и список
прогнозов.
/// Этот метод обеспечивает удобное преобразование сырых данных
API в структурированную модель, готовую для использования в
интерфейсе приложения.
class WeatherFactoryImpl implements WeatherFactory {
  @override
  Future<CurrentWeatherModel> getCurrentWeather(WeatherResponse
```

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

responseCurrent,
    WeatherHistoryResponse? responseHistory) async {
    List<WeatherItemModel> weatherItems = [];
    if(responseHistory!=null) {
        responseHistory!.forecast.forecastDay.map((e) {
            weatherItems.add(WeatherItemModel(date: e.date,
                weatherDays: WeatherDayModel(
                    maxTempC: e.day.maxTempC,
                    minTempC: e.day.minTempC,
                    avgTempC: e.day.avgTempC,
                    text: e.day.condition.text,
                    icon: e.day.condition.icon,
                ),
                astro: Astro(sunrise: e.astro.sunrise,
                    sunset: e.astro.sunset,
                    moonrise: e.astro.moonrise,
                    moonset: e.astro.moonset,
                    moonPhase: e.astro.moonPhase,
                    moonIllumination: e.astro.moonIllumination),
                hours: e.hour.map((element) =>
                    HourModel(time: element.time,
                        tempC: element.tempC,
                        text: element.condition.text,
                        icon: element.condition.icon,
                        windDir: element.windDir,
                        cloud: element.cloud.toDouble(),
                        feelsLikeC: element.feelslikeC,
                        gustKph: element.gustKph,
                        chanceOfRain: element.chanceOfRain.toDouble(),
                        chanceOfSnow: element.chanceOfSnow.toDouble(),
                        visKm: element.visKm)).toList(),
                isNowDay: false,)
            );
        }).toList();
    }
    int indexNowDay = weatherItems.length;
    responseCurrent.forecast.forecastDay.mapIndexed((i,e) {
        weatherItems.add(WeatherItemModel(date: e.date,
            weatherDays: WeatherDayModel(
                maxTempC: e.day.maxTempC,
                minTempC: e.day.minTempC,
                avgTempC: e.day.avgTempC,
                text: e.day.condition.text,
                icon: e.day.condition.icon,
            ),
            astro: Astro(sunrise: e.astro.sunrise,
                sunset: e.astro.sunset,

```

```

        moonrise: e.astro.moonrise,
        moonset: e.astro.moonset,
        moonPhase: e.astro.moonPhase,
        moonIllumination: e.astro.moonIllumination),
    hours: e.hour.map((element)=>HourModel(time:
element.time,
    tempC: element.tempC,
    text: element.condition.text,
    icon: element.condition.icon,
    windDir: element.windDir,
    cloud: element.cloud.toDouble(),
    feelsLikeC: element.feelslikeC,
    gustKph: element.gustKph,
    chanceOfRain: element.chanceOfRain.toDouble(),
    chanceOfSnow: element.chanceOfSnow.toDouble(),
    visKm: element.visKm)).toList(),
    isNowDay: i==0,)
  );
}).toList();

CurrentWeatherModel currentWeatherModel =
CurrentWeatherModel(nameCity: responseCurrent.location.name,
    nameRegion: responseCurrent.location.region,
    feelsLikeC: responseCurrent.current.feelslikeC,
    nowTempC: responseCurrent.current.tempC,
    indexNowDay: indexNowDay,
    weatherItems: weatherItems);

return currentWeatherModel;
}
}

```

Файл «weather_gateway.dart»

```

import
'package:weather_app/model/response_model/weather_response_model
.dart';

/// Абстрактный класс `WeatherGateway` определяет контракт для
взаимодействия с API погоды.
/// Этот интерфейс описывает методы, которые должны быть
реализованы в классах, работающих с API.
abstract class WeatherGateway {
  /// Метод `getCurrentWeather` получает текущую погоду для
указанного города.
  /// Параметры:
  /// - `city`: Название города, для которого запрашивается

```

										Лист
										66
Изм.	Лист	№ докум.	Подпись	Дата						

ПГУ1.090301.12.001 ПЗ

погода.

/// Возвращает объект `WeatherResponse`, содержащий данные о текущей погоде.

```
Future<WeatherResponse> getCurrentWeather(String city);
```

/// Метод `getHistoryWeather` получает исторические данные о погоде для указанного города и периода.

/// Параметры:

/// - `city`: Название города, для которого запрашивается история погоды.

/// - `timeDt`: Начальная дата периода.

/// - `endDt`: Конечная дата периода.

/// Возвращает объект `WeatherHistoryResponse`, содержащий исторические данные о погоде.

```
Future<WeatherHistoryResponse> getHistoryWeather(
```

```
String city, {
```

```
  required String timeDt,
```

```
  required String endDt,
```

```
});
```

```
}
```

Файл «weather_gateway_impl.dart»

```
import 'package:weather_app/gateway/weather_gateway.dart';
```

```
import 'package:weather_app/http_client.dart';
```

```
import
```

```
'package:weather_app/model/response_model/weather_response_model.dart';
```

/// Класс `WeatherGatewayImpl` реализует интерфейс `WeatherGateway`.

/// Этот класс отвечает за взаимодействие с API погоды и получение данных.

```
class WeatherGatewayImpl implements WeatherGateway {
```

```
  final HttpClient _httpClient; // Клиент для выполнения HTTP-запросов
```

/// Конструктор принимает экземпляр `HttpClient` для выполнения запросов.

```
WeatherGatewayImpl(this._httpClient);
```

/// Метод `getCurrentWeather` получает текущую погоду для указанного города.

/// Параметры:

/// - `city`: Название города, для которого запрашивается погода.

