**Оглавление**

# Предметная область

Предметная область «Приложение для отслеживания погоды» охватывает:

1. *Предоставление актуальной информации о текущей погоде*

Приложение должно предоставлять точную и достоверную информацию о погоде в реальном времени, с возможностью отслеживания изменений погоды.

2. *Предоставление актуальной информации о прогнозе погоды*

Необходимо предоставлять точную и достоверную информацию о прогнозе погоды на несколько дней.

3. *Возможность отслеживать погоду в различных городах*

Пользователи должны иметь возможность добавлять избранные города, что полезно для людей с родственниками в других городах или для того, чтобы просто следить за погодой в разных регионах.

В связи с этим приложение, соответствующее данной предметной области, должно реализовывать следующий функционал:

1. *Получение текущей погоды*
2. *Почасовой прогноз погоды*
3. *Прогноз погоды не несколько дней*
4. *Визуализация климатических данных при помощи иконок, графиков и карт*

Информация о погоде должна быть представлена в удобной для пользователя форме, с яркими иконками, информативными графиками, и интуитивно понятным интерфейсом.

5. *Отслеживание погоды в нескольких местах*

Приложение должно позволять пользователям создавать список избранных мест, чтобы отслеживать погоду в разных городах и странах.

6. *Возможность сохранять и удалять избранные места*

Пользователи должны иметь возможность сохранять список избранных мест, чтобы быстро получить информацию о погоде в нужном им месте, без необходимости вводить местоположение каждый раз.

В качестве источников данных могут использоваться:

1. *API погодных служб, такие как OpenWeatherMap или WeatherAPI*
2. *Данные с метеорологических станций*
3. *Данные с метеорологических спутников*

Из-за разнообразия климатических данных, необходимо реализовать хранение и обработку следующих метеорологических характеристик:

- *Температура (текущая, минимальная, максимальная)*
- *Осадки (вид осадков, интенсивность, вероятность)*
- *Влажность*
- *Ветер (скорость, направление)*
- *Атмосферное давление*
- *Облачность*
- *Видимость*

## Схема базы данных

Для обеспечения соответствующих рассмотренной предметной области функций необходимо создать базу данных, которая будет содержать различную информацию о пользователях и об их избранных городах.

Информация о каждом пользователе должна включать в себя: идентификатор пользователя; имя пользователя и пароль. Все параметры обязательно должны быть указаны.

Об избранных городах необходимо хранить следующую информацию: идентификатор города; название города; идентификатор пользователя, добавившего данный город в избранное.

На основании проведенного анализа предметной области «Приложение для отслеживания погоды» можно выделить следующие сущности (таблица 1):

Таблица 1 – Список сущностей

| № | Сущность | Назначение |
|---|---|---|
| 1 | user_entity | Перечень пользователей, зарегистрированных в приложении |
| 2 | chosen_city | Перечень избранных городов пользователей |

Приведем описание атрибутов для каждой сущности в виде таблицы, выделим первичные и внешние ключи и неключевые атрибуты (таблицы 2 – 3).

Таблица 2 – Список атрибутов таблицы “user_entity”

| Ключевое поле | Атрибут | Назначение |
|---|---|---|
| ПК (первичный ключ) | id | Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждому пользователю. Значения автоматически генерируются СУБД при вставке новой записи в таблицу. |
| | username | |
| | password | |

Таблица 3 – Список атрибутов таблицы “chosen_city”

| Ключевое поле | Атрибут | Назначение |
|---|---|---|
| ПК (первичный ключ) | id | Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждому городу. Значения автоматически генерируются СУБД при вставке новой записи в таблицу. |
| | city_name | |
| ВК (внешний ключ) | user_id | |

На основании семантического описания предметной области и списка атрибутов из таблиц 1 – 3 опишем классы сущностей и их свойства, расставим существующие связи между ними и приведем обоснование типов связей. Результат представлен в таблице 4 и на рисунке 1.

