Введение

В последние годы наблюдается стремительный рост интереса к технологиям искусственного интеллекта и их применению для решения прикладных задач. Одним из наиболее динамично развивающихся направлений является создание интеллектуальных диалоговых систем, или чат-ботов, способных взаимодействовать с пользователем на естественном языке. Современные чат-боты эволюционировали от простых систем, работающих по жестко заданным сценариям, до сложных ассистентов, использующих модели обработки естественного языка (NLP) для понимания намерений пользователя и предоставления релевантной информации.

Особый интерес представляет применение таких систем в узкоспециализированных областях, таких как туризм, экология и образование. Предоставление точной, верифицированной и, что немаловажно, наглядной информации является ключевой задачей для повышения осведомленности и интереса к уникальным природным объектам. Байкальский регион, с его эндемичной флорой и фауной, представляет собой идеальную предметную область для применения подобных технологий. Традиционные источники информации, такие как справочники или веб-сайты, часто не обладают необходимой интерактивностью и не способны отвечать на комплексные, контекстуально-зависимые вопросы пользователей.

Разработка специализированного чат-бота, ориентированного на экосистему Байкала, позволяет решить эту проблему, предоставляя пользователю мощный инструмент для исследования региона. Однако создание такого ассистента сопряжено с рядом технических вызовов. Необходимо не только обеспечить точное распознавание запросов, но и реализовать эффективное взаимодействие с базами данных, содержащими разнородную информацию: текстовые описания, изображения и геопространственные данные. Современный подход к решению таких задач заключается в использовании гибридных архитектур, сочетающих сильные стороны классических NLU-фреймворков и больших языковых моделей (LLM).

Цель проекта разработать Telegram-бота "Эко-ассистент Байкала", способного предоставлять пользователям мультимедийную информацию (текст, изображения, интерактивные карты) о флоре и фауне Байкальского региона, основываясь на анализе запросов на естественном языке.
Задачи для достижения цели:

1. Создание структуры базы данных и хранилища файлов для мультимедийной информации, включая её метаописания.
2. Формирование множества признаков, которые будут описывать мультимедийную информацию.
3. Создание сценариев диалогов, которые являются комбинацией формата информации, набора признаков и шаблонов многоактовых диалогов.
4. Разработка требований к исходным данным, результатом которой станут требования к формату данных и их описанию с учётом множества признаков.
5. Разработка программных средств для извлечения данных (создание и тестирование парсеров для источников, таких как iNaturalist и Байкальский музей, получение геокоординат из справочников).
6. Разработка и программная реализация сценариев классификации мультимедийных данных согласно множеству признаков на основе расширяемого набора ИИ-моделей.
7. Разработка программных средств генерации мультимедийных данных с применением больших языковых моделей (на примере текстовых и геоданных), в том числе с использованием архитектуры RAG (Retrieval-Augmented Generation).
8. Заполнение базы данных и хранилища файлов.
9. Реализация сценариев диалогов для разных целевых платформ (Telegram).
10. Протестировать и оценить работоспособность и производительность реализованной системы.

1. Анализ

1.1 Диалоговые системы и их эволюция

Диалоговые системы, или чат-боты, представляют собой программные комплексы, предназначенные для имитации осмысленного разговора с пользователем. Исторически первые чат-боты, такие как ELIZA (1966), работали на основе простого сопоставления с шаблонами и не обладали реальным "пониманием" языка. С развитием технологий обработки естественного языка (Natural Language Processing, NLP) чат-боты стали значительно сложнее.

Современные системы можно условно разделить на две большие категории:

1. Декларативные (Rule-Based): работают по заранее определенным правилам и сценариям. Они предсказуемы, надежны и быстры, но их возможности строго ограничены заложенными в них скриптами. Такие боты часто используются в службах поддержки для ответов на типовые вопросы.

2. Интеллектуальные (AI-Based): используют машинное обучение и NLP для анализа и понимания запросов пользователя. Они способны распознавать намерения (intents), извлекать сущности (entities) и поддерживать более гибкий диалог.

1.2 Архитектура современных NLU-ассистентов

Ключевым компонентом интеллектуального чат-бота является модуль NLU (Natural Language Understanding). Его основная задача – преобразовать неструктурированный текст пользователя в структурированный формат, понятный машине. Стандартный NLU-пайплайн включает в себя несколько этапов:

1. Токенизация: Разбиение текста на отдельные слова или символы (токены).
2. Извлечение сущностей (Entity Extraction): Распознавание в тексте важных фрагментов, таких как имена, даты, географические названия (например, «кедр», «зимой», «Листвянка»).
3. Определение намерения (Intent Classification): Классификация всего запроса пользователя для определения его основной цели (например, получить картинку, получить информацию).

Популярным фреймворком для построения таких систем является Rasa Open Source. Он предоставляет инструменты для создания NLU-моделей и управления диалогом (Dialogue Management), позволяя разработчику полностью контролировать логику бота.

1.3 Большие языковые модели (LLM) в диалоговых системах

С появлением больших языковых моделей, таких как GPT-4 или GigaChat, произошел качественный скачок в возможностях диалоговых систем. В отличие от классических NLU-моделей, которые обучаются на конкретных примерах для распознавания ограниченного набора интентов и сущностей, LLM обладают "общим" пониманием языка.

В контексте чат-ботов LLM могут использоваться для решения двух основных задач:

1. Анализ и структурирование запроса: LLM может выступить в роли "универсального NLU-модуля". Ему можно дать на вход сырой текст пользователя и попросить вернуть структурированный JSON-объект с намерением и извлеченными параметрами. Этот подход обеспечивает невероятную гибкость, так как система способна понимать запросы, которые не были явно предусмотрены в обучающих данных.
2. Генерация ответа: LLM может генерировать человекоподобные текстовые ответы, что особенно полезно для фоллбэк-сценариев, когда в основной базе знаний нет готового ответа.

1.4 Архитектура с переключаемыми пайплайнами обработки

Для решения поставленных задач в рамках проекта "Эко-ассистент Байкала" была спроектирована и реализована архитектура, основанная на двух независимых, переключаемых конвейерах (режимах) обработки запросов. Такой подход позволяет пользователю самому выбирать способ взаимодействия с системой в зависимости от его потребностей, сочетая сильные стороны различных технологий.

Обоснование наличия двух режимов заключается в следующем: режим на основе LLM (GigaChat) является основным и наиболее гибким, тогда как режим Rasa выступает в качестве резервного. Он может быть задействован в случае недоступности LLM по различным причинам, например, при отсутствии финансирования для использования платных API, что обеспечивает непрерывность работы системы.