/// Возвращает объект `WeatherResponse`, содержащий данные о

					ПГУ1.090301.12.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

текущей погоде.

```
@override
Future<WeatherResponse> getCurrentWeather(String city) async {
    // Параметры запроса
    Map<String, dynamic> params = {
        'q': city, // Город
        'days': '3', // Количество дней для прогноза
        'lang': 'ru', // Язык ответа (русский)
    };

    // Выполнение GET-запроса к API
    final response = await _httpClient.get('/forecast.json',
        params: params);

    // Преобразование JSON-ответа в объект `WeatherResponse`
    return WeatherResponse.fromJson(response);
}

/// Метод `getHistoryWeather` получает исторические данные о
погоде для указанного города и периода.
/// Параметры:
/// - `city`: Название города, для которого запрашивается
история погоды.
/// - `timeDt`: Начальная дата периода.
/// - `endDt`: Конечная дата периода.
/// Возвращает объект `WeatherHistoryResponse`, содержащий
исторические данные о погоде.
@override
Future<WeatherHistoryResponse> getHistoryWeather(
    String city, {
        required String timeDt,
        required String endDt,
    }) async {
    // Параметры запроса
    Map<String, dynamic> params = {
        'q': city, // Город
        'lang': 'ru', // Язык ответа (русский)
        'dt': timeDt, // Начальная дата
        'end_dt': endDt, // Конечная дата
    };

    // Выполнение GET-запроса к API
    final response = await _httpClient.get('/history.json',
        params: params);

    // Преобразование JSON-ответа в объект
    `WeatherHistoryResponse`
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        return WeatherHistoryResponse.fromJson(response);
    }
}

```

Файл «current_weather_model.dart»

```

import 'package:equatable/equatable.dart';
// Модели данных для работы с погодой:
// - `CurrentWeatherModel`: Основная модель, содержащая
информацию о текущей погоде в городе, включая название города,
региона, текущую температуру, ощущаемую температуру, индекс
текущего дня и список прогнозов на несколько дней
(`WeatherItemModel`).
// - `WeatherItemModel`: Модель, представляющая данные о погоде
на конкретный день, включая дату, информацию о погоде за день
(`WeatherDayModel`), астрономические данные (`Astro`) и
почасовой прогноз (`HourModel`).
// - `WeatherDayModel`: Модель, содержащая данные о погоде за
день, такие как максимальная, минимальная и средняя температура,
описание погоды и иконка.
// - `HourModel`: Модель, представляющая почасовой прогноз,
включая время, температуру, описание погоды, иконку, направление
ветра, облачность, ощущаемую температуру, скорость ветра,
вероятность дождя и снега, а также видимость.
// - `Astro`: Модель, содержащая астрономические данные, такие
как время восхода и заката солнца, время восхода и захода луны,
фаза луны и освещенность луны. Все модели расширяют `Equatable`
для упрощения сравнения объектов.
class CurrentWeatherModel extends Equatable {
    final String nameCity;
    final String nameRegion;
    final double feelsLikeC;
    final double nowTempC;
    final int indexNowDay;
    final List<WeatherItemModel> weatherItems; // Список прогнозов
на несколько дней

    CurrentWeatherModel({
        required this.nameCity,
        required this.nameRegion,
        required this.feelsLikeC,
        required this.nowTempC,
        required this.indexNowDay,
        required this.weatherItems,
    });
}

```

```

@override
List<Object> get props => [
    nameCity,
    nameRegion,
    feelsLikeC,
    nowTempC,
    indexNowDay,
    weatherItems,
];
}

class WeatherItemModel extends Equatable {
    final String date;
    final WeatherDayModel weatherDays;
    final Astro astro;
    final List<HourModel> hours; // Список почасовых данных
    final bool isNowDay;
    WeatherItemModel({
        required this.date,
        required this.weatherDays,
        required this.hours,
        required this.isNowDay,
        required this.astro
    });

    @override
    List<Object> get props => [
        date, weatherDays, hours, astro
    ];

    Map<String, dynamic> toJson() => {
        'date': date,
        'weatherDays': weatherDays.toJson(),
        'astro': astro.toJson(),
        'hours': hours.map((h) => h.toJson()).toList(),
        'isNowDay': isNowDay,
    };
}

class WeatherDayModel extends Equatable {
    final double maxTempC;
    final double minTempC;
    final double avgTempC;
    final String text;
    final String icon;
    WeatherDayModel({
        required this.maxTempC,
        required this.minTempC,

```

```

        required this.avgTempC,
        required this.text,
        required this.icon,
    });

    @override
    List<Object> get props => [
        maxTempC,
        minTempC,
        avgTempC,
        text,
        icon,
    ];
    Map<String, dynamic> toJson() => {
        'maxTempC': maxTempC,
        'minTempC': minTempC,
        'avgTempC': avgTempC,
        'text': text,
        'icon': icon,
    };
}

class HourModel extends Equatable {
    final String time;
    final double tempC;
    final String text;
    final String icon;
    final String windDir;
    final double cloud;
    final double feelsLikeC;
    final double gustKph; // скорость ветра
    final double chanceOfRain; // вероятность дождя
    final double chanceOfSnow; // вероятность снега
    final double visKm; // видимость в км

    HourModel({
        required this.time,
        required this.tempC,
        required this.text,
        required this.icon,
        required this.windDir,
        required this.cloud,
        required this.feelsLikeC,
        required this.gustKph,
        required this.chanceOfRain,
        required this.chanceOfSnow,
        required this.visKm,
    });
}

```

```

@Override
List<Object> get props => [
    time,
    tempC,
    text,
    icon,
    windDir,
    cloud,
    feelsLikeC,
    gustKph,
    chanceOfRain,
    chanceOfSnow,
    visKm,
];
Map<String, dynamic> toJson() => {
    'time': time,
    'tempC': tempC,
    'text': text,
    'icon': icon,
    'windDir': windDir,
    'cloud': cloud,
    'feelsLikeC': feelsLikeC,
    'gustKph': gustKph,
    'chanceOfRain': chanceOfRain,
    'chanceOfSnow': chanceOfSnow,
    'visKm': visKm,
};
}
class Astro extends Equatable{
    final String sunrise;
    final String sunset;
    final String moonrise;
    final String moonset;
    final String moonPhase;
    final int moonIllumination;

    Astro({
        required this.sunrise,
        required this.sunset,
        required this.moonrise,
        required this.moonset,
        required this.moonPhase,
        required this.moonIllumination,
    });
}