Таблица 4 – Список связей

| № | Сущности, участвующие в связи | Тип связи | Обоснование |
|---|---|---|---|
| 1 | user_entity – chosen_city | 1:N | Каждый пользователь может добавить несколько избранных городов, но каждый избранный город может относиться только к одному пользователю |

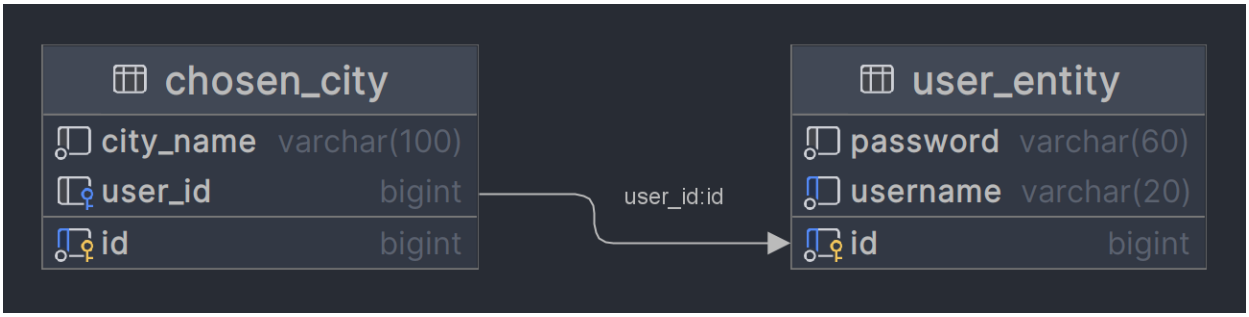


Рисунок 1 – Схема базы данных предметной области «Приложение для отслеживания погоды»

## Архитектура

Разработанное веб-приложение построено на основе архитектурного паттерна MVC. Таким образом оно будет разделено на три основных компонента:

- *Модель*. Содержит данные, логику обработки данных и правила бизнеса.
- *Представление*. Отображает данные модели для пользователя и позволяет взаимодействовать с приложением.
- *Контроллер*. Управляет взаимодействием между моделью и представлением, обрабатывая запросы пользователя и отправляя ответы.

Рассмотрим более подробно компоненты архитектуры приложения.

Модель содержит классы, представляющие климатические данные и данные о местоположении; репозитории, отвечающие за взаимодействие с базой данных для сохранения данных пользователей; а также сервисы, реализующие бизнес-логику приложения, такую как получение данных с внешнего API. Диаграмма информационных классов модели представлена на рисунке 2.