Система включает в себя следующие режимы:

1. Режим Rasa: В данном режиме вся обработка запроса делегируется классическому NLU-фреймворку Rasa Open Source. Этот процесс оптимизирован для распознавания заранее определенных, структурированных команд. Пользовательский ввод проходит через NLU-модель, которая извлекает намерение (intent) и сущности (entities), после чего система выполняет соответствующее действие (action). Этот режим обеспечивает высокую скорость и предсказуемость для стандартных и частотных запросов.

2. Режим GigaChat: В этом режиме система использует большую языковую модель (LLM) GigaChat для семантического анализа запроса. Вместо того чтобы сопоставлять запрос с заранее определенными шаблонами, основной бот отправляет сырой текст пользователя в LLM со специальным системным промптом. Задача LLM – не сгенерировать ответ, а преобразовать неструктурированный запрос в структурированную JSON-команду, которую затем исполняет бот. LLM определяет намерения пользователя согласно заранее заданным классам (шаблонам) и извлекает параметры для решения задачи, то есть осуществляет заполнение слотов в рамках заданных фреймов. Этот подход обеспечивает высокую гибкость и позволяет понимать сложные, вариативные формулировки на естественном языке.

Пользователь может в любой момент переключиться между этими двумя режимами через меню настроек бота. Несмотря на различие в логике обработки, оба пайплайна обращаются к единому бэкенд-сервису для получения фактических данных (текстов, изображений, карт) и могут использовать общий микросервис для фоллбэк-сценариев. Такая архитектура позволяет экспериментально сравнивать эффективность двух различных подходов к созданию диалоговых систем в рамках одного приложения.

1.5 Проблема и актуальность разработки

Проблема**:** Существующие источники информации о флоре и фауне Байкальского региона (веб-сайты, справочники, общие ассистенты) обладают рядом фундаментальных недостатков. Они либо неинтерактивны, либо не обладают специализированной и верифицированной базой данных, предоставляя общую или неточную информацию (как общие ассистенты, склонные к "галлюцинациям"). Отсутствует единый инструмент, который бы сочетал глубину специализированных знаний с гибкостью современного диалогового интерфейса, ориентированного на мультимедийный контент, такой как генерация интерактивных карт ареалов видов, предоставление фотогалерей по сложным запросам или отображение таксономических связей в графическом виде. Ключевая проблема заключается в обработке научно-обоснованных мультимедийных данных с учетом контекста диалога.

Актуальность: Разработка "Эко-ассистента Байкала" является актуальной, поскольку напрямую решает обозначенную проблему. Актуальность проекта заключается в необходимости внедрения современных технологий взаимодействия, таких как цифровые помощники, для потенциальных посетителей Байкальского музея и всех интересующихся регионом. Проект предлагает не просто чат-бота, а полноценный инструмент для исследования экосистемы Байкала, который позволяет не только получать текстовую информацию, но и взаимодействовать с мультимедийными данными в интерактивном режиме. Ценность проекта подкрепляется его высоким потенциалом в сферах туризма, образования и эко-просвещения.

2 Обзор существующих программных средств

Для определения уникальности и актуальности разрабатываемого "Эко-ассистента Байкала" необходимо провести анализ существующих на рынке программных решений, которые частично или полностью пересекаются с его функциональностью. Анализ будет проведен по трем ключевым категориям: общие интеллектуальные ассистенты, специализированные приложения-определители и традиционные информационные чат-боты. Оценка будет проводиться по адаптированным критериям, релевантным для диалоговых систем.

2.1 Общие интеллектуальные ассистенты (Яндекс Алиса, Google Assistant)

Данная категория представляет собой наиболее технологически продвинутые диалоговые системы, способные поддерживать разговор на широкий круг тем.

2.1.1 Общая оценка интерфейса. Интерфейс является преимущественно голосовым, но также поддерживает текстовый ввод. Взаимодействие максимально приближено к естественному человеческому диалогу. Системы отлично справляются с поддержанием контекста и обработкой сложных, многосоставных предложений.

2.1.2 Объем и структура представленной информации. Объем информации практически неограничен, так как ассистенты используют для ответов всю проиндексированную сеть Интернет. Однако это является и их

недостатком: информация не является специализированной и часто представляет собой краткую выдержку из первого найденного источника (например, Википедии). Структура ответа, как правило, – это короткий текстовый блок, иногда сопровождаемый ссылкой или изображением из поиска. Мультимедийные возможности ограничены показом статичных картинок и не включают генерацию интерактивных карт по запросу.

2.1.3 Наличие и структура меню. Меню как таковое в классическом понимании отсутствует. Навигация осуществляется полностью через языковые команды. Существуют стандартные команды для вызова справки, но нет структурированных меню для навигации по узкоспециализированной предметной области.

2.1.4 Удобство форм для ввода информации. Основной формой ввода является текстовая строка или голосовой запрос. Благодаря использованию мощных LLM, эти системы обладают высочайшей гибкостью в понимании естественного языка, синонимов и различных формулировок одного и того же вопроса.

2.1.5 Возможность поиска информации. Поиск является основной функцией, но он носит общий характер. Ассистенты не подключены к специализированным, верифицированным базам данных, что в контексте научной информации о флоре и фауне может приводить к предоставлению неточной, устаревшей или откровенно ложной информации ("галлюцинациям").

2.1.6 Общий вывод по категории. Общие ассистенты предоставляют лучший на рынке опыт естественного диалога, но не могут служить надежным источником для получения специализированных знаний. Их функциональность не приспособлена для решения узких задач, таких как построение карт ареалов или поиск изображений по сложным признакам.

2.2 Специализированные приложения-определители (Picture This, iNaturalist)

Это мобильные приложения, основная задача которых – идентификация видов растений и животных по фотографии пользователя.

2.2.1 Общая оценка интерфейса. Интерфейс является графическим (GUI), а не диалоговым. Взаимодействие с пользователем происходит через элементы управления: кнопки, вкладки, экраны. Интерфейс интуитивно понятен для своей основной задачи – загрузки и анализа фотографий.

2.2.2 Объем и структура представленной информации. Эти приложения обладают огромными, хорошо структурированными и верифицированными базами данных. Информация по каждому виду включает таксономию, качественные фотографии, подробные описания и, в случае iNaturalist, карту с точками наблюдений от других пользователей. Данные представлены в формате карточек, что очень удобно для восприятия.

2.2.3 Наличие и структура меню. Навигация осуществляется через стандартные для мобильных приложений меню (например, нижняя панель вкладок), которые позволяют переключаться между функциями: идентификация, личная коллекция, карта и т.д.

2.2.4 Удобство форм для ввода информации. Основной "формой ввода" является камера или галерея устройства. Текстовый ввод используется только для поиска по названию вида и не предполагает обработки запросов на естественном языке. Задать вопрос вроде "какие хвойные деревья растут на берегу Байкала?" невозможно.

