МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования  
«Пензенский государственный технологический университет»

(ПензГТУ)

Факультет автоматизированных информационных технологий

Кафедра «Информационные технологии и системы»

Дисциплина «Программирование на языках высокого уровня»

КУРСОВАЯ РАБОТА

на тему: «Разработка погодного дэшборда»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ПензГТУ 2.090302.04.ПЗ

Выполнил: студент гр. 23ИСб Сидоров А.А.

Руководитель: к.т.н., доцент каф. ИТС Доросинский А.Ю.

Работа защищена с оценкой:*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Пенза, 2025 г.

Утверждаю

зав. каф. ИТС Михеев М.Ю.

«*\_\_\_\_\_*»*\_\_\_\_\_\_\_\_\_\_\_\_\_*

ЗАДАНИЕ

на курсовую работу по дисциплине  
«Программирование на языках высокого уровня»

Студента Сидоров А.А. Группа 23ИС1б

Тема работы: «Разработка погодного дэшборда»

Исходные данные (технические требования на работу)

1. Назначение приложения: Приложение представляет собой погодный дэшборд, который предоставляет пользователям актуальную метеорологическую информацию в текстовом формате через простой интерфейс.

2. Состав и структура входных и выходных, исходных данных:

Входные данные: *API*-ключи, запросы пользователя, конфигурационные файлы (*JSON*, *YAML*, *INI*)

Исходные данные: *JSON*-объекты от метеорологических *API*

Выходные данные: текстовые сообщения (текущая погода, прогнозы), отчёты и уведомления об ошибках.

3. Перечень сущностей: пользователь, *API*-ключ, запрос пользователя, конфигурационный файл, метеорологический *API*, *JSON*-данные от *API*, текстовое сообщение, отчёты об ошибках, прогноз погоды.

4. Содержание расчетной части: перечень принятых сокращений, введение, анализ предметной области, теоретическая часть, техническое решение, заключение, библиографический список, приложения.

5. Графическая часть:

- схемы приложения (3 листа формата А4);

- интерфейс приложения (1 лист формата А4);

6. Экспериментальная часть:

- разработка приложения на языке *Java*.

7. Срок выполнения работы по разделам:

|  |  |
| --- | --- |
| 1. Постановка задачи до |  |
| 2. Анализ предметной области |  |
| 3. Разработка ПО |  |
| 4. Отладка сетевого кода |  |
| 5. Оформление графической части до |  |
| 6. Оформление пояснительной записки до |  |

Студент гр. 23ИС1б Сидоров А.А. / *\_\_\_\_\_\_\_\_\_\_* /

Задание получил «*\_\_\_*» *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* 202*\_* г.

Руководитель к.т.н., доцент каф. ИТС Доросинский А.Ю. / *\_\_\_\_\_\_\_\_\_\_* /

Задание выдал «*\_\_\_*» *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* 202*\_* г.

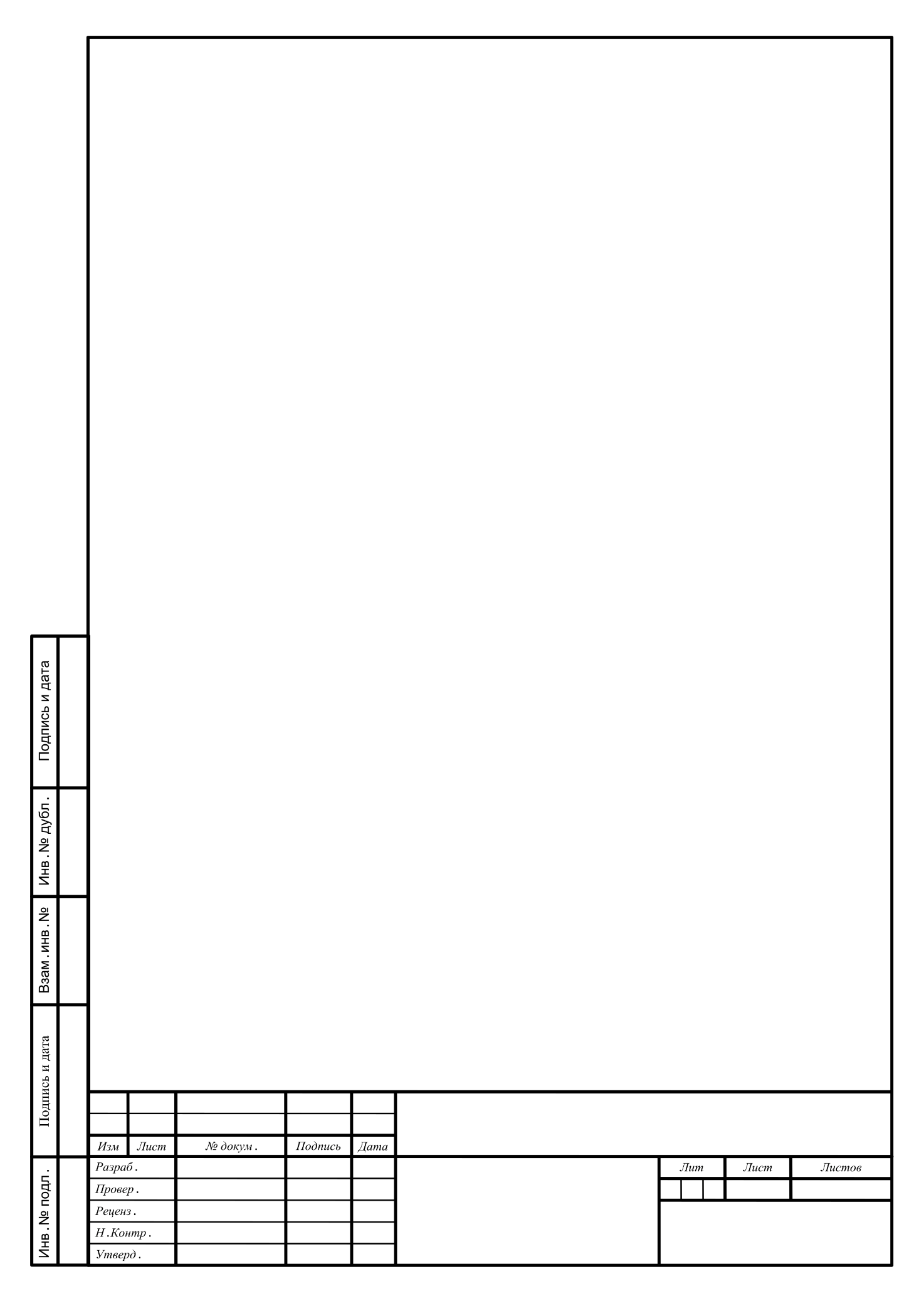
Дата защиты работы *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Студент гр. 23ИС1б Сидоров А.А. / *\_\_\_\_\_\_\_\_\_\_* /

Руководитель к.т.н., доцент каф. ИТС Доросинский А.Ю. / *\_\_\_\_\_\_\_\_\_\_\_* /

Перечень принятых сокращений

1. ***API*** – *Application Programming Interface* (интерфейс программирования приложений).
2. ***JSON*** – *JavaScript Object Notation* (формат обмена данными, основанный на тексте).
3. ***UML*** *– Unified Modeling Language* (унифицированный язык моделирования).
4. ***CLI*** *– Command Line Interface* (интерфейс командной строки).
5. ***HTTP*** *– HyperText Transfer Protocol* (протокол передачи гипертекста).
6. ***URL*** *– Uniform Resource Locator* (унифицированный указатель ресурса).
7. ***IDE*** *– Integrated Development Environment* (интегрированная среда разработки).
8. ***JUnit*** – библиотека для модульного тестирования на языке *Java*.
9. ***Gson*** – библиотека для работы с *JSON* в *Java*.
10. ***ASCII*** – *American Standard Code for Information Interchange* (американский стандартный код для обмена информацией).
11. **TЗ** – Техническое задание.
12. **ГОСТ** – Государственный стандарт.
13. ***PVS-Studio*** – инструмент для статического анализа кода.
14. ***Valgrind*** – инструмент для анализа памяти и производительности.
15. ***JMeter*** – инструмент для тестирования производительности и нагрузки.
16. ***UTC*** *– Coordinated Universal Time* (всемирное координированное время).
17. ***JAR*** *– Java Archive* (архивный файл для *Java*-приложений).
18. ***JDK*** *– Java Development Kit* (комплект разработчика на *Java*).
19. ***JRE*** *– Java Runtime Environment* (среда выполнения *Java*).