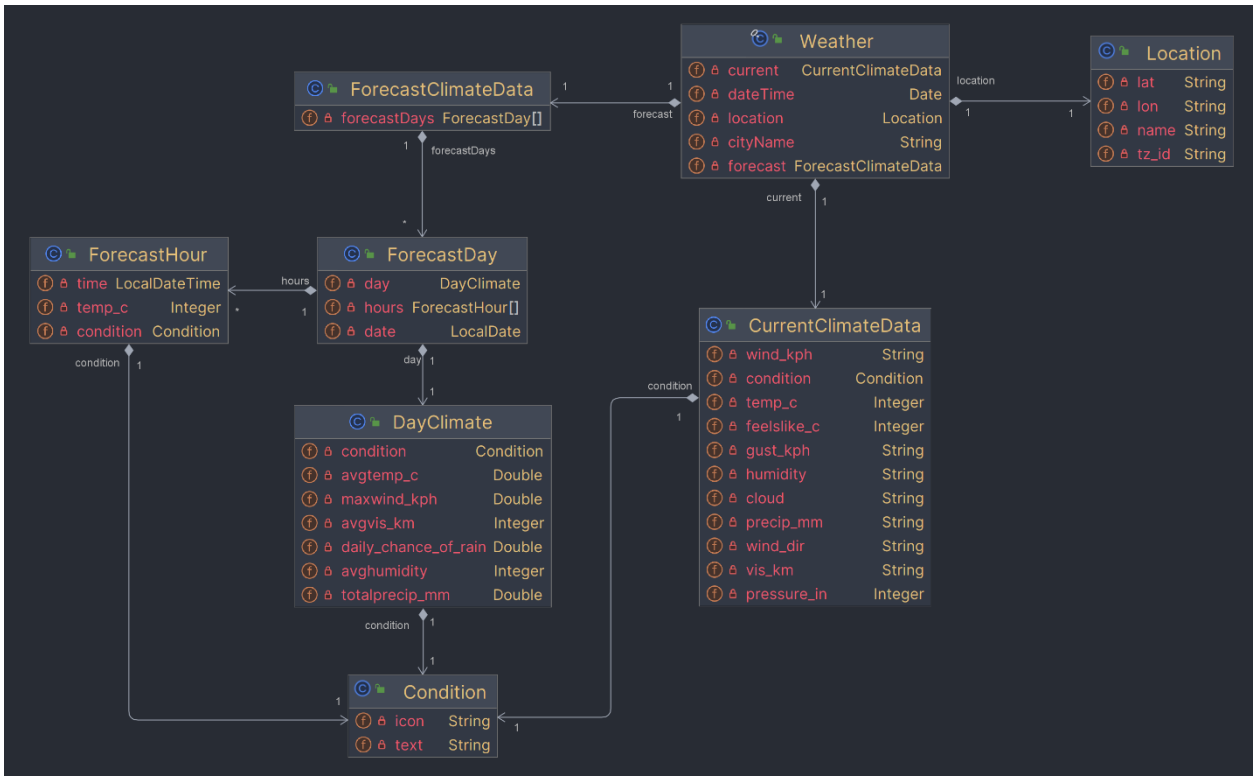


Рисунок 2 – Информационные классы модели

Представление включает в себя HTML-шаблоны и CSS-стили, определяющие структуру и внешний вид приложения; JavaScript-код, используемый для динамического взаимодействия с пользователем; шаблонизатор Thymeleaf, позволяющий отображать данных модели на веб-странице.

Контроллер отвечает за обработку пользовательских запросов, взаимодействие с моделью для подготовки данных, перенаправление запросов к конкретным представлениям, а также за обработку ошибок, которые могут возникнуть при взаимодействии пользователя с представлением.

Укрупненная схема приложения, соответствующая нотации IDEF0, представлена на рисунке 3.

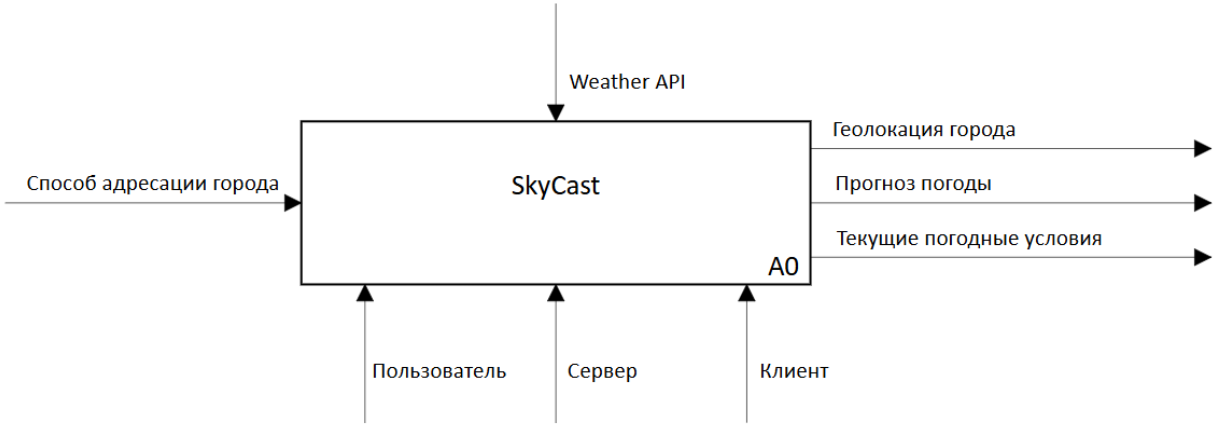


Рисунок 3 – Укрупненная схема приложения

При стандартном сценарии использования приложения имеет место следующее взаимодействие компонентов

1. Клиентская часть приложения формирует и обрабатывает запрос пользователя и отправляет его на сервер
2. Серверная часть получает и обрабатывает запрошенные климатические данные, а затем передает их обратно на клиент
3. Клиент выводит полученные климатические данные

Данное взаимодействие отражено на схеме IDEF0 для компонента A0 (рисунок 4).