2.2.5 Возможность поиска информации. Поиск ограничен либо названием вида, либо идентификацией по фото. Сложные, многокритериальные запросы не поддерживаются.

2.2.6 Общий вывод по категории. Приложения-определители являются превосходным источником достоверной и хорошо структурированной информации. Однако они не являются диалоговыми системами и не способны обеспечить интерактивное исследование предметной области через вопросы на естественном языке.

2.3 Традиционные информационные чат-боты (банковские, справочные)

К этой категории относятся чат-боты, работающие по заранее заданным сценариям и правилам, часто встречающиеся на сайтах компаний для поддержки клиентов.

2.3.1 Общая оценка интерфейса. Интерфейс строго функционален, часто основан на кнопочных меню. Диалог ощущается как "механический" и негибкий.

2.3.2 Объем и структура представленной информации. Объем информации строго ограничен базой знаний, заложенной разработчиками. Ответы представляют собой заранее написанные текстовые блоки.

2.3.3 Наличие и структура меню. Меню является основным элементом навигации. Оно имеет древовидную структуру и позволяет пользователю предсказуемо перемещаться по доступным опциям.

2.3.4 Удобство форм для ввода информации. Понимание естественного языка у таких ботов крайне ограничено или отсутствует. Они хорошо распознают ключевые слова ("доставка", "цена"), но любой отход от ожидаемого формата запроса приводит к ошибке "я вас не понял".

2.3.5 Возможность поиска информации. Поиск основан на простом сопоставлении с ключевыми словами или навигации по базе знаний через меню.

2.3.6 Общий вывод по категории. Традиционные чат-боты надежны и быстры в рамках своих узких, заранее определенных задач. Однако они абсолютно не подходят для роли исследовательского инструмента, так как не обладают гибкостью для понимания разнообразных вопросов пользователя.

2.4 Общий вывод

Проведенный анализ существующих программных средств показывает, что на рынке отсутствует решение, которое бы объединяло сильные стороны всех рассмотренных категорий. Общие ассистенты предлагают гибкий диалог, но не имеют доступа к специализированной и достоверной базе данных. Приложения-определители обладают такой базой, но лишены диалогового интерфейса. Традиционные чат-боты слишком ограничены в своих возможностях. Ценность и новизна "Эко-ассистента Байкала" заключается в том, что он спроектирован для заполнения именно этой ниши. Он сочетает гибкость понимания естественного языка, характерную для современных ассистентов, со специализированной, верифицированной базой знаний, дополняя это уникальными мультимедийными возможностями, такими как генерация интерактивных карт по запросу. Это делает его не просто очередным чат-ботом, а полноценным интерактивным инструментом для исследования.

3 Процесс AS IS vs TO BE

Для детального обоснования актуальности и ценности разрабатываемого "Эко-ассистента Байкала" был проведен анализ бизнес-процессов получения специализированной информации пользователем. С помощью нотации BPMN 2.0 (Business Process Model and Notation) были смоделированы два состояния процесса: текущее, до внедрения системы (AS IS), и целевое, после ее внедрения (TO BE). Сравнение этих моделей наглядно демонстрирует решаемые проблемы и эффективность предлагаемого архитектурного решения.

3.1 Модель процесса AS IS: Существующий порядок получения информации

Текущий процесс получения пользователем мультимедийной информации об экосистеме Байкала является фрагментированным, неструктурированным и возлагает основную когнитивную нагрузку на самого пользователя. Диаграмма процесса AS IS (Рисунок 1 – BPMN диаграмма процесса AS IS) иллюстрирует данный подход.

3.2 Описание диаграммы

Процесс, представленный на диаграмме, можно описать следующими этапами:

1. Инициация и выбор инструмента: процесс начинается с возникновения у пользователя информационной потребности. Ключевой проблемой на данном этапе является задача выбора релевантного инструмента. Пользователь вынужден самостоятельно принимать решение об использовании поисковых систем (Google, Yandex), общих интеллектуальных ассистентов (Яндекс Алиса), специализированных мобильных приложений-определителей (iNaturalist) или научных веб-сайтов. Выбор зависит от предполагаемого типа информации (текст, изображение, идентификация вида), что требует от пользователя предварительных знаний о возможностях и ограничениях каждого источника.

2. Поиск и фильтрация: на следующем этапе пользователь выполняет поиск и осуществляет ручную фильтрацию полученных результатов. Этот шаг сопряжен с анализом большого объема неструктурированных данных, включая общие статьи, блоги и форумы, которые часто не обладают необходимой верификацией.

3. Синтез данных: наиболее трудоемким и неэффективным этапом является задача ручного сопоставления и синтеза информации из различных, не связанных между собой источников. Для получения ответа на комплексный, многокритериальный запрос (например, "где рядом с мысом Хобой можно встретить цветущий эдельвейс?") пользователь должен самостоятельно объединить текстовые описания ареалов, данные с картографических сервисов и релевантные изображения.

4. Результат: исход процесса непредсказуем. Как показывает шлюз на диаграмме, поиск часто либо прекращается неудачей, либо приводит к получению неполного или неточного ответа, лишенного необходимой мультимедийной наглядности (например, интерактивной карты).

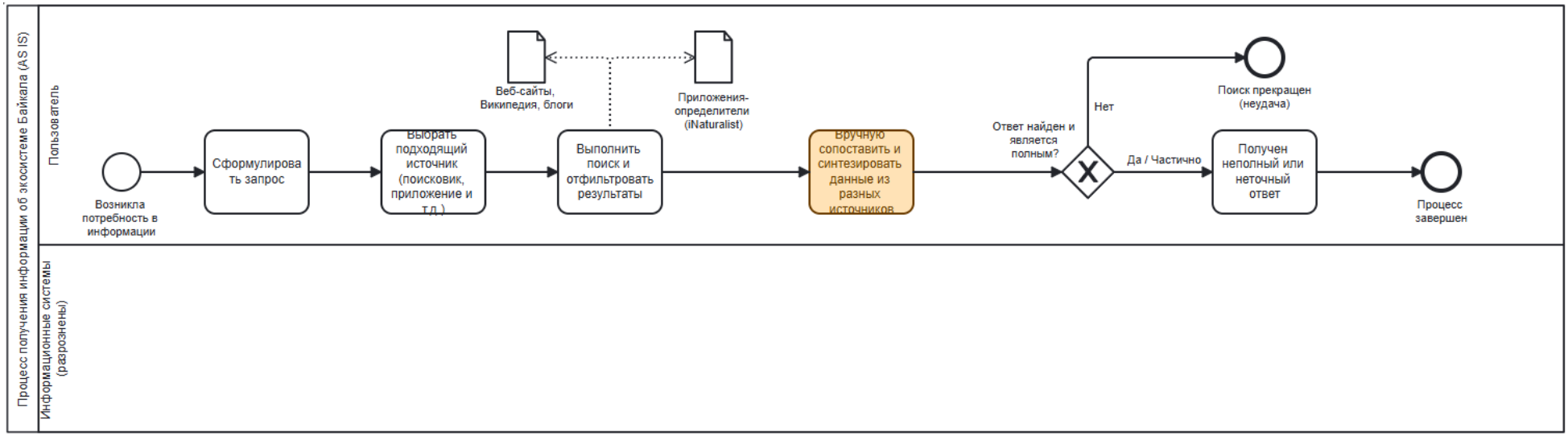

Рисунок 1 – BPMN диаграмма процесса AS IS

Таким образом, модель AS IS демонстрирует, что существующий процесс является неэффективным, требует от пользователя значительных временных затрат и экспертных навыков поиска, не гарантируя при этом получения достоверного и комплексного результата.