СОДЕРЖАНИЕ

[Перечень принятых сокращений 4](#_Toc189152622)

[Введение 7](#_Toc189152623)

[Глава 1 Анализ предметной области 9](#_Toc189152624)

[1.1 Общие сведения 9](#_Toc189152625)

[1.2 Аналоги и их принцип работы 10](#_Toc189152626)

[1.3 Анализ языков программирования и инструментов разработки 12](#_Toc189152627)

[1.4 Выводы по разделу 13](#_Toc189152628)

[Глава 2 Теоретическая часть 15](#_Toc189152629)

[2.1 Общие сведения 15](#_Toc189152630)

[2.2 Моделирование *UML*-Диаграмм 15](#_Toc189152631)

[2.3 Спецификация *frontend* части программы 16](#_Toc189152632)

[2.4 Спецификация *backend* части программы 16](#_Toc189152633)

[2.5 Выводы по разделу 17](#_Toc189152634)

[Глава 3 Техническое решение 18](#_Toc189152635)

[3.1 Общие сведения 18](#_Toc189152636)

[3.2 Выбор и обоснование использования зависимостей проекта 18](#_Toc189152637)

[3.3 Описание разработки и результатов *Frontend* части 18](#_Toc189152638)

[3.4 Описание разработки и результатов *Backend* части 21](#_Toc189152639)

[3.5 Тестирование 24](#_Toc189152640)

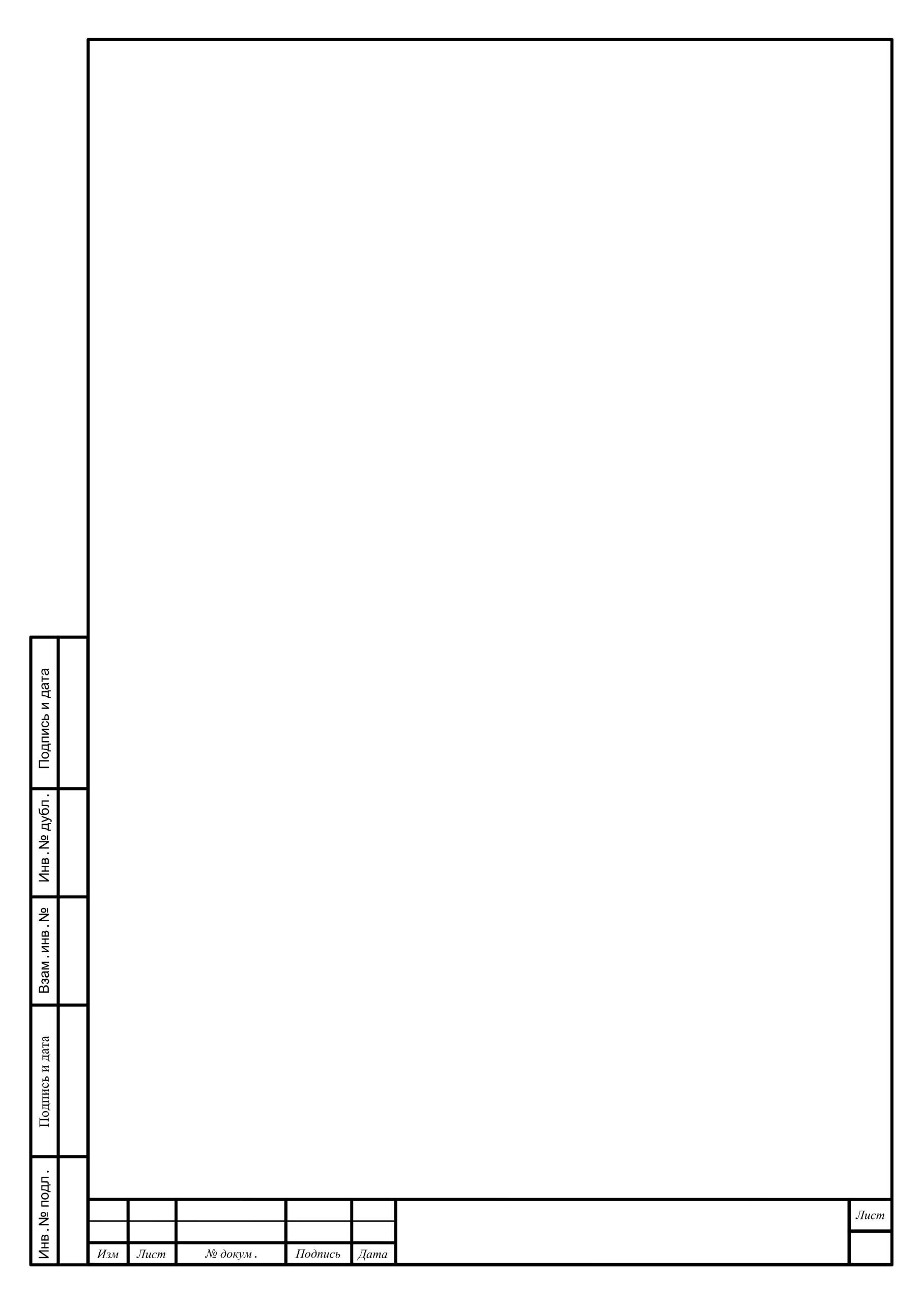
[3.6 Выводы по разделу 25](#_Toc189152641)

[Заключение 26](#_Toc189152642)

[Библиографический список 27](#_Toc189152643)

[Приложение А – *UML*-Диаграмма программы (*Use*-*case*) 30](#_Toc189152644)

[Приложение Б – *UML*- Диаграмма программы (*Activity*) 31](#_Toc189152645)

[Приложение В - *UML*- Диаграмма программы (*Sequence*) 32](#_Toc189152646)

[Приложение Г – *Frontend* часть программы 33](#_Toc189152647)

[Приложение Д – *Backend* часть программы 34](#_Toc189152648)

[Приложение Е – *POM*.*XML* для *Frontend* части программы 37](#_Toc189152649)

[Приложение Ё – *POM*.*XML* для *Backend* части программы 39](#_Toc189152650)

Введение

Современный мир неотъемлемо связан с анализом и прогнозированием погодных условий, что играет ключевую роль в различных сферах деятельности, таких как сельское хозяйство, транспорт, строительство и даже повседневная жизнь людей. Прогнозирование погоды помогает людям планировать свои действия, минимизировать риски и повышать эффективность их работы. Например, фермеры используют прогнозы для оптимизации времени посева и сбора урожая, а транспортные компании - для обеспечения безопасности перевозок. Важно отметить, что с учетом глобальных изменений климата, экстремальных погодных явлений, таких как ураганы, наводнения и засухи, точные прогнозы становятся не только важным инструментом для бизнеса, но и жизненно необходимыми для предотвращения катастроф.

Актуальность точного и своевременного получения метеорологической информации становится всё более значимой в условиях нынешних вызовов. Технологии обработки данных и прогнозирования погоды развиваются стремительными темпами, что открывает новые возможности для создания более точных и доступных сервисов для пользователей. В этом контексте разработка погодных дэшбордов, которые собирают и отображают информацию в реальном времени, становится не только полезной, но и необходимой задачей. Погодные сервисы, предоставляющие информацию через простые и удобные интерфейсы, позволяют эффективно реагировать на изменения погоды и принимать обоснованные решения в различных областях, что существенно повышает качество жизни и работы людей в условиях переменчивой природы.

Цель: разработка погодного дэшборда, который позволит пользователям получать комплексную информацию о текущих и прогнозируемых погодных условиях через простой интерфейс в терминале.

Задача: создание интуитивно понятного и удобного для восприятия интерфейса, способного обрабатывать данные из различных источников и предоставлять их в виде текстовых отчетов.

В первом разделе рассматриваются основные аспекты метеорологических систем, включая методы сбора и обработки данных. Особое внимание уделяется использованию *API* для получения актуальной погодной информации и требованиям пользователей к интерфейсу. Приведён анализ существующих решений, что позволяет выявить лучшие практики и возможности для улучшения. Также рассматривается выбор языков программирования и инструментов разработки, которые будут использоваться в проекте. Описываются средства анализа кода, помогающие обеспечить качество и производительность системы.