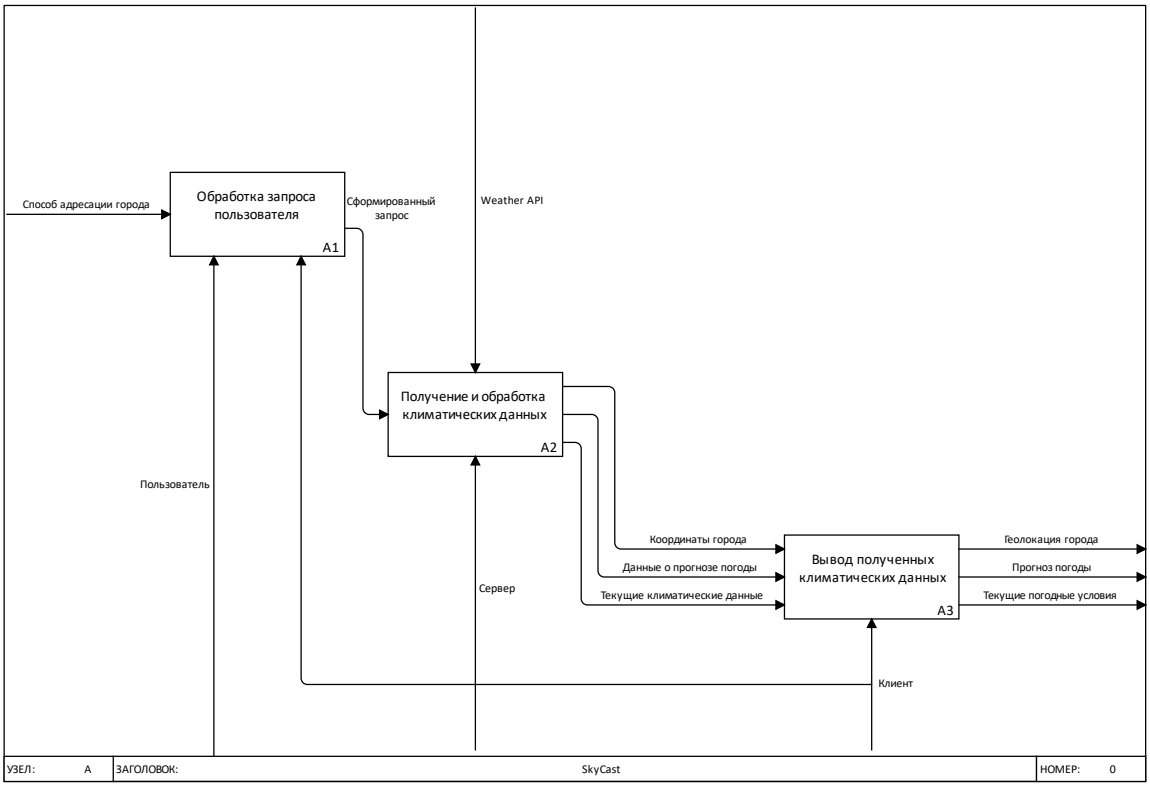


Рисунок 4 – Компонент A0

Рассмотрим модули схемы более подробно:

1. *Обработка запроса пользователя (рисунок 5)*

a. Пользователь выбирает способ адресации города: ввод названия или выбор из избранного

b. Клиентская часть приложения формирует запрос к серверу на основе выбранного названия города