3.3 Модель процесса TO BE: Оптимизация с помощью «Эко-ассистента»

Разрабатываемый «Эко-ассистент Байкала» призван кардинально трансформировать описанный выше процесс, выступая в роли единой точки входа и автоматизированного инструмента для получения информации. Диаграмма процесса TO BE (Рисунок 2 – BPMN диаграмма процесса TO BE) моделирует новый, оптимизированный рабочий поток.

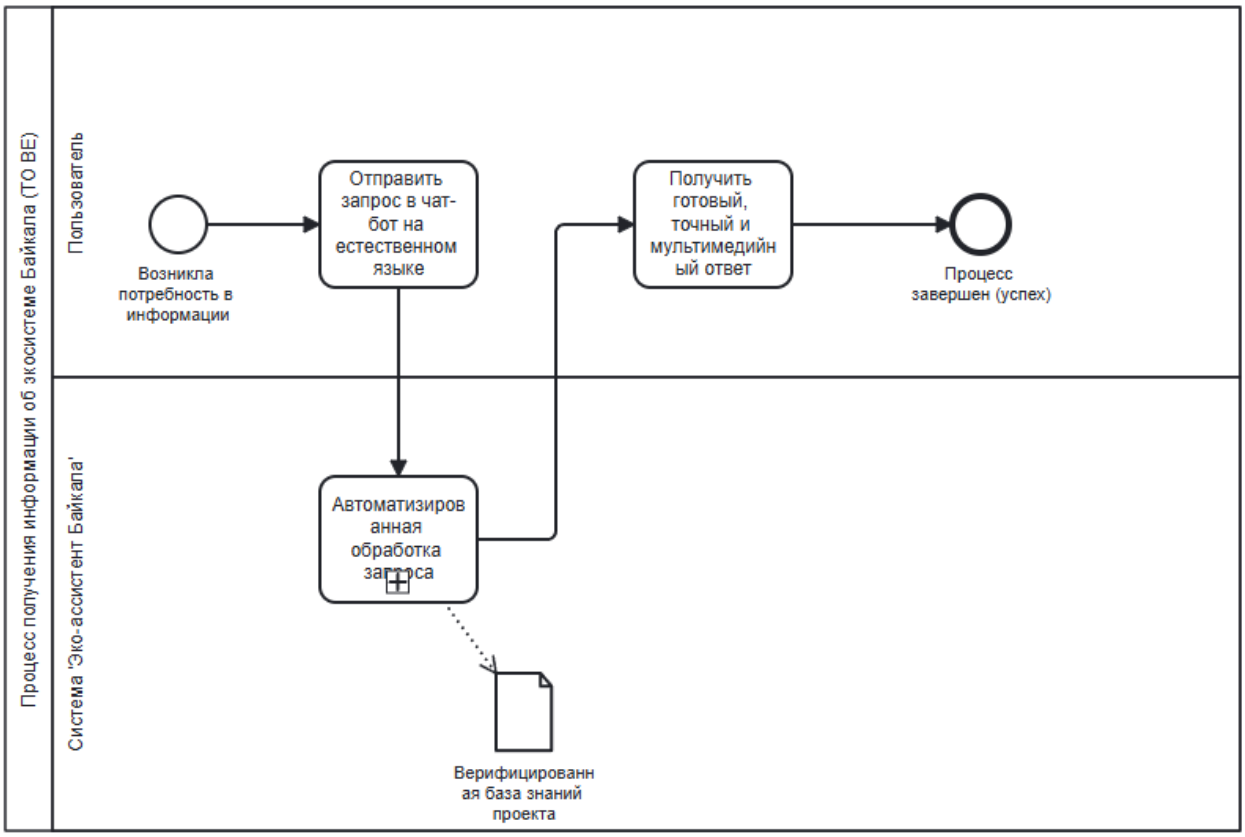

Рисунок 2 – BPMN диаграмма процесса TO BE

Ключевые изменения и преимущества нового процесса:

1. Упрощение действий пользователя: в модели TO BE единственными активными действиями пользователя являются формулировка запроса на естественном языке и получение готового ответа. Сложные этапы выбора источника, фильтрации и синтеза данных полностью исключены из его зоны ответственности.

2. Автоматизация обработки: Вся сложность процесса переносится на сторону системы. Подпроцесс "Автоматизированная обработка запроса" инкапсулирует в себе всю внутреннюю логику:

   a. Анализ запроса: Система, используя гибридную архитектуру NLU-фреймворка и большой языковой модели (LLM), точно распознает намерение и извлекает все параметры запроса.

   b. Взаимодействие с базой знаний: в отличие от поиска по всей сети Интернет, ассистент обращается к собственной специализированной и верифицированной базе данных, что гарантирует точность и достоверность информации.

с. Генерация мультимедийного ответа: Система автоматически агрегирует все необходимые компоненты (текстовое описание, изображения по заданным признакам, интерактивную карту) в единый, комплексный ответ.

3. Гарантированное завершение и определенность результата: в отличие от модели AS IS, где исход непредсказуем и часто приводит к прекращению поиска пользователем, процесс TO BE всегда завершается определенным и полезным для пользователя результатом. Даже в случае отсутствия информации в базе знаний, система не оставляет пользователя в неведении, а предоставляет четкий ответ. Это делает "Эко-ассистент" не просто чат-ботом, а полноценным и, что ключевое, надежным интерактивным инструментом для исследования региона.

3.4 Примеры сценариев использования в процессах AS IS и TO BE

Для иллюстрации практических различий между процессами AS IS и TO BE рассмотрим гипотетический, но репрезентативный сценарий. Предположим, пользователь (например, студент-биолог) пытается найти ответ на узкоспециализированный запрос: "Какие эндемичные мхи произрастают на скальных выходах острова Ольхон?"