Второй раздел охватывает теоретические основы работы систем, методы прогнозирования и обработки данных. Далее рассматриваются моделирование ПО с использованием различных *UML*-диаграмм для понимания взаимодействий и процессов. Описываются функциональность фронт-части, включая примеры взаимодействия пользователя, и бэк-части, включая серверную логику и структуры данных.

В третьем разделе описывает процесс разработки погодного дэшборда, начиная с выбора технологий и инструментов. Обоснован выбор библиотек и зависимостей для необходимого функционала. Описано создание пользовательского интерфейса с примерами кода и тестированием. Рассмотрена разработка серверной части для обработки данных и взаимодействия с внешними источниками. Тестирование включает юнит-тестирование отдельных компонентов и интеграционное/приёмочное тестирование всей системы.

Глава 1 Анализ предметной области

1.1 Общие сведения

Предметная область данной курсовой работы охватывает разработку погодного дэшборда, предназначенного для отображения текущих и прогнозируемых погодных условий через простой интерфейс в терминале.

Основное внимание уделяется методам сбора данных из различных источников. В данном случае используются *API* метеорологического сервиса *OpenWeatherMap*. Эти сервисы предоставляют данные в формате *JSON*, содержащем информацию о текущей погоде, прогнозах на ближайшие дни, температуре, влажности, атмосферном давлении и других параметрах. Сбор данных из надежных источников обеспечивает высокую точность информации, предоставляемой пользователю.

Процесс обработки данных также играет важную роль. Полученные *JSON*-объекты необходимо преобразовать в удобочитаемый текстовый формат, который будет выводиться в терминал. Это требует применения современных технологий анализа данных и обработки информации. Важным элементом работы является выбор и адаптация алгоритмов прогнозирования, которые будут использоваться для предсказания погодных условий на основе исторических данных. Такой подход позволяет не только предоставлять текущую информацию, но и делать точные прогнозы, что значительно повышает ценность системы для пользователей.

Кроме того, анализ предметной области включает рассмотрение аналогичных решений и принципов их работы. Анализ существующих погодных дэшбордов помогает выявить лучшие практики и возможности для улучшения. Особое внимание уделяется функциональности, удобству использования и качеству предоставляемых данных. Выявление недостатков существующих систем позволяет определить направления для совершенствования разрабатываемого приложения.

В рамках анализа также рассматриваются языки программирования и инструменты разработки, которые будут использованы в проекте. Например, язык *Java* может быть выбран для создания серверной части системы, обеспечивающей обработку запросов и взаимодействие с внешними источниками данных. Для анализа кода и обеспечения его качества могут применятся такие инструменты, как *Valgrind*, *JMeter* и *PVS*-*Studio*. Эти средства позволяют выявлять ошибки на ранних этапах разработки и повышать производительность системы.

Разработка технического задания (ТЗ) включает формирование и структурированное описание требований к системе в соответствии с ГОСТ 34.602-89. ТЗ определяет цели и область применения системы, функциональные и технические требования, сроки выполнения, а также порядок испытаний и приемки. Основными разделами ТЗ являются: назначение системы, требования к функциональности, интерфейсу, надежности, эксплуатационные характеристики и поддержка. Этот документ обеспечивает нормативную основу для проектирования, разработки, тестирования и ввода системы в эксплуатацию, гарантируя соответствие установленным требованиям.

1.2 Аналоги и их принцип работы

При разработке погодного дэшборда важно изучить существующие аналоги, их функциональные возможности и принципы работы. Это поможет определить лучшие подходы и избежать возможных недостатков при создании собственной системы.

**1. *OpenWeatherMap* *CLI***

*OpenWeatherMap* предоставляет *API* для получения погодных данных, которые можно использовать в терминальных приложениях. Доступ к данным осуществляется через *HTTP*-запросы, а обработка информации происходит с использованием *JSON*-ответов.

**Функции**:

* Отображение текущей температуры, влажности, скорости ветра.
* Возможность запроса прогноза на несколько дней.
* Поддержка различных геолокаций.
* Необходимость *API*-ключа для работы.

**2. *wttr*.*in***

Это веб-сервис, который использует данные *OpenWeatherMap* и предоставляет их в удобном текстовом формате, доступном через командную строку.

**Функции**:

* Запрос погоды через команду.
* Интерактивный интерфейс с *ASCII*-графикой.
* Возможность получения прогноза на несколько дней.
* Автоматическое определение местоположения.

**3. *WeatherCLI***

**Принцип работы**: Это *Python*-библиотека для получения информации о погоде через различные *API*.

**Функции**:

* Поддержка нескольких поставщиков погодных данных (*OpenWeatherMap*, *WeatherAPI* и др.).
* Гибкость в выборе формата отображения информации.
* Возможность настройки выходных данных.
* Отображение погодных условий в реальном времени и прогнозов.

**4. *WeGo***

Это утилита на *Go*, которая использует различные источники (например, *OpenWeatherMap*) для получения информации о погоде и выводит её в консоль в удобочитаемом виде.

**Функции**:

* Поддержка нескольких поставщиков погодных данных.
* Отображение данных в графическом формате с символами *ASCII*.
* Гибкая настройка параметров отображения.
* Возможность работы без графического интерфейса.

1.3 Анализ языков программирования и инструментов разработки

При выборе языков программирования и инструментов разработки для погодного дэшборда необходимо учитывать несколько ключевых факторов. Важными аспектами являются производительность языка, его способность эффективно работать с сетевыми запросами, удобство интеграции с *API* и сторонними библиотеками, а также поддержка инструментов анализа кода и тестирования. Также следует учитывать доступность документации, активность сообщества разработчиков и совместимость с различными платформами. Все эти факторы влияют на конечное качество и надежность разрабатываемого программного продукта.

**Языки программирования:**

* *Java*: Надежный кроссплатформенный язык с поддержкой многопоточного выполнения, что делает его удобным для обработки погодных данных.
* *C*++: Высокопроизводительный язык, позволяющий эффективно управлять ресурсами, но требующий больше усилий в разработке.
* *TypeScript*: Расширение *JavaScript* с типизацией, удобное для разработки веб-интерфейсов погодных сервисов.

Среды разработки:

* *VS* *Code*: Легковесный редактор с широкими возможностями расширения и поддержки различных языков.
* *Zed*: Производительная среда разработки с акцентом на многопоточные вычисления.
* *IntelliJ* *IDEA*: Интегрированная среда разработки, оптимизированная для *Java* и обеспечивающая удобные инструменты рефакторинга.

**Анализ кода:**

* *Valgrind*, *JMeter* и другие: Инструменты для тестирования производительности и выявления утечек памяти.
* *PVS*-*Studio* и другие: Средства статического анализа кода, помогающие обнаружить потенциальные ошибки и уязвимости.

Выбор языков и инструментов зависит от требований проекта. *Java* в сочетании с *VSCode* являются сбалансированным решением для разработки надёжного и простого для разработки погодного дэшборда. А использование различных средств анализа кода поможет определить недостатки и преимущества разработанного приложения.

1.4 Выводы по разделу

В рамках анализа предметной области был рассмотрен процесс разработки погодного дэшборда, который включает сбор и обработку данных через *API*, таких как *OpenWeatherMap*, для отображения актуальной информации о погоде. Ключевыми аспектами являются точность данных, предоставляемых сторонними сервисами, и необходимость их качественной обработки для вывода в удобочитаемом формате в терминале. Также важным элементом является использование алгоритмов прогнозирования на основе исторических данных для обеспечения точных прогнозов. Анализ существующих аналогичных решений позволил выделить лучшие практики и определить направления для улучшения разрабатываемой системы. Выбор технологий, таких как *Java*, *C*++, а также использование инструментов анализа и тестирования кода, как *Valgrind* и *PVS*-*Studio*, позволит повысить производительность и надежность приложения. В целом, разработка погодного дэшборда требует внимательного подхода к каждому этапу, от сбора данных до разработки интерфейса и обеспечения высококачественного функционирования системы.

Глава 2 Теоретическая часть

2.1 Общие сведения

Погодные дэшборды представляют собой программные решения, предназначенные для отображения метеорологических данных в удобной и наглядной форме. Они могут использоваться как конечными пользователями для получения оперативной информации о погодных условиях, так и различными организациями для анализа климатических тенденций. В основе работы таких систем лежит взаимодействие с погодными *API*, обработка полученных данных и их визуализация в удобном формате.