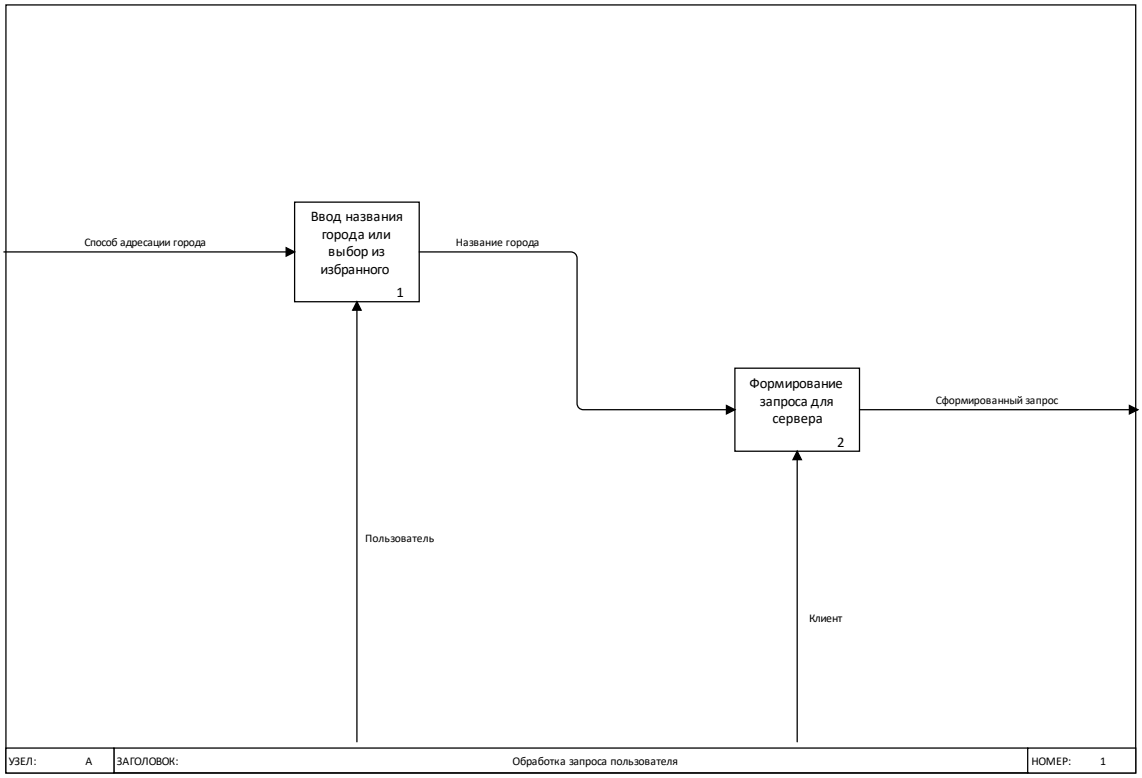


Рисунок 5 – Компонент A1

2. *Получение и обработка климатических данных (рисунок 6)*

a. Сформированный пользователем запрос поступает на контроллер, который инициирует получение климатических данных из внешнего API

b. Полученные необработанные климатические данные парсятся сервисным слоем и передаются в модель