Сценарий в рамках процесса AS IS:

1. Взаимодействие с поисковыми системами и общими ассистентами: при вводе данного запроса в поисковую систему или диалоге с общим ассистентом (например, Яндекс Алисой) пользователь получит набор неструктурированных ссылок. Результаты поиска будут включать общие статьи о флоре Ольхона, туристические форумы, возможно, научные публикации без прямого ответа и коммерческие предложения. Пользователь будет вынужден потратить значительное время на ручную фильтрацию и анализ источников, с высокой вероятностью, не найдя точного ответа. Результат: большие временные затраты при неопределенном исходе.

2. Взаимодействие с большими языковыми моделями (LLM) общего назначения: при обращении с тем же запросом к неспециализированной LLM (например, ChatGPT, GigaChat) существует значительный риск получения фактологически некорректного ответа, известного как "галлюцинация". Модель может сгенерировать правдоподобный, но вымышленный список видов (например, "Сфагнум байкальский") или неверно классифицировать существующие организмы. Результат: получение дезинформации, что подрывает доверие к технологии и может привести к ошибкам в дальнейшей работе пользователя.

Оба варианта в рамках процесса AS IS демонстрируют его ключевые недостатки: неэффективность и ненадежность.

Сценарий в рамках процесса ТО ВЕ (с использованием "Эко-ассистента"):

При обращении с тем же запросом к "Эко-ассистенту Байкала" система инициирует четкий и надежный рабочий поток, который приводит к одному из двух полезных для пользователя исходов:

1. Исход 1 (Данные найдены): система анализирует запрос, обращается к своей верифицированной базе знаний, находит релевантную информацию и предоставляет пользователю структурированный, мультимедийный ответ, включающий список видов, их описание и, возможно, карту с точками находок. Результат: быстрый и точный ответ.

2. Исход 2 (Данные не найдены): если в базе знаний отсутствует информация по столь специфическому запросу, система формирует честный и однозначный ответ, например: "В моей верифицированной базе знаний отсутствует информация об эндемичных мхах, произрастающих именно на скалах острова Ольхон".

Польза такого ответа заключается не в предоставлении искомых данных, а в создании информационной определенности. Пользователь экономит время, которое было бы потрачено на безрезультатный ручной поиск, и, что критически важно, защищен от дезинформации. Такой подход формирует доверие к системе как к надежному и честному источнику специализированных знаний, который четко осознает границы своей компетенции.

Процесс ТО ВЕ, реализуемый "Эко-ассистентом", трансформирует пользовательский опыт. Он заменяет неопределенность, ручной труд и риск получения ложной информации на скорость, точность и надежность. Даже в случае отсутствия данных, система предоставляет пользователю ценный результат – определенность и экономию времени, что является фундаментальным улучшением по сравнению с существующим процессом.

3.5 Заключение по анализу процессов

Сравнение моделей AS IS и ТО ВЕ наглядно доказывает актуальность и практическую значимость проекта. "Эко-ассистент Байкала" решает фундаментальную проблему фрагментации информации, автоматизируя рутинные и трудоемкие задачи пользователя и предоставляя ему мощный, надежный и удобный инструмент для взаимодействия с уникальной экосистемой Байкальского региона.

4 Описание вариантов использования

Для определения и формализации функциональных требований к системе "Эко-ассистент Байкала" была разработана диаграмма вариантов использования (Use Case Diagram) в соответствии со стандартом UML (Unified Modeling Language). Данная диаграмма описывает систему с точки зрения пользователя, отвечая на вопрос "что система должна делать?", и служит основой для дальнейшего проектирования и реализации функционала.

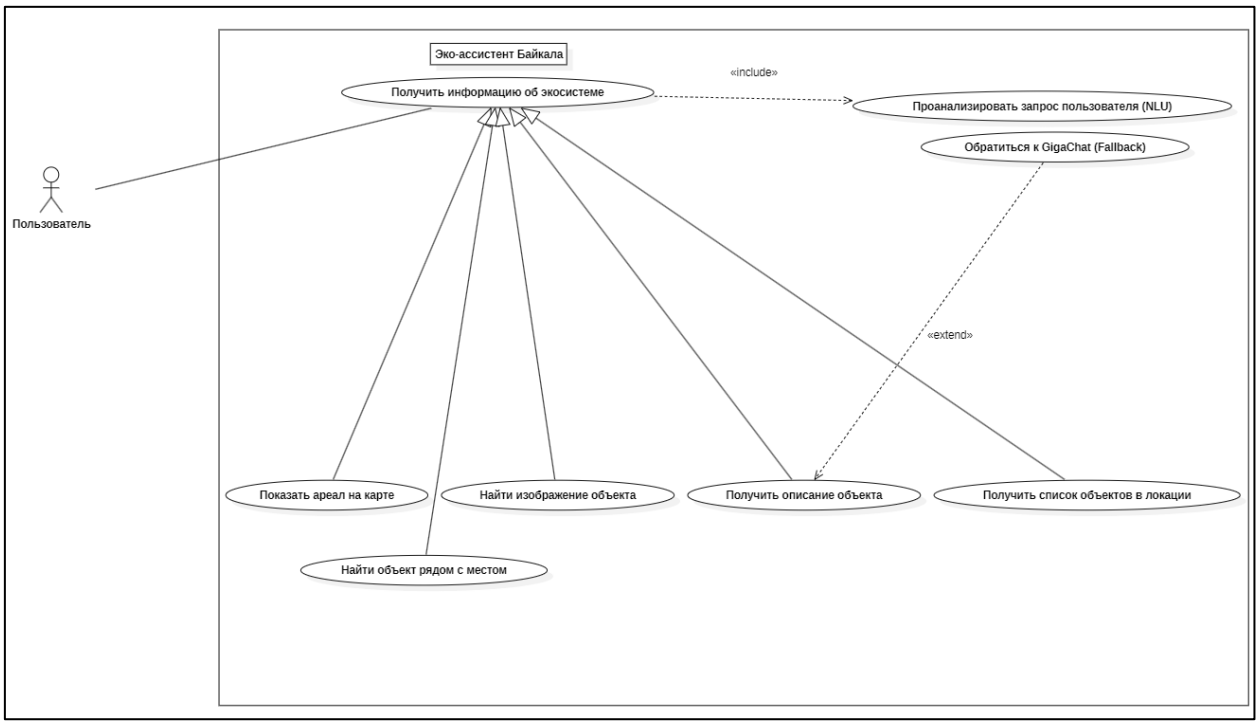

Рисунок 3 – Диаграмма вариантов использования (Use Case)

4.1 Описание компонентов диаграммы

Действующие лица (Actors):

«Пользователь»: Единственное действующее лицо, представляющее всех конечных пользователей системы (туристов, студентов, исследователей). Пользователь инициирует все взаимодействия с системой с целью получения информации.