Современные погодные дэшборды разрабатываются с учетом требований к высокой точности отображаемой информации, скорости обработки запросов и удобству использования. Они могут работать в различных средах – от веб-интерфейсов до консольных приложений. В зависимости от целей и задач системы, разработчики выбирают соответствующие языки программирования, инструменты разработки и методы анализа кода.

В данной главе рассматриваются основные теоретические аспекты, связанные с разработкой погодных дэшбордов. Также будут рассмотрена работа приложения погодного дэшборда в виде *UML*-диаграмм и описана работа *frontend* и *backend* частей приложения.

2.2 Моделирование *UML*-Диаграмм

Для проектирования архитектуры погодного дэшборда используются *UML*-диаграммы, которые позволяют визуализировать структуру и взаимодействие компонентов системы. В данном проекте будут построены следующие *UML*-диаграммы:

**Диаграмма вариантов использования (*Use*-*case*)** - описывает взаимодействие пользователя с системой, определяя основные функции и роли.

**Диаграмма активности (*Activity*)** - демонстрирует последовательность действий пользователя и потоки выполнения различных процессов.

**Диаграмма последовательности (*Sequence*)** - иллюстрирует порядок выполнения операций и взаимодействие объектов в рамках конкретных сценариев использования.

Эти диаграммы помогут структурировать проект и упростят процесс его разработки и дальнейшего сопровождения.

2.3 Спецификация *frontend* части программы

*Frontend*-часть погодного дэшборда представляет собой текстовый интерфейс, реализованный в консольном приложении. Основные функции *Frontend*:

1. Ввод данных пользователем - приложение запрашивает у пользователя необходимую информацию, такую как название города для получения прогноза погоды.
2. Отправка запросов на *Backend* - после ввода данных пользователем, *Frontend* формирует запрос и передает его *Backend*-части для получения актуальной информации о погоде.
3. Отображение информации - полученные от *Backend* данные о погоде, такие как температура, описание погодных условий и прогноз на несколько дней, выводятся в текстовом формате для пользователя.

*Frontend* использует стандартные средства ввода и вывода данных через консоль. Взаимодействие с *Backend*-частью осуществляется через вызов соответствующих методов для получения и обработки данных.

2.4 Спецификация *backend* части программы

*Backend*-часть погодного дэшборда отвечает за обработку запросов от *Frontend*, взаимодействие с внешним погодным *API* и формирование необходимого ответа.

Основные компоненты *Backend*:

1. Модуль взаимодействия с *API* - этот компонент отвечает за отправку *HTTP*-запросов к внешнему сервису погоды, получая информацию в формате *JSON*. Он формирует *URL*-адрес запроса, который включает параметры, такие как название города, единицы измерения и ключ *API*.
2. Модуль обработки данных - данный модуль занимается разбором полученного *JSON*-ответа, извлечением нужных данных (температуры, описания погоды и т. д.), а также их группировкой, например, по дням и временным промежуткам. Он форматирует информацию для удобного отображения на *Frontend*.
3. Модуль логики приложения - этот компонент управляет параметрами пользователя (например, название города), формирует запросы для *API* и обрабатывает ответы, передавая данные в нужном формате на *Frontend*.

*Backend* реализует логику взаимодействия с внешним сервисом, обработку данных и передачу информации в структуре, удобной для отображения на стороне *Frontend*. Основной технологический стек включает использование языка *Java* для работы с *HTTP*-запросами и *JSON*.

2.5 Выводы по разделу

В данной главе были рассмотрены ключевые теоретические аспекты разработки погодного дэшборда, включая его архитектуру, используемые технологии и спецификации. *UML*-диаграммы (*Use* *Case*, *Activity*, *Sequence*) помогут структурировать проект, определить взаимодействие компонентов и уточнить логику работы системы. Спецификация *Frontend* и *Backend* частей описывает основные модули системы, что обеспечивает гибкость разработки, возможность масштабирования и дальнейшего улучшения функционала.

Глава 3 Техническое решение

3.1 Общие сведения

Разработка погодного дэшборда включает создание двух основных частей системы: *Frontend* и *Backend*. *Frontend* отвечает за взаимодействие с пользователем, прием запросов и вывод данных о погоде в удобочитаемом формате. *Backend* обрабатывает запросы, получает данные о погоде с внешнего *API*, анализирует их и передает в *Frontend*. Основной целью системы является предоставление пользователю точных и своевременных данных о погодных условиях в его регионе. Для реализации проекта использовались *Java* и *Maven*, что обеспечило эффективную структуру для разработки и тестирования.

3.2 Выбор и обоснование использования зависимостей проекта

Для разработки системы были выбраны несколько ключевых зависимостей. В первую очередь, для обработки данных о погоде на *Backend* использована библиотека *Gson* (версии 2.8.8) для работы с *JSON*-форматом, который предоставляет *API* *OpenWeatherMap*. Для *Frontend* части использована та же библиотека для парсинга данных и отображения их пользователю в текстовом виде. Важно отметить, что использование зависимостей от *Maven* позволило упростить управление зависимостями, улучшить структуру проекта и облегчить его расширение в будущем. Также была включена зависимость от *backend* части проекта для взаимодействия между компонентами.

3.3 Описание разработки и результатов *Frontend* части

Полный код, разработанный для *Backend* части программы можно увидеть в Приложении Г.

*Frontend* часть системы предназначена для взаимодействия с пользователем, сбора входных данных и отображения результатов в удобном текстовом формате. Она реализована с использованием языка программирования *Java* и работает в командной строке. Основная задача *Frontend* — это запрос у пользователя города, формирование *URL* для *API*-запроса, отправка запроса на *Backend* и вывод полученных данных в терминал.

Процесс работы *Frontend* начинается с запроса у пользователя названия города:

// Запрашиваем название города у пользователя

System.out.print("Введите город: ");

String city = scanner.nextLine();

Листинг 1 – *frontend* часть программы с запросом названия города у пользователя

Этот ввод используется для формирования *URL*, который будет отправлен на *Backend* для получения данных о погоде. В коде используется *API* *OpenWeatherMap*, и *URL* выглядит следующим образом:

// Формируем URL для API с городом

/String apiUrl = "https://api.openweathermap.org/data/2.5/forecast?q=" + city + "&cnt=40&appid=f65beff8ef56a2a09fcc6b795bce048d&units=metric&lang=ru";

Листинг 2 – Форимрование *URL* для *API* с городом

После того как *URL* сформирован, *Frontend* передает его на *Backend* для получения прогноза погоды:

// Получаем данные о погоде с backend

String jsonResponse = WeatherBackend.getWeatherData(apiUrl);

Листинг 3 – Получение данных о погоде с *backend* части программы

Метод *getWeatherData* отправляет запрос на сервер *OpenWeatherMap* через *Backend*, где обрабатывается *JSON*-ответ. Если данные были успешно получены, они передаются в метод *parseWeatherData*, который разбивает их на удобные для отображения строки:

if (jsonResponse != null) {

// Обрабатываем полученные данные

String forecast = WeatherBackend.parseWeatherData(jsonResponse);

System.out.println(forecast); // Выводим прогноз

} else {

System.out.println("Не удалось получить данные о погоде.");

}

Листинг 4 – Обработка и вывод полученных данных с *backend* части программы

Результатом работы *Frontend* является вывод пользователю прогноза погоды в текстовом формате. В случае ошибки с получением данных выводится соответствующее сообщение. Также для корректной работы программы нужен *pom*.*xml* файл с существующими зависимостями, плагинами (См. Приложение Е).

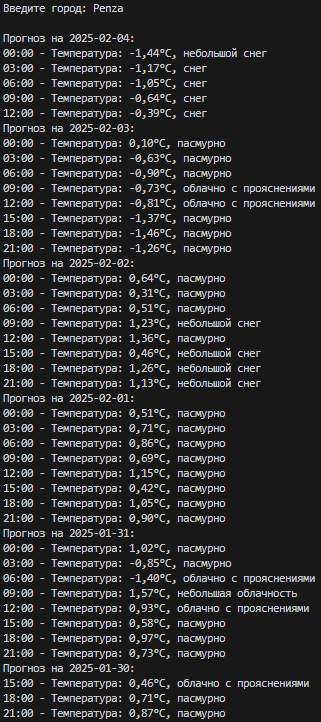


Рис. 1 – Результат работы *Frontend* части программы

Таким образом, *Frontend* часть системы обеспечивает простое взаимодействие с пользователем и корректно отображает данные о погоде, полученные с *Backend*.