```

```
List<Object?> get props => [sunrise,sunset,moonrise, moonset,
moonPhase, moonIllumination];
```

```
Map<String, dynamic> toJson() => {
  'sunrise': sunrise,
  'sunset': sunset,
  'moonrise': moonrise,
  'moonset': moonset,
  'moonPhase': moonPhase,
  'moonIllumination': moonIllumination,
};
}
```

Файл «weather_response_model.dart»

```
import 'package:json_annotation/json_annotation.dart';
import 'package:weather_app/utils.dart';
part 'weather_response_model.g.dart';
/**
```

В этом классе находятся модели данных присылаемых по апи
здесь используем библиотеку генерации моделей, чтобы вручную
не писать однотипные конструкции маппинга
просто описываем json перечисляя необходимые поля, здесь
представлены все поля запросов,
не нужные закомментированы
команда для генерации моделей есть в readme.md
**/

/// Модели данных, представленные в этом файле, используются для
сохранения и обработки данных, полученных из API погоды.
/// Они описывают структуру ответов API, включая информацию о
местоположении, текущей погоде, прогнозе на несколько дней,
почасовых данных и астрономических явлениях.
/// Для автоматической генерации кода сериализации и
десериализации JSON используется пакет `json\_serializable`
(версия ^6.8.0) в сочетании с `build\_runner` (версия ^2.4.9).
/// Это позволяет избежать ручного написания boilerplate-кода
для преобразования JSON в объекты Dart и обратно.
/// Команда `build\_runner` генерирует методы `fromJson` и
`toJson` на основе аннотаций `@JsonSerializable`, что
значительно упрощает работу с API и уменьшает вероятность
ошибок.

```
@JsonSerializable()
class LocationResponse {
  final String name;
  final String region;
  final String country;
  final double lat;
```

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

```

    final double lon;
    @JsonKey(name: 'tz_id')
    final String tzId;
    @JsonKey(name: 'localtime_epoch')
    final int localtimeEpoch;
    final String localtime;

    LocationResponse({
        required this.name,
        required this.region,
        required this.country,
        required this.lat,
        required this.lon,
        required this.tzId,
        required this.localtimeEpoch,
        required this.localtime,
    });

    factory LocationResponse.fromJson(Map<String, dynamic> json)
    => _$LocationResponseFromJson(json);
    Map<String, dynamic> toJson() =>
    _$LocationResponseToJson(this);
}

@JsonSerializable()
class ConditionResponse {
    final String text;
    final String icon;
    final int code;

    ConditionResponse({
        required this.text,
        required this.icon,
        required this.code,
    });

    factory ConditionResponse.fromJson(Map<String, dynamic> json)
    => _$ConditionResponseFromJson(json);
    Map<String, dynamic> toJson() =>
    _$ConditionResponseToJson(this);
}

@JsonSerializable()
class CurrentWeatherResponse {
    @JsonKey(name: 'last_updated_epoch')
    final int lastUpdatedEpoch;
    @JsonKey(name: 'last_updated')

```

```

final String lastUpdated;
@JsonKey(name: 'temp_c')
final double tempC;
@JsonKey(name: 'temp_f')
final double tempF;
@JsonKey(name: 'is_day')
final int isDay;
final ConditionResponse condition;
@JsonKey(name: 'wind_kph')
final double windKph;
@JsonKey(name: 'wind_dir')
final String windDir;
@JsonKey(name: 'pressure_mb')
final double pressureMb;
@JsonKey(name: 'humidity')
final int humidity;
@JsonKey(name: 'cloud')
final int cloud;
@JsonKey(name: 'feelslike_c')
final double feelslikeC;
@JsonKey(name: 'gust_kph')
final double gustKph;

CurrentWeatherResponse({
  required this.lastUpdatedEpoch,
  required this.lastUpdated,
  required this.tempC,
  required this.tempF,
  required this.isDay,
  required this.condition,
  required this.windKph,
  required this.windDir,
  required this.pressureMb,
  required this.humidity,
  required this.cloud,
  required this.feelslikeC,
  required this.gustKph,
});

factory CurrentWeatherResponse.fromJson(Map<String, dynamic>
json) => _$CurrentWeatherResponseFromJson(json);
Map<String, dynamic> toJson() =>
_$CurrentWeatherResponseToJson(this);
}

@JsonSerializable()
class HourResponse {

```

```

@JsonKey(name: 'time_epoch')
final int timeEpoch;
final String time;
@JsonKey(name: 'temp_c')
final double tempC;
final ConditionResponse condition;
@JsonKey(name: 'wind_kph')
final double windKph;
@JsonKey(name: 'wind_dir')
final String windDir;
@JsonKey(name: 'humidity')
final int humidity;
@JsonKey(name: 'cloud')
final int cloud;
@JsonKey(name: 'feelslike_c')
final double feelslikeC;
@JsonKey(name: 'chance_of_rain')
final int chanceOfRain;
@JsonKey(name: 'chance_of_snow')
final int chanceOfSnow;
@JsonKey(name: 'vis_km')
final double visKm;
@JsonKey(name: 'gust_kph')
final double gustKph;

HourResponse({
  required this.timeEpoch,
  required this.time,
  required this.tempC,
  required this.condition,
  required this.windKph,
  required this.windDir,
  required this.humidity,
  required this.cloud,
  required this.feelslikeC,
  required this.chanceOfRain,
  required this.chanceOfSnow,
  required this.visKm,
  required this.gustKph,
});

factory HourResponse.fromJson(Map<String, dynamic> json) =>
  _$HourResponseFromJson(json);
Map<String, dynamic> toJson() => _$HourResponseToJson(this);
}

@JsonSerializable()