Граница системы (System Boundary):

«Эко-ассистент Байкала»: Прямоугольник, обозначающий границы разрабатываемой системы. Все варианты использования (Use Cases), находящиеся внутри, являются функциями, реализуемыми непосредственно ассистентом.

Варианты использования (Use Cases) и их отношения:

Базовый вариант использования (Parent Use Case):

«Получить информацию об экосистеме»: является обобщенным (родительским) вариантом использования, который отражает основную цель Пользователя. Он напрямую не выполняется, а конкретизируется через дочерние Use Cases. Пользователь напрямую ассоциирован с этой целью.

Специализированные варианты использования (Child Use Cases):

Эти варианты наследуются от базового, представляя собой конкретные способы получения информации:

– «Получить описание объекта»: Цель пользователя – получить текстовую справку о конкретном виде флоры или фауны.

– «Найти изображение объекта»: Цель – получить визуальное представление объекта, возможно, с учетом определенных признаков (например, "зимой").

– «Показать ареал на карте»: Цель – увидеть на карте общую область распространения вида.

– «Найти объект рядом с местом»: более сложный гео-запрос, целью которого является поиск мест обитания вида вблизи указанной локации.

– «Получить список объектов в локации»: Цель – получить перечень всех известных видов, встречающихся в заданной географической области.

Обязательный служебный вариант использования (с отношением <<include>>):

– «Проанализировать запрос пользователя (NLU)»: Данный Use Case является обязательной частью основного сценария «Получить информацию об экосистеме». Отношение <<include>> показывает, что для выполнения любого информационного запроса система всегда и в обязательном порядке должна сначала выполнить анализ текста пользователя для распознавания его намерения и извлечения сущностей. Это ядро интеллектуальной составляющей системы.

Опциональный служебный вариант использования (с отношением <<extend>>):

– "Обратиться к GigaChat (Fallback)": Этот Use Case расширяет функционал варианта "Получить описание объекта". Отношение <<extend>> показывает, что данное действие является опциональным и выполняется только при определенных условиях: если в основной верифицированной базе знаний не нашлось данных, и, если сам Пользователь предварительно активировал данную функцию в настройках. Такое моделирование точно отражает архитектурное решение об ограниченном и контролируемом применении внешних LLM для сохранения достоверности данных.

4.2 Перспективы расширения функционала

Представленная диаграмма описывает функциональные возможности системы на текущем этапе реализации. Важно отметить, что выбранная архитектура и группировка функций с помощью наследования (Generalization) обеспечивают высокую расширяемость системы. В будущем набор вариантов использования может быть легко дополнен новыми функциями. Эти новые возможности логично впишутся в существующую структуру как новые дочерние элементы общего варианта использования "Получить информацию об экосистеме", не требуя кардинального пересмотра архитектуры.

5 Выработка требований и постановка задачи

Настоящий раздел формализует требования к программному продукту «Эко-ассистент Байкала» и определяет комплексную задачу для его разработки. Требования были выработаны на основе анализа предметной области, изучения существующих решений, а также моделирования бизнес-процессов (AS IS и TO BE) и вариантов использования (Use Case).

5.1 Требования к системе

Требования к системе декомпозируются на три ключевые категории: функциональные, нефункциональные и системные.

– ФТ-1: Анализ запроса пользователя: Система должна обеспечивать анализ входящих текстовых сообщений от пользователя на естественном языке для определения его намерения и извлечения ключевых сущностей (название объекта, географическая локация, признаки);

– ФТ-2: Предоставление текстового описания: Система должна предоставлять пользователю возможность получить верифицированную текстовую информацию (описание, факты) о запрашиваемом объекте флоры или фауны;

– ФТ-3: Предоставление изображений: Система должна предоставлять пользователю галерею изображений по запросу о биологическом объекте;

– ФТ-4: Отображение ареала на карте: Система должна быть способна сгенерировать и отобразить карту с общим ареалом обитания запрашиваемого вида;

– ФТ-5: Поиск объектов вблизи локации: Система должна предоставлять пользователю возможность найти места обитания указанного вида вблизи заданной географической точки;

– ФТ-6: Формирование списков объектов: Система должна уметь формировать и предоставлять списки видов флоры и фауны, встречающихся в указанной пользователем локации;

– ФТ-7: Управляемый фоллбэк-механизм: Система должна включать опциональный механизм фоллбэка к большой языковой модели (GigaChat) для получения текстового описания в случае отсутствия данных в основной базе. Данная функция должна быть активируема пользователем в настройках;

– ФТ-8: Переключение режимов обработки: Система должна предоставлять пользователю возможность вручную переключаться между двумя режимами NLU-обработки через меню настроек.

– НФТ-1: Надежность и достоверность информации: это ключевое требование. Система должна использовать в качестве основного источника только специализированную и верифицированную базу данных. Применение внешних LLM должно быть строго ограничено для предотвращения фактологических ошибок и «галлюцинаций»;

– НФТ-2: Производительность: Время ответа системы на простой запрос (например, получение описания) не должно превышать 5 секунд. Для комплексных запросов, требующих генерации карты, время ответа не должно превышать 15 секунд;

– НФТ-3: Удобство использования: Взаимодействие с системой должно происходить в интуитивно понятном диалоговом режиме через интерфейс мессенджера Telegram. Система должна корректно обрабатывать вариативные формулировки на естественном русском языке;

– НФТ-4: Расширяемость: Архитектура системы должна быть модульной и позволять в будущем добавлять новые функции (варианты использования) и источники данных без необходимости полной переработки существующих компонентов;

– НФТ-5: Доступность: Система должна быть доступна для пользователей в режиме 24/7.

– СТ-1: Система должна быть реализована в виде чат-бота для платформы Telegram и взаимодействовать с пользователями через Telegram Bot API.

– СТ-2: Система должна быть реализована на основе архитектуры с двумя независимыми, переключаемыми пользователем режимами (конвейерами) обработки запросов: один на базе NLU-фреймворка Rasa, второй – на базе большой языковой модели GigaChat.

– СТ-3: Система должна взаимодействовать с внешним бэкенд-сервисом (API) для доступа к верифицированной базе знаний и генерации мультимедийного контента.

5.2 Постановка задачи

На основании вышеизложенных требований, ставится следующая комплексная задача на разработку:

Разработать и реализовать программный комплекс «Эко-ассистент Байкала» в виде Telegram-бота, предназначенного для предоставления пользователям верифицированной мультимедийной информации о флоре и фауне Байкальского региона.