3.4 Описание разработки и результатов *Backend* части

Полный код, разработанный для *Backend* части программы можно увидеть в Приложении Д.

*Backend* часть системы занимается обработкой запросов, полученных от *Frontend*, и взаимодействием с внешним *API* для получения данных о погоде. Основная цель *Backend* — это получить данные от *OpenWeatherMap*, обработать их и предоставить их *Frontend* в формате, удобном для отображения.

Первый этап работы *Backend* — это отправка *HTTP*-запроса на *OpenWeatherMap* *API*. Для этого используется класс *HttpURLConnection*, который позволяет установить соединение с сервером и получить ответ:

// Метод для получения данных с API

public static String getWeatherData(String urlString) {

try {

URL url = new URL(urlString);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String inputLine;

StringBuilder response = new StringBuilder();

while ((inputLine = in.readLine()) != null) {

response.append(inputLine);

}

in.close();

return response.toString();

} catch (Exception e) {

System.out.println("Ошибка запроса: " + e.getMessage());

return null;

}

}

Листинг 5 – Метод для получения данных по *API*-ключу

В этом методе происходит отправка *GET*-запроса по сформированному *URL* и чтение ответа от сервера. Ответ сохраняется в строку и передается в дальнейшем для обработки.

После получения данных, их необходимо обработать. *JSON*-ответ от *OpenWeatherMap* может содержать множество данных, таких как температура, описание погоды, время, дата и другие параметры. Для удобства работы с *JSON* используется библиотека *Gson*. В частности, в методе *parseWeatherData* данные из *JSON* распарсиваются и группируются по датам и временам:

// Метод для обработки данных о погоде и группировки их по дням и временным промежуткам

public static String parseWeatherData(String jsonResponse) {

StringBuilder forecastOutput = new StringBuilder();

try {

JsonObject jsonObject = JsonParser.parseString(jsonResponse).getAsJsonObject();

JsonArray list = jsonObject.getAsJsonArray("list");

// Карта для хранения прогноза по датам

Map<String, List<String>> dailyWeather = new HashMap<>();

// Создаем объект SimpleDateFormat для формата даты

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm");

// Устанавливаем временную зону UTC

dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));

for (JsonElement element : list) {

JsonObject weatherData = element.getAsJsonObject();

long dt = weatherData.get("dt").getAsLong();

// Получаем дату и время в формате yyyy-MM-dd HH:mm

String dateTime = dateFormat.format(new java.util.Date(dt \* 1000));

// Разделяем дату и время

String date = dateTime.split(" ")[0];

String time = dateTime.split(" ")[1];

// Получаем описание погоды и температуру

String weatherDescription = weatherData.getAsJsonArray("weather").get(0).getAsJsonObject().get("description").getAsString();

double temperature = weatherData.getAsJsonObject("main").get("temp").getAsDouble();

// Формируем строку прогноза для текущего времени

String forecast = String.format("%s - Температура: %.2f°C, %s", time, temperature, weatherDescription);

// Добавляем прогноз для текущего дня в карту

dailyWeather.computeIfAbsent(date, k -> new ArrayList<>()).add(forecast);

}

// Формируем вывод прогноза, сгруппированного по дням и времени

for (String date : dailyWeather.keySet()) {

forecastOutput.append("\nПрогноз на ").append(date).append(":");

for (String forecast : dailyWeather.get(date)) {

forecastOutput.append("\n").append(forecast);

}

}

} catch (JsonSyntaxException e) {

forecastOutput.append("Ошибка обработки данных: ").append(e.getMessage());

}

return forecastOutput.toString();

}

Листинг 6 - Метод для обработки данных о погоде и группировки их по дням и временным промежуткам

*Backend* часть не только получает данные от *API*, но и структурирует их таким образом, чтобы *Frontend* мог эффективно их отобразить. Также для корректной работы программы нужен *pom*.*xml* файл с существующими зависимостями, плагинами (См. Приложение Ё).

3.5 Тестирование

Процесс тестирования включал проверку каждой из частей системы как на уровне функциональности, так и на уровне производительности. Для *frontend* было важно убедиться в корректности ввода данных пользователем и правильности отображения прогноза погоды. Для *backend* проводились тесты на корректность обработки данных, а также на устойчивость к возможным ошибкам при запросах к внешнему *API*. Использовались такие инструменты, как *JUnit* для тестирования функционала, а также простые консольные проверки, чтобы удостовериться, что данные о погоде правильно выводятся на экран. Все компоненты были протестированы на реальных запросах к *OpenWeatherMap* *API*, что обеспечило точность и актуальность информации.

3.6 Выводы по разделу

В ходе разработки погодного дэшборда была создана система, состоящая из *Frontend* и *Backend* частей. *Frontend* обеспечивает взаимодействие с пользователем, отправляя запросы и выводя прогноз погоды в удобном формате. *Backend* получает данные с *OpenWeatherMap*, обрабатывает *JSON*-ответ и передает их обратно в *Frontend*.

Использование *Gson* упростило обработку *JSON*-данных, а *HttpURLConnection* позволил стабильно запрашивать данные с *API*. Проведенное тестирование подтвердило корректность работы системы и ее устойчивость к возможным ошибкам. В дальнейшем возможны улучшения, такие как интеграция с графическим интерфейсом и расширение функционала.