c. Данные модели поступают на контроллер

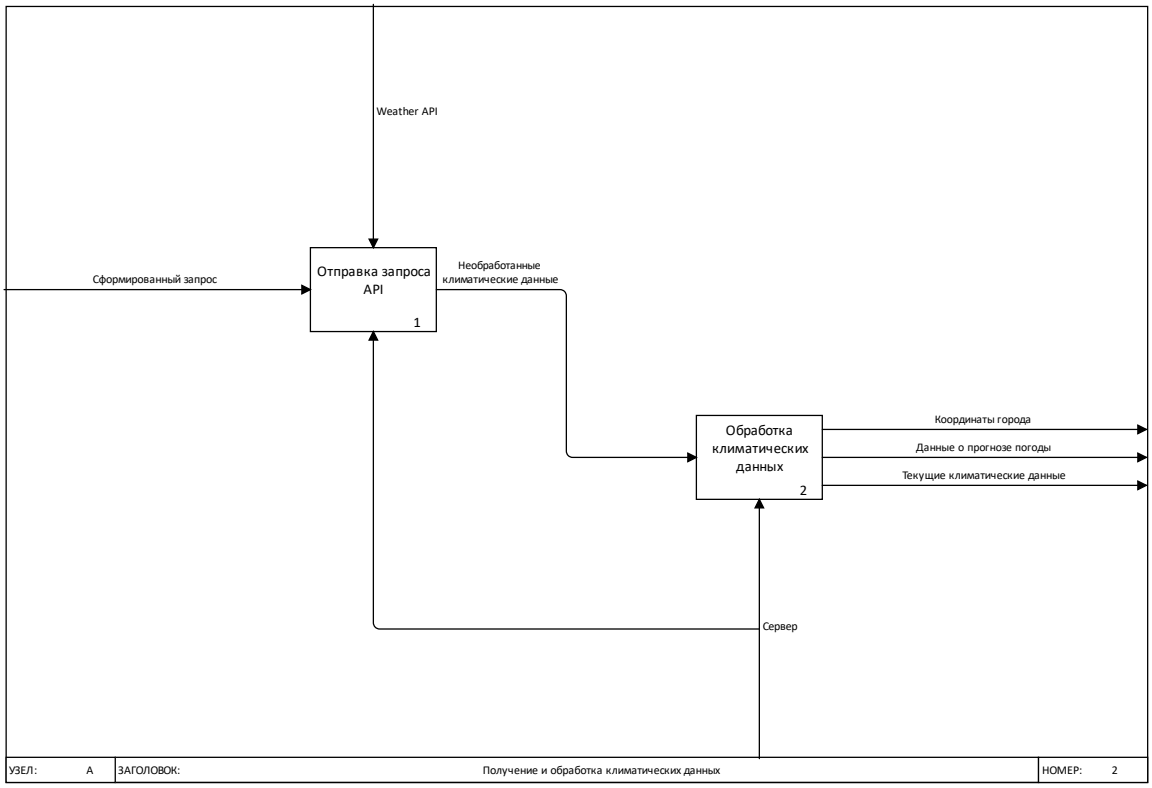


Рисунок 6 – Компонент A2

3. *Вывод полученных климатических данных (рисунок 7)*

a. Контроллер перенаправляет запрос к конкретному представлению

b. Представление отображает полученные данные

Описанные этапы взаимодействия компонентов приложения показаны на диаграмме IDEF0 (изображения на слайдах).