Для достижения поставленной цели необходимо выполнить следующие подзадачи:

– спроектировать микросервисную архитектуру программного комплекса, включающую модуль взаимодействия с Telegram, два независимых NLU-конвейера и модуль интеграции с внешним API;

– реализовать модуль взаимодействия с пользователем, обеспечивающий прием, отправку, корректное отображение сообщений и интерфейс для переключения режимов в Telegram;

– реализовать два независимых конвейера обработки NLU-запросов: конвейер на основе фреймворка Rasa для распознавания структурированных команд; конвейер, использующий большую языковую модель GigaChat для семантического анализа запросов на естественном языке;

– реализовать модуль генерации ответов, отвечающий за формирование запросов к внешнему API и преобразование полученных данных в удобный для пользователя формат (текст, галереи изображений, интерактивные карты).

– реализовать опциональный фоллбэк-механизм для текстовых запросов с возможностью его активации пользователем;

– провести комплексное тестирование системы, включая функциональное тестирование (проверка всех вариантов использования в обоих режимах), тестирование производительности и оценку качества пользовательского опыта (UX).

6 Выбор и обоснование средств проектирования и реализации

Выбор инструментария для разработки программного комплекса «Эко-ассистент Байкала» осуществлялся исходя из требований к надежности, масштабируемости и необходимости реализации гибридной архитектуры обработки естественного языка. Стек технологий был разделен на две категории: средства концептуального моделирования (для этапа анализа) и средства программной реализации (для этапа разработки).

6.1 Средства проектирования

Для формализации требований и моделирования бизнес-процессов были выбраны нотации, являющиеся отраслевыми стандартами, что обеспечивает однозначность интерпретации проектных решений.

1. Нотация BPMN 2.0 (Business Process Model and Notation): использовалась для моделирования процессов получения информации пользователем в состояниях «AS IS» (как есть) и «TO BE» (как будет). Выбор данной нотации обоснован её способностью наглядно отображать временную последовательность действий, потоки данных и зоны ответственности. Это позволило выявить «узкие места» текущего процесса (фрагментация источников, ручная фильтрация) и продемонстрировать эффективность автоматизации через внедрение бота.

2. Язык UML (Unified Modeling Language): применялся для построения диаграммы вариантов использования (Use Case Diagram). UML позволяет четко очертить границы системы (System Boundary) и определить функциональные требования с точки зрения конечного пользователя. Это критически важно для разделения базового функционала (получение справки) и расширенного (фоллбэк-сценарии, геопоиск).

6.2 Средства реализации

Технологический стек проекта сформирован на базе микросервисной архитектуры, что продиктовано необходимостью интеграции разнородных компонентов (классического NLU и LLM) и различиями в их программных окружениях.

6.2.1 Язык программирования и основной фреймворк

В качестве основного языка разработки выбран Python (версии 3.9 для Rasa и 3.10 для остальных сервисов). Выбор обоснован доминирующим положением Python в сфере искусственного интеллекта и наличием обширной экосистемы библиотек для обработки данных.

Для реализации логики Telegram-бота использован асинхронный фреймворк aiogram 2.x. В отличие от синхронных библиотек (например, python-telegram-bot), aiogram позволяет обрабатывать тысячи запросов конкурентно, не блокируя поток выполнения при ожидании ответа от внешних API (базы данных, картографического сервиса или GigaChat). Это критически важно для соблюдения нефункционального требования по времени отклика (НФТ-2).

6.2.2 Реализация двух независимых NLU-контуров

Вместо использования одной универсальной модели было принято решение реализовать два раздельных режима обработки – Rasa и GigaChat. Это решение обосновано необходимостью баланса между стоимостью эксплуатации, надежностью и интеллектуальными возможностями системы.

Контур на базе Rasa Open Source (Локальный NLU)

Используется как базовый, автономный режим работы. Rasa – это фреймворк с открытым исходным кодом для создания диалоговых систем, который разворачивается локально (On-Premise) внутри контура приложения.

Обоснование выбора:

1. Экономическая эффективность: Rasa является полностью бесплатным инструментом. Обработка запросов в этом режиме не расходует платные токены внешних моделей.

2. Автономность и надежность: Работа этого модуля не зависит от наличия интернет-соединения с внешними облачными провайдерами. Это гарантирует работоспособность базовых функций бота даже при сбоях в работе сторонних API.

3. Предсказуемость: Rasa работает на основе жестко заданных интентов (намерений). Это идеально подходит для навигации по меню, выполнения конкретных команд и ответов на часто задаваемые вопросы (FAQ), где недопустима вариативность или «творчество» нейросети

Контур на базе GigaChat API (Облачный LLM)

Используется как продвинутый режим для семантического анализа сложных запросов. Интеграция с большой языковой моделью от Сбера через REST API и библиотеку LangChain.

Обоснование выбора:

1. Глубина понимания языка: Классические NLU-системы (как Rasa) требуют огромных размеченных датасетов для понимания сложных, нешаблонных запросов. GigaChat, обладая знаниями «из коробки» (Zero-shot learning), способен анализировать произвольный текст пользователя и извлекать из него сложные сущности и контекст без предварительного обучения.

2. Работа с неструктурированными данными: Данный режим позволяет обрабатывать запросы, которые не были заранее предусмотрены разработчиком, обеспечивая гибкость диалога.

6.2.3 Инфраструктурные средства

1. Redis: Выбран в качестве in-memory хранилища данных. В условиях наличия двух независимых режимов Redis выступает единым хранилищем контекста пользователя. Это позволяет сохранять историю диалога и пользовательские настройки (например, выбранный режим) при перезапуске отдельных сервисов.

2. Docker & Docker Compose: используются для изоляции окружений. Поскольку Rasa и GigaChat-модуль требуют разных версий Python и зависимостей, контейнеризация является единственным способом обеспечить их бесконфликтную работу на одном сервере.

7 Проектирование архитектуры приложения

Разработанная система построена на принципах микросервисной архитектуры. Это решение позволяет изолировать разнородные технологии (разные версии Python для Rasa и основного бота), обеспечить независимую масштабируемость компонентов и повысить отказоустойчивость системы в целом.

Взаимодействие между компонентами осуществляется посредством REST API по протоколу HTTP, а управление состоянием диалогов вынесено в отдельное высокопроизводительное хранилище.

7.1 Схема архитектуры

На Рисунке 4 представлена компонентная диаграмма системы, демонстрирующая потоки данных между модулями.

Рисунок 4 – Архитектура программного комплекса «Эко-ассистент Байкала»

7.2 Описание компонентов системы

Архитектура состоит из следующих функциональных блоков:

1. Telegram Bot Service (Контроллер):

Центральный модуль системы, реализованный на aiogram. Он выполняет роль API-шлюза и маршрутизатора (Router).