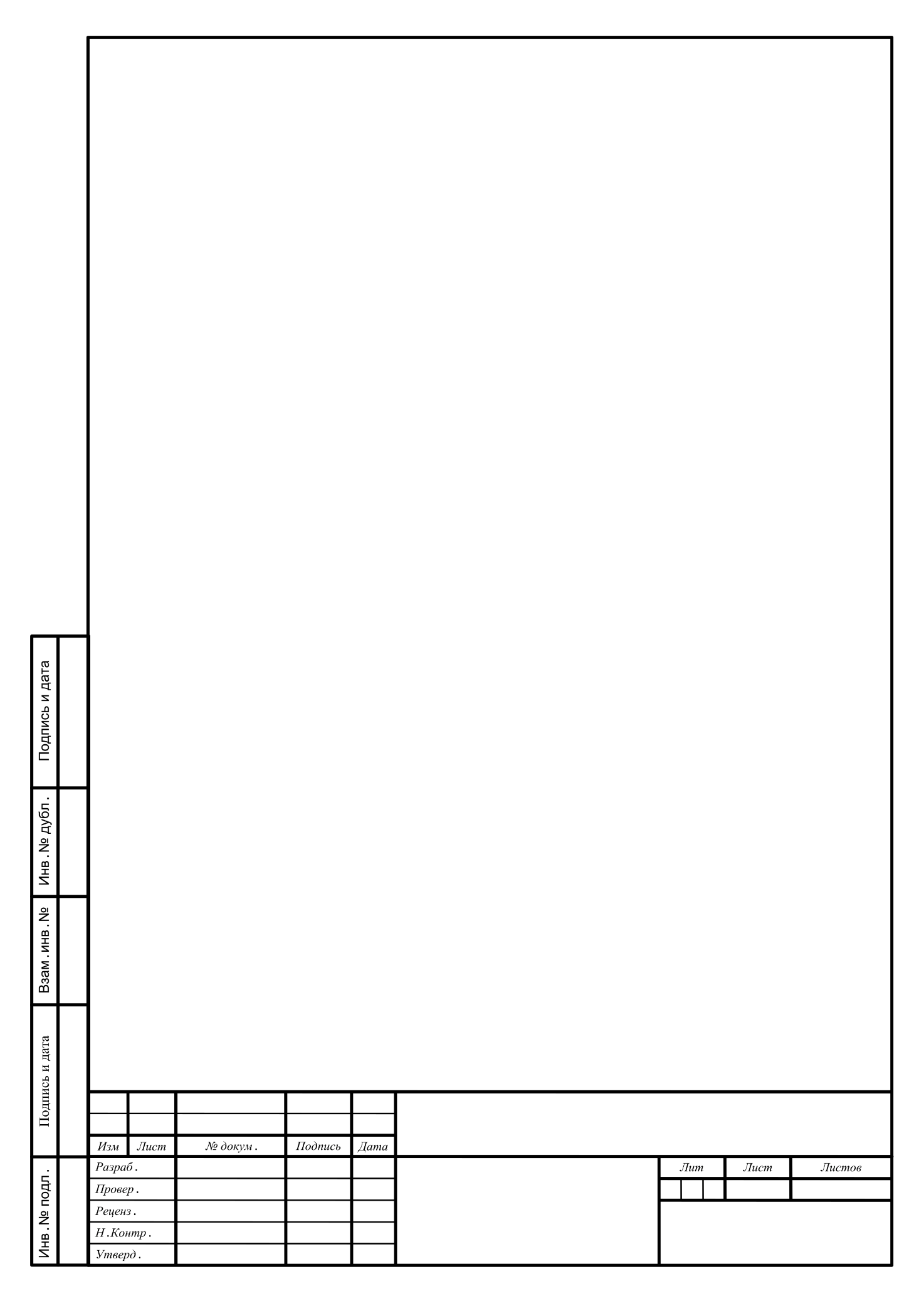
Заключение

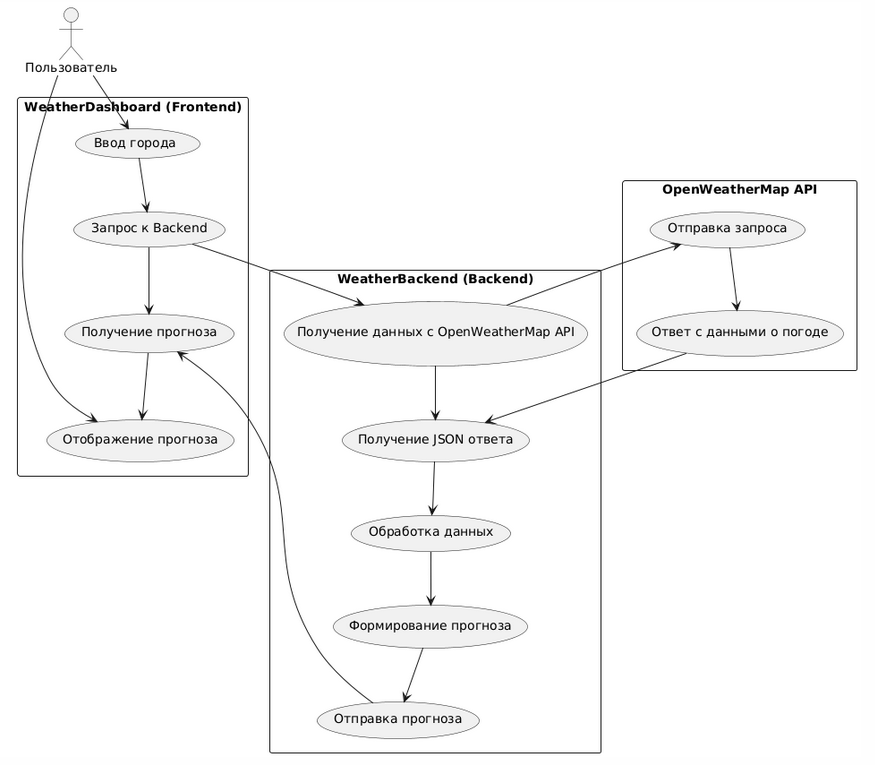
В рамках курсовой работы был разработан погодный дэшборд для отображения актуальных и прогнозируемых погодных условий через удобный интерфейс в терминале. Основной целью проекта было создание системы, позволяющей эффективно собирать, обрабатывать и отображать данные с *API* *OpenWeatherMap*. Архитектура включает *Frontend* для взаимодействия с пользователем и *Backend*, обрабатывающий запросы и структурирующий информацию. Для реализации использовались *Java*, библиотека *Gson* для работы с *JSON* и *Maven* для управления зависимостями, что обеспечило надежность и высокую производительность системы.

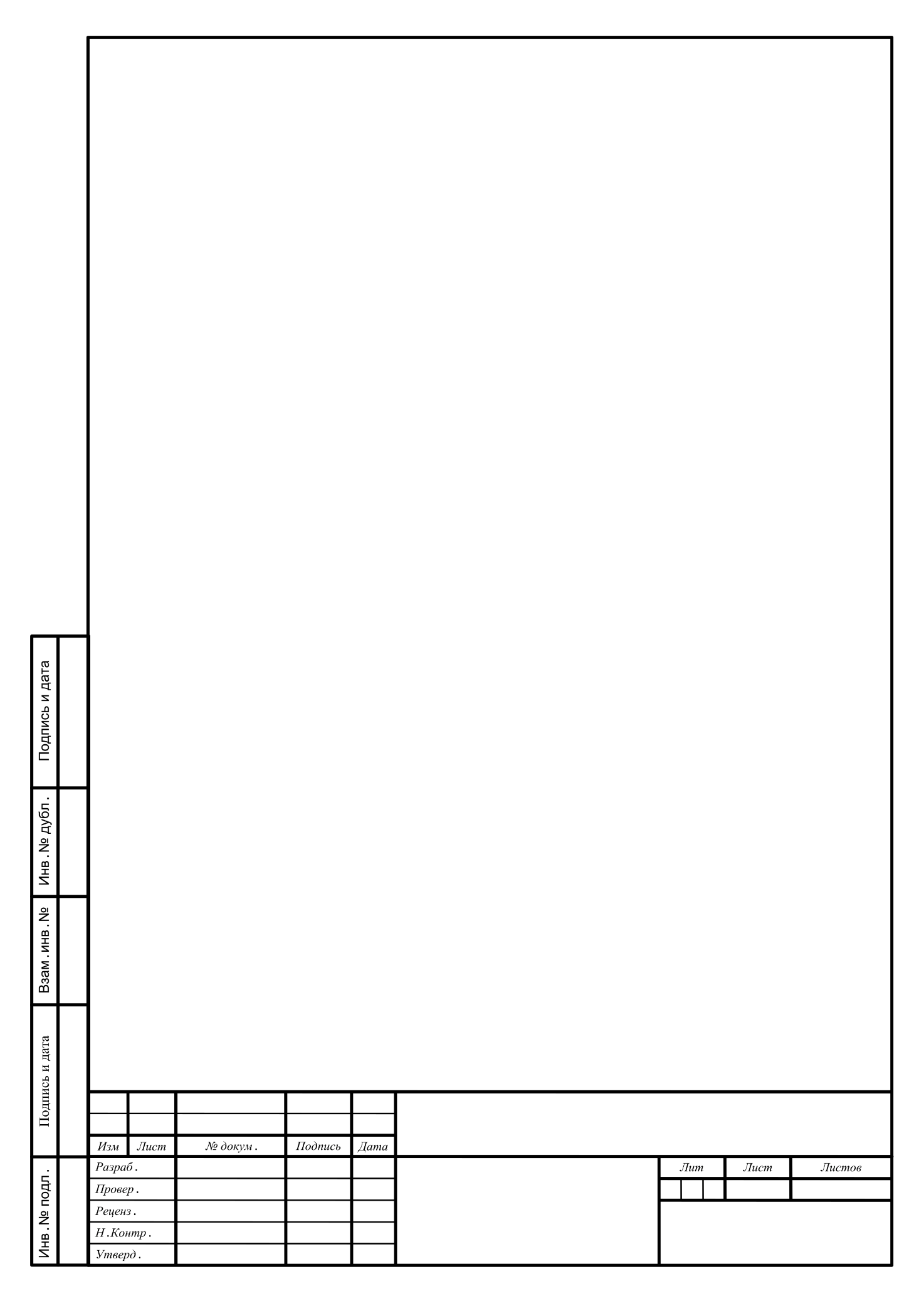
Тестирование с использованием *JUnit* подтвердило корректность работы всех компонентов: *Frontend* успешно выводит прогноз, а *Backend* стабильно извлекает и передает данные. Разработанный дэшборд отличается простотой использования, высокой точностью и возможностью работы без установки дополнительного ПО. В перспективе возможны расширения, такие как графический интерфейс, интеграция с другими источниками данных и улучшенные алгоритмы прогнозирования. Итогом работы стало создание эффективного инструмента для обработки и визуализации метеоданных.