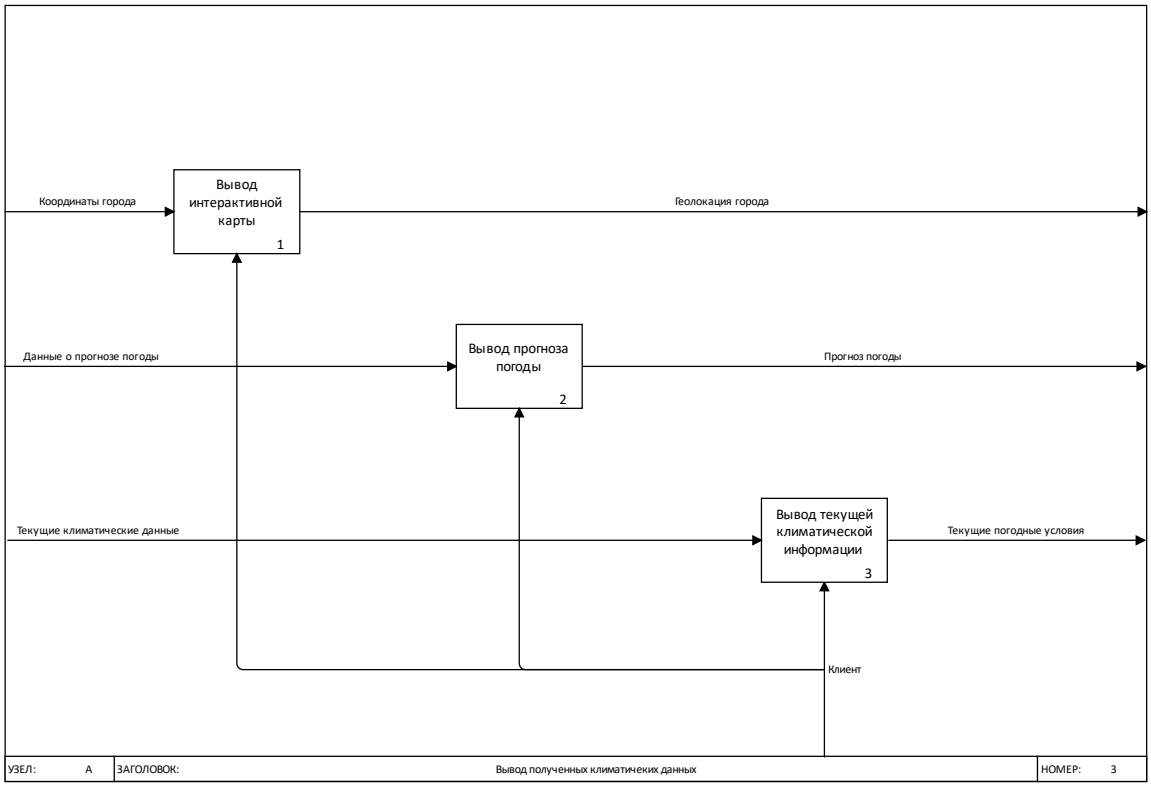


Рисунок 7 – Компонент А3

## Используемые технологии

При backend-разработке использовались следующие технологии:

- *Java 21*
- *Spring Boot* – предоставляет удобные инструменты для конфигурации, автоконфигурации, запуска и развертывания приложений
- *Spring Boot DevTools* – ускоряет процесс разработки за счет автоматического перезапуска сервера, а также LiveReload (позволяет мгновенно обновлять страницу в браузере)
- *Spring Web* – предоставляет инструменты для создания MVC веб-приложений
- *Spring Security* – отвечает за аутентификацию и авторизацию пользователей, также ограничивает доступ пользователей к ресурсам
- *Spring Validation* – используется для валидации данных, проверяет правильность ввода и позволяет обрабатывать ошибки

- *Spring Data JPA* – абстракция над JPA, которая упрощает работу с базой данных. Включает в себя Hibernate ORM
- *Project Lombok* – используется для генерации шаблонного кода
- *H2 Database* – in-memory БД, используемая при разработке и тестировании
- *PostgreSQL* – реляционная СУБД, предназначенная для промышленной развертки
- *Apache Tomcat* – встроенный веб-сервер, на котором запускается приложение
- *HikariCP* – пул соединений к базе данных
- *Log4J* – используется для записи событий, ошибок и отладочной информации в различных форматах
- *Logback* – настройка параметров логирования
- *Jackson* – сериализация и десериализация JSON; аннотации для конфигурации формата полей даты-времени, десериализуемых из JSON
- *OkHttp* – HTTP-клиент, использующийся для выполнения запросов к WeatherAPI

При тестировании использовались следующие технологии:

- *AssertJ* – написание утверждений в юнит-тестах и проверки результатов тестов
- *Junit 5* – фреймворк для юнит-тестирования
- *Mockito* – создание моков объектов в юнит-тестах

При frontend-разработке и веб-дизайне использовались следующие технологии:

- *HTML5*
- *CSS3*
- *JavaScript*
- *Thymeleaf* – создание шаблонов веб-страниц
- *Adobe Photoshop* – создание логотипа проекта

- *Figma*, *Pixso* – разработка макетов клиентской части

## Команда

Над проектом работала команда из трех человек, каждый из которых отвечал за определенный этап разработки и часть приложения. Роли были распределены следующим образом:

- Backend-разработка, тестирование – Сокол Илья
- Frontend-разработка, веб-дизайн – Лапин Кирилл
- Аналитика – Москвитин Дмитрий

Backend-разработчик занимался реализацией методов взаимодействия с внешним API, а также способов передачи обработанных на сервере данных на клиентскую часть. В области ответственности находилось разработка и подключение базы данных, хранящая данные о пользователях приложения и их избранных городах. Также backend-разработчик занимался настройки безопасности приложения, включающей в себя методы регистрации, авторизации и аутентификации. Основная задача backend-разработчика заключалась в реализации контроллеров и бизнес-логики, включающей в себя обработку данных с внешнего API. Было настроено полное логирование приложения, а также добавлена валидация и обработка ошибок данных форм клиентской части. В дополнение ко всему, была разработана модель, позволяющая передавать данные для отображения на устройстве пользователя.