```

```

class WeatherDayResponse {
    @JsonKey(name: 'maxtemp_c')
    final double maxTempC;
    @JsonKey(name: 'mintemp_c')
    final double minTempC;
    @JsonKey(name: 'avgtemp_c')
    final double avgTempC;
    @JsonKey(name: 'maxwind_kph')
    final double maxWindKph;
    @JsonKey(name: 'totalprecip_mm')
    final double totalPrecipMm;
    @JsonKey(name: 'avghumidity')
    final int avgHumidity;
    @JsonKey(name: 'daily_chance_of_rain')
    final int dailyChanceOfRain;
    @JsonKey(name: 'daily_chance_of_snow')
    final int dailyChanceOfSnow;
    final ConditionResponse condition;

    WeatherDayResponse({
        required this.maxTempC,
        required this.minTempC,
        required this.avgTempC,
        required this.maxWindKph,
        required this.totalPrecipMm,
        required this.avgHumidity,
        required this.dailyChanceOfRain,
        required this.dailyChanceOfSnow,
        required this.condition,
    });

    factory WeatherDayResponse.fromJson(Map<String, dynamic> json)
=> _$WeatherDayResponseFromJson(json);
    Map<String, dynamic> toJson() =>
_$WeatherDayResponseToJson(this);
}

@JsonSerializable()
class WeatherForecastResponse {
    @JsonKey(name: 'forecastday')
    final List<ForecastItemResponse> forecastDay;

    WeatherForecastResponse({required this.forecastDay});

    factory WeatherForecastResponse.fromJson(Map<String, dynamic>
json) => _$WeatherForecastResponseFromJson(json);
    Map<String, dynamic> toJson() =>

```

					ПГУ1.090301.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

    _$WeatherForecastResponseToJson(this);
}

@JsonSerializable()
class WeatherResponse {
    final LocationResponse location;
    final CurrentWeatherResponse current;
    final WeatherForecastResponse forecast;

    WeatherResponse({
        required this.location,
        required this.current,
        required this.forecast,
    });

    factory WeatherResponse.fromJson(Map<String, dynamic> json) =>
        _$WeatherResponseFromJson(json);
    Map<String, dynamic> toJson() =>
        _$WeatherResponseToJson(this);
}

@JsonSerializable()
class WeatherHistoryResponse {
    final LocationResponse location;
    final WeatherForecastResponse forecast;

    WeatherHistoryResponse({
        required this.location,
        required this.forecast,
    });

    factory WeatherHistoryResponse.fromJson(Map<String, dynamic>
json) => _$WeatherHistoryResponseFromJson(json);
    Map<String, dynamic> toJson() =>
        _$WeatherHistoryResponseToJson(this);
}

@JsonSerializable()
class ForecastItemResponse {
    final String date;
    final WeatherDayResponse day;
    final AstroResponse astro;
    @JsonKey(name: 'hour')
    final List<HourResponse> hour;
    ForecastItemResponse({

```

```

        required this.date,
        required this.day,
        required this.hour,
        required this.astro
    });

    factory ForecastItemResponse.fromJson(Map<String, dynamic>
json) => _$ForecastItemResponseFromJson(json);
    Map<String, dynamic> toJson() =>
_$ForecastItemResponseToJson(this);
}
@JsonSerializable()
class AstroResponse {
    final String sunrise;
    final String sunset;
    final String moonrise;
    final String moonset;
    @JsonKey(name: 'moon_phase')
    final String moonPhase;
    @JsonKey(name: 'moon_illumination')
    final int moonIllumination;

    AstroResponse({
        required this.sunrise,
        required this.sunset,
        required this.moonrise,
        required this.moonset,
        required this.moonPhase,
        required this.moonIllumination,
    });

    factory AstroResponse.fromJson(Map<String, dynamic> json) =>
_$AstroResponseFromJson(json);

    Map<String, dynamic> toJson() => _$AstroResponseToJson(this);
}

```

Файл «weather_response_model.g.dart»

```

// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'weather_response_model.dart';

//
*****
*****
// JsonSerializableGenerator
//

```

					ПГУ1.090301.12.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

```
*****
*****
```

```
LocationResponse _$LocationResponseFromJson(Map<String, dynamic>
json) =>
```

```
    LocationResponse(
        name: json['name'] as String,
        region: json['region'] as String,
        country: json['country'] as String,
        lat: (json['lat'] as num).toDouble(),
        lon: (json['lon'] as num).toDouble(),
        tzId: json['tz_id'] as String,
        localtimeEpoch: (json['localtime_epoch'] as num).toInt(),
        localtime: json['localtime'] as String,
    );
```

```
Map<String, dynamic> _$LocationResponseToJson(LocationResponse
instance) =>
```

```
    <String, dynamic>{
        'name': instance.name,
        'region': instance.region,
        'country': instance.country,
        'lat': instance.lat,
        'lon': instance.lon,
        'tz_id': instance.tzId,
        'localtime_epoch': instance.localtimeEpoch,
        'localtime': instance.localtime,
    };
```

```
ConditionResponse _$ConditionResponseFromJson(Map<String,
dynamic> json) =>
```

```
    ConditionResponse(
        text: json['text'] as String,
        icon: json['icon'] as String,
        code: (json['code'] as num).toInt(),
    );
```

```
Map<String, dynamic> _$ConditionResponseToJson(ConditionResponse
instance) =>
```

```
    <String, dynamic>{
        'text': instance.text,
        'icon': instance.icon,
        'code': instance.code,
    };
```

```
CurrentWeatherResponse _$CurrentWeatherResponseFromJson(
    Map<String, dynamic> json) =>
```

```

CurrentWeatherResponse(
    lastUpdatedEpoch: (json['last_updated_epoch'] as
num).toInt(),
    lastUpdated: json['last_updated'] as String,
    tempC: (json['temp_c'] as num).toDouble(),
    tempF: (json['temp_f'] as num).toDouble(),
    isDay: (json['is_day'] as num).toInt(),
    condition:
        ConditionResponse.fromJson(json['condition'] as
Map<String, dynamic>),
    windKph: (json['wind_kph'] as num).toDouble(),
    windDir: json['wind_dir'] as String,
    pressureMb: (json['pressure_mb'] as num).toDouble(),
    humidity: (json['humidity'] as num).toInt(),
    cloud: (json['cloud'] as num).toInt(),
    feelslikeC: (json['feelslike_c'] as num).toDouble(),
    gustKph: (json['gust_kph'] as num).toDouble(),
);