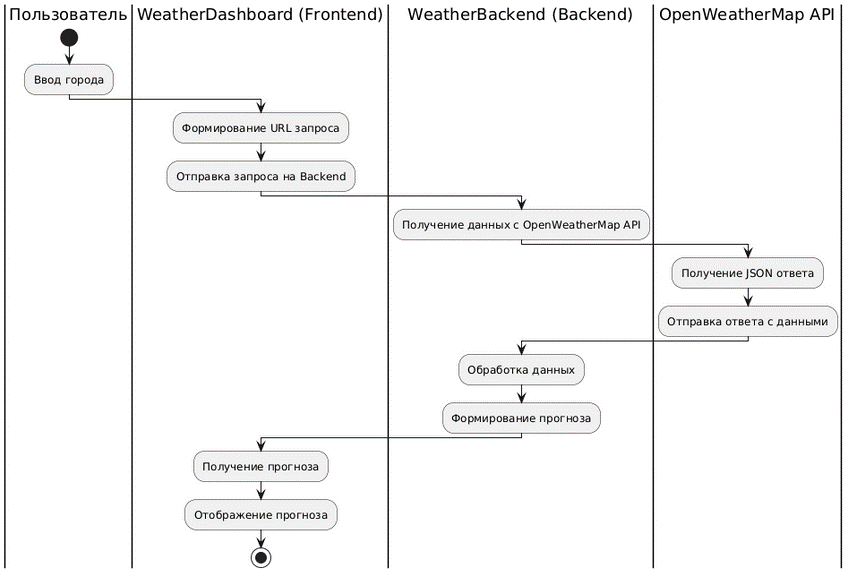
Библиографический список

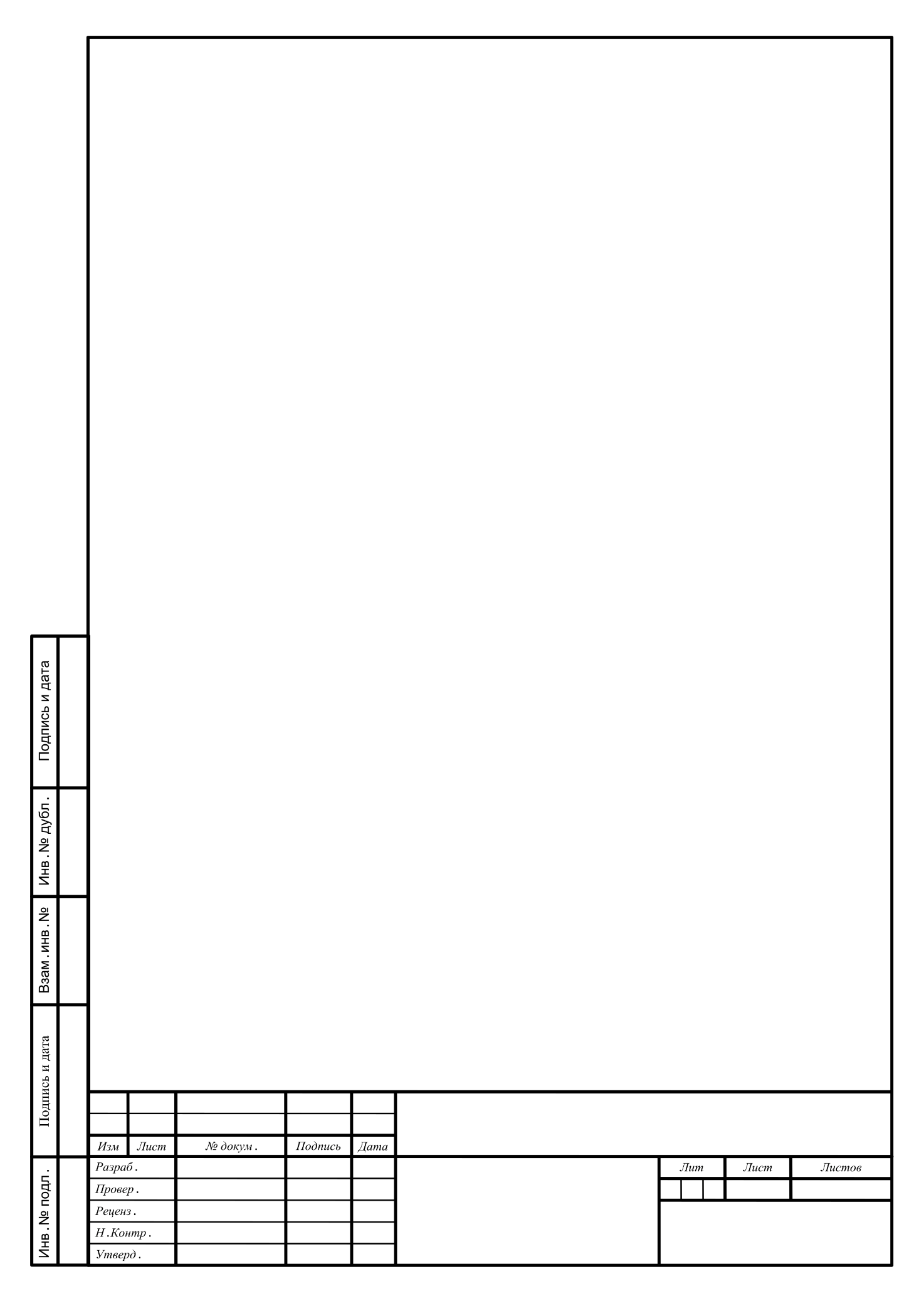
1. Гримм, Р. *C*++20 в деталях / Р. Гримм ; под редакцией А. Ю. Романова ; перевод с английского А. В. Борескова; под науч. ред. А. Ю. Романова, И. И. Романовой. — Москва : ДМК Пресс, 2023. — 518 с. — *ISBN* 978-5-97060-956-9. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/315479 (дата обращения: 28.10.2024). — Режим доступа: для авториз. пользователей.
2. Сергеева, О. А. Программирование на *Python* : учебно-методическое пособие / О. А. Сергеева. — Кемерово : КемГУ, 2024. — 157 с. — *ISBN* 978-5-8353-3123-9. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/420758 (дата обращения: 28.10.2024). — Режим доступа: для авториз. пользователей.
3. Крыжановская, Ю. А. Основы *JAVA* : учебное пособие / Ю. А. Крыжановская, В. Г. Ляликова, М. М. Безрядин. — Воронеж : ВГУ, 2020. — 96 с. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/433010 (дата обращения: 28.10.2024). — Режим доступа: для авториз. пользователей.
4. Ляликова, В. Г. Основные классы *Java* : учебное пособие / В. Г. Ляликова, Ю. А. Крыжановская, М. М. Безрядин. — Воронеж : ВГУ, 2020. — 80 с. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/433007 (дата обращения: 28.10.2024). — Режим доступа: для авториз. пользователей.
5. Курбатова, И. В. Основы программирования на языке *Java* : учебное пособие для спо / И. В. Курбатова, А. В. Печкуров. — Санкт-Петербург : Лань, 2024. — 348 с. — *ISBN* 978-5-507-48516-1. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/385925 (дата обращения: 29.10.2024). — Режим доступа: для авториз. пользователей.
6. Федоричев, Л. А. Реализация многопоточности в языке *Java* / Л. А. Федоричев, О. В. Букунова. — Санкт-Петербург : Лань, 2024. — 72 с. — *ISBN* 978-5-507-48153-8. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/367400 (дата обращения: 29.10.2024). — Режим доступа: для авториз. пользователей.
7. Завьялов, А. В. Диаграммы *UML* для анализа и проектирования информационных систем : учебно-методическое пособие / А. В. Завьялов. — Москва : РТУ МИРЭА, 2021. — 65 с. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/218630 (дата обращения: 30.10.2024). — Режим доступа: для авториз. пользователей.
8. Давыдовский, М. А. Проектирование программной системы в *UML* *Designer* : учебное пособие / М. А. Давыдовский, М. Н. Никольская. — Москва : РУТ (МИИТ), 2019. — 129 с. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/175651 (дата обращения: 30.10.2024). — Режим доступа: для авториз. пользователей.
9. Лоре, А. Проектирование веб-*API* : руководство / А. Лоре ; перевод с английского Д. А. Беликова. — Москва : ДМК Пресс, 2020. — 440 с. — *ISBN* 978-5-97060-861-6. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/179498 (дата обращения: 10.11.2024). — Режим доступа: для авториз. пользователей.
10. Недашковский, В. М. Протокол удаленного вызова процедур *JSON* – *RPC* : учебно-методическое пособие / В. М. Недашковский, Д. А. Локтев. — Москва : МГТУ им. Н.Э. Баумана, 2019. — 20 с. — *ISBN* 978-5-7038-5301-6. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/205115 (дата обращения: 30.01.2025). — Режим доступа: для авториз. пользователей.
11. Заяц, А. М. Проектирование и разработка *WEB*-приложений. Введение в *frontend* и *backend* разработку на *JavaScript* и *node*.*js* : учебное пособие для вузов / А. М. Заяц, Н. П. Васильев. — 3-е изд., стер. — Санкт-Петербург : Лань, 2021. — 120 с. — *ISBN* 978-5-8114-7042-6. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/154380 (дата обращения: 13.11.2024). — Режим доступа: для авториз. пользователей.
12. Недашковский, В. М. Протокол удаленного вызова процедур *JSON* – *RPC* : учебно-методическое пособие / В. М. Недашковский, Д. А. Локтев. — Москва : МГТУ им. Н.Э. Баумана, 2019. — 20 с. — *ISBN* 978-5-7038-5301-6. — Текст : электронный // Лань : электронно-библиотечная система. — *URL*: *https*://*e*.*lanbook*.*com*/*book*/205115 (дата обращения: 30.01.2025). — Режим доступа: для авториз. пользователей.