Frontend-разработчик занимался реализацией клиентской части веб-приложения. В области ответственности находилась разработка макетов и верстка. В результате создания макетов был определен внешний вид приложения. После этого макеты были реализованы в коде с использованием HTML, CSS и JS. Для отображения полученных с серверной части данных был применен шаблонизатор, позволяющий представлять информацию в удобной форме при помощи генерации блоков, условных операторов, контекстных ссылок и атрибутов модели.

Аналитик отвечал за выбор предметной области, составление и согласование с остальными участниками команды технического задания и проектирование архитектуры приложения. Также производился анализ основных и побочных задач, составляющих процесс разработки проекта. На начальных этапах отвечал за проектирование функциональной модели приложения IDEF0, на основе которой в последствии велась разработка; а также за выбор стека технологий.

Команда работала по методологии Kanban, которая подразумевает предварительное планирование объема работы, перемещение задач согласно этапам рабочего процесса и возможность свободного их выполнения.

## Задачи, которые вызвали наибольший интерес

В процессе разработки наибольший интерес вызвала работа с API, включающая в себя парсинг JSON и отображение климатических данных на frontend.

Для удобства работы с API были добавлены дополнительные конфигурационные свойства, определяющие ключ API, а также не изменяющиеся параметры запроса. Обращения к API производились с помощью библиотеки OkHttp, предоставляющей простые и легкие в использовании инструменты для выполнения HTTP-запросов. Для парсинга JSON-объектов и JSON-массивов использовался встроенный функционал Java, а именно библиотека *org.json.* Для маппинга JSON на поля объектов использовалась библиотека Jackson, а в частности ObjectMapper.

Для отображения климатических данных на frontend был использован шаблонизатор Thymeleaf. Была проведена работа с условными операторами *th:if* и *th:unless*, генерацией блоков при помощи *th:each* и *th:error*, а также встроенными методами обработки числовой информации.

## Задачи, которые вызвали наибольшие трудности

Наибольшие трудности при разработке приложения вызвала реализация методов добавления и удаления избранных городов пользователей из базы данных. Так как в качестве ORM использовался Hibernate, необходимо было

учитывать специфические особенности работы с данной библиотекой, что приводило к некорректному изменению содержимого базы данных. При решении данных проблемы было потрачено достаточно много времени на изучение документации и статей с примерами реализации схожих функций. В итоге были написаны методы, осуществляющие корректное добавление и удаление избранных городов пользователей.

## Что сделано?

Разработано веб-приложение для отслеживания погоды, которое успешно реализует основные функции, обозначенные предметной областью.

В приложении реализованы:

- Получение актуальных климатических данных
- Отображение информации о погоде в удобной и интуитивно понятной форме
- Возможность добавления и удаления избранных городов

К преимуществам разработанного приложения можно отнести:

1. *Современный стек технологий*

При разработке использовались актуальные технологии разработки ПО, такие как Java 21, Spring Boot, Thymeleaf и PostgreSQL.

2. *Интуитивный интерфейс*

При разработке было рассмотрено множество различных способов представления информации, и был выбран наиболее подходящий вариант, позволяющий пользователю легко адаптироваться к приложению.

3. *Актуальные данные*

Получение климатических данных из надежного API обеспечивает точность и актуальность данных.

4. *Гибкая архитектура*

Применение архитектурного паттерна MVC обеспечивает простоту модификации, доработки и расширения функционала приложения.

К недостаткам приложения относятся:

1. *Ограниченная функциональность*

Некоторые функции, такие как сравнение погоды в разных местах, графики изменения метеорологических характеристик не были реализованы.

2. *Зависимость от API*

Приложение сильно зависит от конкретного API, что может привести к проблемам в случае недоступности API.

В целом, разработанное приложение является перспективным проектом, который может быть улучшен за счет доработки функционала, добавления мобильной версии, и решения проблем, связанных с зависимостью от API.