Map<String, dynamic> _$CurrentWeatherResponseToJson(
    CurrentWeatherResponse instance) =>
<String, dynamic>{
    'last_updated_epoch': instance.lastUpdatedEpoch,
    'last_updated': instance.lastUpdated,
    'temp_c': instance.tempC,
    'temp_f': instance.tempF,
    'is_day': instance.isDay,
    'condition': instance.condition,
    'wind_kph': instance.windKph,
    'wind_dir': instance.windDir,
    'pressure_mb': instance.pressureMb,
    'humidity': instance.humidity,
    'cloud': instance.cloud,
    'feelslike_c': instance.feelslikeC,
    'gust_kph': instance.gustKph,
};

HourResponse _$HourResponseFromJson(Map<String, dynamic> json)
=> HourResponse(
    timeEpoch: (json['time_epoch'] as num).toInt(),
    time: json['time'] as String,
    tempC: (json['temp_c'] as num).toDouble(),
    condition:
        ConditionResponse.fromJson(json['condition'] as
Map<String, dynamic>),
    windKph: (json['wind_kph'] as num).toDouble(),
    windDir: json['wind_dir'] as String,

```

```

        humidity: (json['humidity'] as num).toInt(),
        cloud: (json['cloud'] as num).toInt(),
        feelslikeC: (json['feelslike_c'] as num).toDouble(),
        chanceOfRain: (json['chance_of_rain'] as num).toInt(),
        chanceOfSnow: (json['chance_of_snow'] as num).toInt(),
        visKm: (json['vis_km'] as num).toDouble(),
        gustKph: (json['gust_kph'] as num).toDouble(),
    );

Map<String, dynamic> _$HourResponseToJson(HourResponse instance)
=>
    <String, dynamic>{
        'time_epoch': instance.timeEpoch,
        'time': instance.time,
        'temp_c': instance.tempC,
        'condition': instance.condition,
        'wind_kph': instance.windKph,
        'wind_dir': instance.windDir,
        'humidity': instance.humidity,
        'cloud': instance.cloud,
        'feelslike_c': instance.feelslikeC,
        'chance_of_rain': instance.chanceOfRain,
        'chance_of_snow': instance.chanceOfSnow,
        'vis_km': instance.visKm,
        'gust_kph': instance.gustKph,
    };

WeatherDayResponse _$WeatherDayResponseFromJson(Map<String,
dynamic> json) =>
    WeatherDayResponse(
        maxTempC: (json['maxtemp_c'] as num).toDouble(),
        minTempC: (json['mintemp_c'] as num).toDouble(),
        avgTempC: (json['avgtemp_c'] as num).toDouble(),
        maxWindKph: (json['maxwind_kph'] as num).toDouble(),
        totalPrecipMm: (json['totalprecip_mm'] as num).toDouble(),
        avgHumidity: (json['avghumidity'] as num).toInt(),
        dailyChanceOfRain: (json['daily_chance_of_rain'] as
num).toInt(),
        dailyChanceOfSnow: (json['daily_chance_of_snow'] as
num).toInt(),
        condition:
            ConditionResponse.fromJson(json['condition'] as
Map<String, dynamic>),
    );

Map<String, dynamic>
_$WeatherDayResponseToJson(WeatherDayResponse instance) =>

```

```

<String, dynamic>{
    'maxtemp_c': instance.maxTempC,
    'mintemp_c': instance.minTempC,
    'avgtemp_c': instance.avgTempC,
    'maxwind_kph': instance.maxWindKph,
    'totalprecip_mm': instance.totalPrecipMm,
    'avghumidity': instance.avgHumidity,
    'daily_chance_of_rain': instance.dailyChanceOfRain,
    'daily_chance_of_snow': instance.dailyChanceOfSnow,
    'condition': instance.condition,
};

WeatherForecastResponse _$WeatherForecastResponseFromJson(
    Map<String, dynamic> json) =>
    WeatherForecastResponse(
        forecastDay: (json['forecastday'] as List<dynamic>)
            .map((e) => ForecastItemResponse.fromJson(e as
Map<String, dynamic>))
            .toList(),
    );

Map<String, dynamic> _$WeatherForecastResponseToJson(
    WeatherForecastResponse instance) =>
    <String, dynamic>{
        'forecastday': instance.forecastDay,
    };

WeatherResponse _$WeatherResponseFromJson(Map<String, dynamic>
json) =>
    WeatherResponse(
        location:
            LocationResponse.fromJson(json['location'] as
Map<String, dynamic>),
        current: CurrentWeatherResponse.fromJson(
            json['current'] as Map<String, dynamic>),
        forecast: WeatherForecastResponse.fromJson(
            json['forecast'] as Map<String, dynamic>),
    );

Map<String, dynamic> _$WeatherResponseToJson(WeatherResponse
instance) =>
    <String, dynamic>{
        'location': instance.location,
        'current': instance.current,
        'forecast': instance.forecast,
    };

```

```

WeatherHistoryResponse _$WeatherHistoryResponseFromJson(
    Map<String, dynamic> json) =>
    WeatherHistoryResponse(
        location:
            LocationResponse.fromJson(json['location'] as
Map<String, dynamic>),
        forecast: WeatherForecastResponse.fromJson(
            json['forecast'] as Map<String, dynamic>),
    );

Map<String, dynamic> _$WeatherHistoryResponseToJson(
    WeatherHistoryResponse instance) =>
    <String, dynamic>{
        'location': instance.location,
        'forecast': instance.forecast,
    };

ForecastItemResponse _$ForecastItemResponseFromJson(
    Map<String, dynamic> json) =>
    ForecastItemResponse(
        date: json['date'] as String,
        day: WeatherDayResponse.fromJson(json['day'] as
Map<String, dynamic>),
        hour: (json['hour'] as List<dynamic>)
            .map((e) => HourResponse.fromJson(e as Map<String,
dynamic>))
            .toList(),
        astro: AstroResponse.fromJson(json['astro'] as Map<String,
dynamic>),
    );