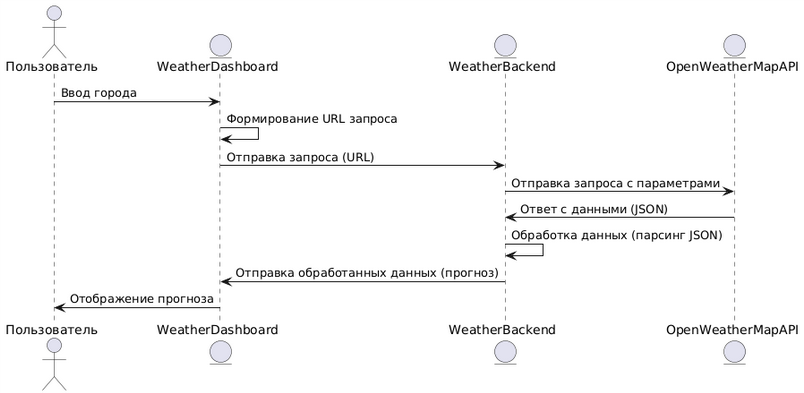
Приложение А – *UML*-Диаграмма программы (*Use*-*case*)



Приложение Б – *UML*- Диаграмма программы (*Activity*)



Приложение В - *UML*- Диаграмма программы (*Sequence*)



Приложение Г – *Frontend* часть программы

package com.example.frontend;

import java.util.Scanner;

import com.example.backend.WeatherBackend;

public class WeatherDashboard {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Запрашиваем название города у пользователя

        System.out.print("Введите город: ");

        String city = scanner.nextLine();

        // Формируем URL для API с городом

        String apiUrl = "https://api.openweathermap.org/data/2.5/forecast?q=" + city + "&cnt=40&appid=f65beff8ef56a2a09fcc6b795bce048d&units=metric&lang=ru";

        // Получаем данные о погоде с backend

        String jsonResponse = WeatherBackend.getWeatherData(apiUrl);

        if (jsonResponse != null) {

            // Обрабатываем полученные данные

            String forecast = WeatherBackend.parseWeatherData(jsonResponse);

            System.out.println(forecast);  // Выводим прогноз

        } else {

            System.out.println("Не удалось получить данные о погоде.");

        }

        scanner.close();

    }

}

Приложение Д – *Backend* часть программы

package com.example.backend;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.TimeZone;

import com.google.gson.JsonArray;

import com.google.gson.JsonElement;

import com.google.gson.JsonObject;

import com.google.gson.JsonParser;

import com.google.gson.JsonSyntaxException;

public class WeatherBackend {

    private static final String API\_KEY = "f65beff8ef56a2a09fcc6b795bce048d";  // Ваш API ключ

    // Метод для получения данных с API

    public static String getWeatherData(String urlString) {

        try {

            URL url = new URL(urlString);

            HttpURLConnection connection = (HttpURLConnection) url.openConnection();

            connection.setRequestMethod("GET");

            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

            String inputLine;

            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {

                response.append(inputLine);

            }

            in.close();

            return response.toString();

        } catch (Exception e) {

            System.out.println("Ошибка запроса: " + e.getMessage());

            return null;

        }

    }

    // Метод для обработки данных о погоде и группировки их по дням и временным промежуткам

    public static String parseWeatherData(String jsonResponse) {

        StringBuilder forecastOutput = new StringBuilder();

        try {

            JsonObject jsonObject = JsonParser.parseString(jsonResponse).getAsJsonObject();

            JsonArray list = jsonObject.getAsJsonArray("list");

            // Карта для хранения прогноза по датам

            Map<String, List<String>> dailyWeather = new HashMap<>();

            // Создаем объект SimpleDateFormat для формата даты

            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm");

            // Устанавливаем временную зону UTC

            dateFormat.setTimeZone(TimeZone.getTimeZone("UTC"));

            for (JsonElement element : list) {

                JsonObject weatherData = element.getAsJsonObject();

                long dt = weatherData.get("dt").getAsLong();

                // Получаем дату и время в формате yyyy-MM-dd HH:mm

                String dateTime = dateFormat.format(new java.util.Date(dt \* 1000));

                // Разделяем дату и время

                String date = dateTime.split(" ")[0];

                String time = dateTime.split(" ")[1];

                // Получаем описание погоды и температуру

                String weatherDescription = weatherData.getAsJsonArray("weather").get(0).getAsJsonObject().get("description").getAsString();

                double temperature = weatherData.getAsJsonObject("main").get("temp").getAsDouble();

                // Формируем строку прогноза для текущего времени

                String forecast = String.format("%s - Температура: %.2f°C, %s", time, temperature, weatherDescription);

                // Добавляем прогноз для текущего дня в карту

                dailyWeather.computeIfAbsent(date, k -> new ArrayList<>()).add(forecast);

            }

            // Формируем вывод прогноза, сгруппированного по дням и времени

            for (String date : dailyWeather.keySet()) {

                forecastOutput.append("\nПрогноз на ").append(date).append(":");

                for (String forecast : dailyWeather.get(date)) {

                    forecastOutput.append("\n").append(forecast);

                }

            }

        } catch (JsonSyntaxException e) {

            forecastOutput.append("Ошибка обработки данных: ").append(e.getMessage());

        }

        return forecastOutput.toString();

    }

}

Приложение Е – *POM*.*XML* для *Frontend* части программы

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>weather-frontend</artifactId>

    <version>1.0-SNAPSHOT</version>

    <packaging>jar</packaging>

    <properties>

        <maven.compiler.source>21</maven.compiler.source>

        <maven.compiler.target>21</maven.compiler.target>

    </properties>

    <dependencies>

        <dependency>

            <groupId>com.google.code.gson</groupId>

            <artifactId>gson</artifactId>

            <version>2.8.8</version>

        </dependency>

        <dependency>

            <groupId>com.example</groupId>

            <artifactId>weather-backend</artifactId>

            <version>1.0-SNAPSHOT</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.8.1</version>

                <configuration>

                    <source>21</source>

                    <target>21</target>

                </configuration>

            </plugin>

            <plugin>

                <groupId>org.codehaus.mojo</groupId>

                <artifactId>exec-maven-plugin</artifactId>

                <version>3.5.0</version>

                <configuration>

                    <mainClass>com.example.frontend.WeatherDashboard</mainClass>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

Приложение Ё – *POM*.*XML* для *Backend* части программы

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>weather-backend</artifactId>

    <version>1.0-SNAPSHOT</version>

    <packaging>jar</packaging>

    <properties>

        <maven.compiler.source>21</maven.compiler.source>

        <maven.compiler.target>21</maven.compiler.target>

    </properties>

    <dependencies>

        <dependency>

            <groupId>com.google.code.gson</groupId>

            <artifactId>gson</artifactId>

            <version>2.8.8</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.8.1</version>

                <configuration>

                    <source>21</source>

                    <target>21</target>

                </configuration>

            </plugin>

            <plugin>

                <groupId>org.codehaus.mojo</groupId>

                <artifactId>exec-maven-plugin</artifactId>

                <version>3.5.0</version>

                <configuration>

                    <mainClass>com.example.backend.WeatherBackend</mainClass>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>