Map<String, dynamic> _$ForecastItemResponseToJson(
    ForecastItemResponse instance) =>
    <String, dynamic>{
        'date': instance.date,
        'day': instance.day,
        'astro': instance.astro,
        'hour': instance.hour,
    };

AstroResponse _$AstroResponseFromJson(Map<String, dynamic> json)
=>
    AstroResponse(
        sunrise: json['sunrise'] as String,
        sunset: json['sunset'] as String,
        moonrise: json['moonrise'] as String,
        moonset: json['moonset'] as String,
    );

```

```

        moonPhase: json['moon_phase'] as String,
        moonIllumination: (json['moon_illumination'] as
num).toInt(),
    );

Map<String, dynamic> _$AstroResponseToJson(AstroResponse
instance) =>
    <String, dynamic>{
        'sunrise': instance.sunrise,
        'sunset': instance.sunset,
        'moonrise': instance.moonrise,
        'moonset': instance.moonset,
        'moon_phase': instance.moonPhase,
        'moon_illumination': instance.moonIllumination,
    };

```

Файл «weather_service.dart»

```

import 'package:weather_app/model/current_weather_model.dart';
/// WeatherService - это абстрактный класс, который определяет
контракт для сервиса погоды.
/// Он содержит метод `getCurrentWeather`, который должен быть
реализован в классах, реализующих этот интерфейс.
/// Метод возвращает объект `CurrentWeatherModel`, содержащий
данные о текущей погоде.
abstract class WeatherService {

    /// Метод для получения текущей погоды.
    /// Принимает:
    /// - `place` (место) - название города или координаты
(широта, долгота).
    /// - `timeDt` (начальная дата) - дата начала периода для
получения исторических данных.
    /// - `endDt` (конечная дата) - дата окончания периода для
получения исторических данных.
    /// Возвращает:
    /// - `Future<CurrentWeatherModel>` - объект с данными о
текущей погоде.
    Future<CurrentWeatherModel> getCurrentWeather({required String
place, required String timeDt, required String endDt});
}

class WeatherException implements Exception {
    final String message;

```

```

WeatherException(this.message);
}

```

Файл «weather_service_impl.dart»

```

import 'package:weather_app/factory/weather_factory.dart';
import 'package:weather_app/gateway/weather_gateway.dart';
import 'package:weather_app/model/current_weather_model.dart';
import 'package:weather_app/service/weather_service.dart';

/// WeatherServiceImpl - это реализация интерфейса
WeatherService.
/// Он использует WeatherGateway для получения данных о погоде и
WeatherFactory для создания моделей
class WeatherServiceImpl implements WeatherService {
  final WeatherGateway _weatherGateway;
  final WeatherFactory _weatherFactory;

  WeatherServiceImpl(
    this._weatherGateway,
    this._weatherFactory,
  );

  @override
  Future<CurrentWeatherModel> getCurrentWeather({required String
place,required String timeDt, required String endDt}) async {
    // Получаем текущую погоду
    final responseCurrent = await
_weatherGateway.getCurrentWeather(place);
    //TODO костыль задержка, бесплатное апи имеет ограничение по
количеству запросов в секунду
    // Искусственная задержка для избежания ограничений API
    await Future.delayed(const Duration(seconds: 2));
    final responseHistory = await
_weatherGateway.getHistoryWeather(place,timeDt: timeDt,endDt:
endDt);
    // Создаем и возвращаем модель текущей погоды
    return await
_weatherFactory.getCurrentWeather(responseCurrent,
responseHistory);
  }
}

```

Файл «snack_bar_widget.dart»

					ПГУ1.090301.12.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

import 'package:flutter/material.dart';
/// Функция для отображения SnackBar с сообщением.
/// SnackBar - это небольшое всплывающее окно в нижней части
экрана,
/// которое используется для отображения кратких сообщений
пользователю.
///
/// Параметры:
/// - `context` (BuildContext): Контекст текущего виджета,
необходим для доступа к ScaffoldMessenger.
/// - `message` (String): Текст сообщения, которое будет
отображено в SnackBar.
void showSnackBar(BuildContext context, String message) {
  final snackBar = SnackBar(
    backgroundColor: Colors.white,
    content: Text(message, style: TextStyle(
      fontSize: 16,
      color: Colors.blue,
      fontWeight: FontWeight.bold,
    )),
    duration: Duration(seconds: 2), // Время, через которое окно
исчезнет
  );

  // Показываем SnackBar
  ScaffoldMessenger.of(context).showSnackBar(snackBar);
}

```

Файл «weather_astro_widget.dart»

```

import 'package:flutter/material.dart';
import 'package:weather_app/model/current_weather_model.dart';
import 'package:weather_app/utils.dart';
/// Виджет `WeatherAstroWidget` отображает астрономические
данные, такие как время восхода и заката солнца,
/// а также время восхода и захода луны.
class WeatherAstroWidget extends StatelessWidget {
  final Astro astro;

  const WeatherAstroWidget({super.key, required this.astro});

  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Container(

```

					ПГУ1.090301.12.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

        margin: EdgeInsets.only(left: 8),
        child: Text("Астрономические данные",
            style: TextStyle(
                fontSize: 18,
                color: Colors.blue[800],
                fontWeight: FontWeight.bold,
            )),
    ),
    SizedBox(height: 20),
    Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
            _buildWeatherBox("Восход солнца", astro.sunrise,
Icons.wb_sunny),
            _buildWeatherBox("Закат", astro.sunset,
Icons.nightlight_round),
            _buildWeatherBox("Восход Луны", astro.moonrise,
Icons.brightness_3),
            _buildWeatherBox("Заход луны", astro.moonset,
Icons.brightness_2),
        ],
    ),
],
);
}

```

```

Widget _buildWeatherBox(String label, String value, IconData
icon) {
    return Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
            Icon(icon, size: 40, color: Colors.blueGrey),
            SizedBox(height: 8),
            Text(
                label,
                style: TextStyle(
                    fontSize: 14,
                    fontWeight: FontWeight.bold,
                ),
            ),
            SizedBox(height: 4),
            Text(convertToRussianTimeFormat(value),
                style: TextStyle(
                    fontSize: 14,
                    fontWeight: FontWeight.bold,
                )),
        ],
    ),
}

```

```

    );
  }
}

```

Файл «weather_graph_widget.dart»

```

import 'dart:ui';

import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';
import 'package:weather_app/model/current_weather_model.dart';
import 'package:weather_app/utils.dart';
/// Виджет `WeatherGraphWidget` отображает почасовые данные о
погоде в виде горизонтального списка
/// и графика температуры. Пользователь может выбирать
конкретный час, нажимая на соответствующий элемент.
class WeatherGraphWidget extends StatefulWidget {
  final List<HourModel> hours;
  final String date;
  final void Function(int index) onTapHour;
  final int currentHourIndex;

  const WeatherGraphWidget(
    {Key? key,
      required this.hours,
      required this.date,
      required this.onTapHour,
      required this.currentHourIndex})
    : super(key: key);

  @override
  _WeatherGraphWidgetState createState() =>
    _WeatherGraphWidgetState();
}

class _WeatherGraphWidgetState extends State<WeatherGraphWidget>
{
  final ScrollController _scrollController = ScrollController();

  @override
  Widget build(BuildContext context) {
    double maxTemp =
      widget.hours.map((e) => e.tempC).reduce((a, b) => a > b ? a
: b);
    double minTemp =
      widget.hours.map((e) => e.tempC).reduce((a, b) => a < b ? a
: b);

```

					ПГУ1.090301.12.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

```

double graphWidth = widget.hours.length * 88.0;

return Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Container(
      margin: EdgeInsets.only(left: 12),
      child: Text(
        'Данные о погоде ${formatMonthDay(widget.date)} по
часам',
        style: TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,
          color: Colors.blue[800],
        ),
      ),
    ),
    SizedBox(
      height: 12,
    ),
    SingleChildScrollView(
      controller: _scrollController,
      scrollDirection: Axis.horizontal,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: [
          Row(
            children: List.generate(
              widget.hours.length,
              (i) => GestureDetector(
                onTap: () {
                  widget.onTapHour(i);
                },
                child: Container(
                  width: 84,
                  margin: const
EdgeInsets.symmetric(horizontal: 4),
                  decoration: BoxDecoration(
                    borderRadius:
BorderRadius.all(Radius.circular(12)),
                    color: i == widget.currentHourIndex
? Colors.blue.withOpacity(0.60)
: Colors.white30.withOpacity(0.35),
                  ),
                  child: Column(
                    children: [
                      Text(widget.hours[i].time.split('

```

[illegible]

					<p style="text-align: center;"><i>ПГУ1.090301.12.001 ПЗ</i></p>	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ),
        ],
    ),
)
],
);
}
}
/// Класс `TempGraphPainter` отвечает за отрисовку графика
температуры.
class TempGraphPainter extends CustomPainter {
    final List<HourModel> hours;
    final double maxTemp;
    final double minTemp;
    final double paddingY = 15;

    TempGraphPainter(this.hours, this.maxTemp, this.minTemp);

    @override
    void paint(Canvas canvas, Size size) {
        final Paint gridPaint = Paint()
            ..color = Colors.grey
            ..strokeWidth = 1;

        final Paint linePaint = Paint()
            ..color = Colors.black
            ..strokeWidth = 3
            ..style = PaintingStyle.stroke
            ..strokeCap = StrokeCap.round;

        final Path path = Path();
        final double stepX = size.width / (hours.length - 1);
        final double stepY = (size.height - paddingY * 2) / (maxTemp
- minTemp);

        for (int i = 0; i < hours.length; i++) {
            double x = i * stepX;
            canvas.drawLine(Offset(x, 0), Offset(x, size.height),
gridPaint);
        }
        for (double temp = minTemp; temp <= maxTemp; temp++) {
            double y = size.height - paddingY - ((temp - minTemp) *
stepY);
            canvas.drawLine(Offset(0, y), Offset(size.width, y),
gridPaint);
        }
    }
}

```

```

    for (int i = 0; i < hours.length; i++) {
      double x = i * stepX;
      double y = size.height - paddingY - ((hours[i].tempC -
minTemp) * stepY);
      if (i == 0) {
        path.moveTo(x, y);
      } else {
        path.lineTo(x, y);
      }
    }

    canvas.drawPath(path, linePaint);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) =>
true;
}

```

Файл «weathe_hour.dart»

```

import 'package:flutter/material.dart';
import 'package:weather_app/model/current_weather_model.dart';
import 'package:weather_app/utils.dart';
/// Виджет `WeatherHourWidget` отображает информацию о погоде
для конкретного часа.
/// Использует `GridView` для организации данных в виде сетки.
class WeatherHourWidget extends StatelessWidget {
  const WeatherHourWidget({super.key, required this.hour});

  final HourModel hour;

  @override
  Widget build(BuildContext context) {
    return Container(
      margin: const EdgeInsets.symmetric(horizontal: 8),
      child: GridView.count(
        physics: NeverScrollableScrollPhysics(),
        shrinkWrap: true,
        crossAxisCount: 3,
        childAspectRatio: 1.0,
        mainAxisSpacing: 8.0,
        crossAxisSpacing: 8.0,
        children: [
          _buildWeatherItem(Icons.access_time, "Время",
formatDateTime(hour.time)),

```

					ПГУ1.090301.12.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

        _buildWeatherItem(Icons.thermostat, "Температура",
"${hour.tempC}°C"),
        _buildWeatherItem(Icons.cloud, "Облачность",
"${hour.cloud}%"),
        _buildWeatherItem(Icons.wind_power, "Ветер",
"${hour.windDir} на ${hour.gustKph} км/ч"),
        _buildWeatherItem(Icons.thermostat_outlined,
"Ощущается как", "${hour.feelsLikeC}°C"),
        _buildWeatherItem(Icons.water_drop, "Шанс дождя",
"${hour.chanceOfRain}%"),
        _buildWeatherItem(Icons.ac_unit, "Шанс снега",
"${hour.chanceOfSnow}%"),
        _buildWeatherItem(Icons.visibility, "Видимость",
"${hour.visKm} км"),
        _buildWeatherItem(Icons.sunny, "Условие", hour.text),
    ],
    ),
);
}
/// Каждый элемент содержит иконку, заголовок и значение.
///
/// Параметры:
/// - `icon` (IconData): Иконка для элемента.
/// - `label` (String): Заголовок элемента.
/// - `value` (String): Значение элемента.
Widget _buildWeatherItem(IconData icon, String label, String
value) {
    return Column(
        mainAxisAlignment: MainAxisAlignment.start,
        children: [
            Icon(icon, size: 30, color: Colors.blue),
            SizedBox(height: 4),
            Text(label, style: TextStyle(fontSize: 14, fontWeight:
FontWeight.bold)),
            SizedBox(height: 4),
            Text(value, style: TextStyle(fontSize: 14), textAlign:
TextAlign.center),
        ],
    );
}
}

```

Приложение В

(Обязательное)

Мобильное приложение «Погода»

Презентация

					ПГУ1.090301.12.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ «ПОГОДА» НА FLUTTER

Автор работы:
студент группы 21ВВП1 Кривенкова В.С.
Научный руководитель:
д.т.н., профессор кафедры ВТ ПГУ Зинкин С.А.

Цель работы

Целью является разработка приложения для прогноза погоды. В рамках проекта необходимо:

- изучить особенности работы мобильных сервисов погоды;
- определить ключевые функции и создать удобный интерфейс для взаимодействия с приложением;
- разработать современное приложение, которое сможет предоставить точные прогнозы;
- протестировать систему, чтобы повысить ее надёжность.

Постановка задачи

Основная цель приложения - обеспечить пользователей легким доступом к актуальной информации о погоде, чтобы они могли планировать свои дела.

Кроме того, приложение должно обладать высокой производительностью и быть легко масштабируемым. Это позволит в будущем добавлять новые функции и улучшения, если возникнет такая необходимость. В приложении должно быть реализовано:

- отображение погоды на несколько дней;
- отображение текущей погоды;
- выбор местоположения для отображения погоды;
- отображение дополнительных данных: влажность, ветер.

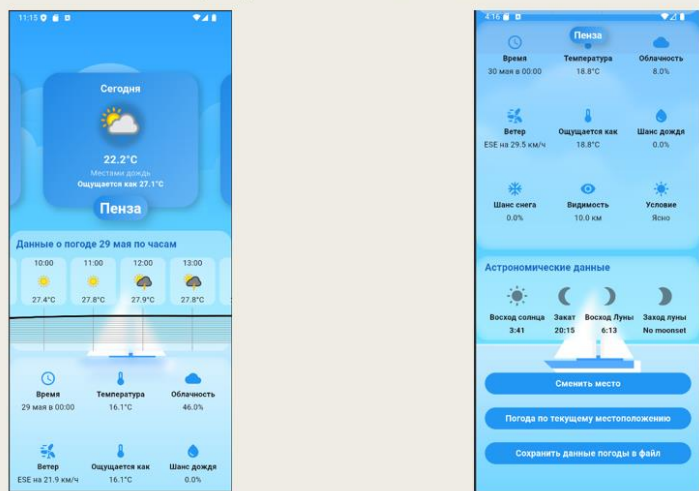
Аналоги

На сегодняшний день существует множество приложений для прогнозирования погоды, каждое из которых имеет свои особенности. Я рассмотрела 3 популярных сервиса:

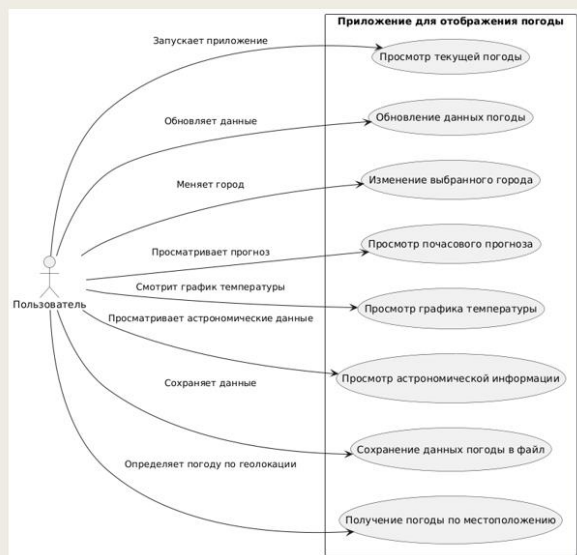
- [Яндекс.Погода](#);
- [Gismeteo](#);
- Приложение «Погода».

					ПГУ1.090301.12.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

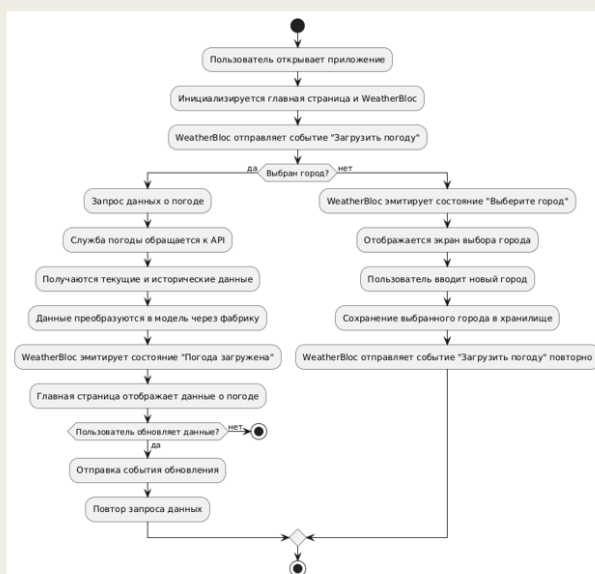
Интерфейс программы:



UML-диаграмма вариантов использования



UML-диаграмма деятельности



Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильное приложение «Погода». Приложение предоставляет данные о текущей температуре, влажности, скорости ветра, а также почасовой прогноз. Кроме того, оно включает астрономические сведения, такие как время восхода и заката солнца и луны, что делает его функционал более полным и полезным.