– Функции: принимает сообщения от Telegram, проверяет настройки пользователя в Redis (выбранный режим работы) и направляет запрос в соответствующий NLU-конвейер (Rasa или GigaChat). После обработки запроса он формирует итоговый ответ пользователю.

– Особенность: Данный сервис не содержит бизнес-логики по флоре и фауне, он лишь управляет потоками данных.

2. NLU Конвейеры (Обработчики смысла):

– Rasa Server: Изолированный контейнер с Rasa Open Source. Принимает текст, возвращает классифицированный интент (намерение) и извлеченные сущности. Используется для детерминированных сценариев.

– GigaChat Integration: Модуль, отвечающий за взаимодействие с LLM. Он формирует промпты, отправляет их в API Сбера и парсит ответ, преобразуя неструктурированный текст в JSON-команды для бота.

3. Redis (Хранилище состояния):

In-memory база данных, используемая для реализации паттерна FSM (Finite State Machine).

– Назначение: хранит сессионные данные (история диалога для контекста, текущий режим обработки, настройки пользователя, временные кэши пагинации). Это позволяет сервису бота быть «stateless» (без состояния) и легко перезапускаться без потери данных диалога.

4. External Backend API (Источник данных):

Внешний веб-сервис (testecobot.ru), выступающий в роли «Единого источника истины» (Single Source of Truth).

– Роль: Бот обращается к этому API только когда намерения пользователя уже понятны. Бэкенд выполняет поиск по базе данных, генерирует карты ареалов и возвращает ссылки на медиа-контент.

– Архитектурное решение: Вынос базы данных и логики поиска во внешний сервис позволяет использовать эти же данные для других клиентов (например, веб-сайта или мобильного приложения), не дублируя логику внутри Telegram-бота.

7.3 Обоснование выбора архитектуры

Выбор микросервисной архитектуры с внешним источником данных обусловлен следующими факторами:

1. Разделение ответственности: логика общения с пользователем (Telegram Bot) полностью отделена от логики хранения биологических данных (Backend API). Это позволяет менять структуру базы данных или алгоритмы поиска на бэкенде, не останавливая работу бота.

2. Изоляция окружений: для работы Rasa требуется специфическая версия Python (3.9) и набор библиотек TensorFlow, которые конфликтуют с современными библиотеками для работы с LLM (langchain, Python 3.10+). Использование Docker-контейнеров позволяет запускать эти сервисы на одной машине без конфликтов зависимостей.

3. Гибкость маршрутизации: наличие центрального контроллера (Bot Service) позволяет динамически переключать потоки обработки (Rasa/GigaChat) «на лету» для каждого конкретного пользователя, основываясь на данных из Redis, что было бы невозможно при монолитной архитектуре.

4. Отказоустойчивость: при падении одного из NLU-сервисов (например, недоступности API Сбера), архитектура позволяет быстро переключить пользователей на резервный локальный контур (Rasa), сохраняя базовую работоспособность системы.

8 Проектирование хранилища данных

В качестве системы управления базами данных (СУБД) была выбрана PostgreSQL. Выбор обусловлен необходимостью работы с геопространственными данными (расширение PostGIS) и векторными представлениями текстов для семантического поиска (расширение pgvector).

8.1 Нормализация и структура данных

Проектирование базы данных выполнено с соблюдением требований третьей нормальной формы (3НФ) для обеспечения целостности данных и устранения избыточности:

– 1НФ: все атрибуты атомарны (за исключением специализированных полей feature_data типа JSONB, используемых для хранения слабоструктурированных метаданных, что является осознанным архитектурным решением для обеспечения гибкости).

– 2НФ: все неключевые атрибуты зависят от полного первичного ключа.

– 3НФ: отсутствуют транзитивные зависимости между неключевыми атрибутами. Например, информация об авторах вынесена в отдельную таблицу author и связывается с контентом через таблицу развязки entity_author, а не хранится внутри таблиц с текстами или изображениями.

8.2 Основные сущности

В основе хранилища лежит разделение на сущности предметной области (биология, географии) и типы контента (текст, медиа, карты). Описание основных сущностей представлено в таблице 1.

Таблица 1 – Описание основных сущностей БД

Название таблицы	Назначение	Основные атрибуты
biological_entity	Хранение информации о видах флоры и фауны.	id, common_name_ru (общее название), scientific_name (латынь), type (флора/фауна), feature_data (метаданные).
geographical_entity	Хранение географических объектов (топонимов).	id, name_ru, type (тип объекта), description.
text_content	Текстовые описания, статьи и факты. Включает векторные эмбеддинги.	id, title, content, embedding (вектор для поиска), structured_data (JSON).
image_content	Метаданные изображений.	id, title, description, feature_data (EXIF, классификация).
map_content	Картографические данные и геометрия.	id, title, geometry (PostGIS объект: точки, полигоны).
entity_identifier	Реестр универсальных идентификаторов для поиска.	id, name_ru, name_en, name_latin.
reliability	Оценка достоверности информации.	entity_table, entity_id, reliability_value (уровень доверия к источнику).

8.3 Логическая модель данных

Архитектура базы данных использует паттерн «Полиморфные связи» и «Звезда».

1. Централизация связей: вместо создания жестких связей между всеми таблицами, используется универсальная таблица связей entity_relation.

Она связывает любую сущность (Source) с любой другой (Target) через поля entity_type и entity_id.

Это позволяет гибко строить граф знаний (например, связать «Текст» с «Видом животного», а «Вид животного» с «Географическим объектом»).

2. Специализированные таблицы развязки: для часто используемых связей созданы отдельные таблицы:

– entity_geo – связь контента с географией.

– entity_author – связь контента с авторами.

– entity_temporal – привязка к временным меткам (сезоны, даты).

3. Векторное пространство: Таблица text_content содержит поле embedding (вектор размерности 768 или 1024, в зависимости от модели). Для ускорения поиска ближайших соседей (Cosine Similarity) используется индекс IVFFlat.

8.4 Ограничения целостности

Для поддержания согласованности данных реализованы следующие ограничения на уровне СУБД:

1. Первичные ключи: Каждая таблица имеет суррогатный ключ id SERIAL PRIMARY KEY для однозначной идентификации записей.

2. Ссылочная целостность. Таблицы связей (например, entity_identifier_link) используют внешние ключи с опцией ON DELETE CASCADE. Это гарантирует, что при удалении основной сущности (например, статьи) автоматически удалятся все её связи, предотвращая появление «битых» ссылок.

3. Ограничения доменов:

– В таблице reliability: проверка, что имя таблицы не пустое (CHECK (entity_table <> '')).

– В таблице weather_reference: влажность ограничена диапазоном 0–100% (CHECK (humidity BETWEEN 0 AND 100)).

– В таблице temporal_reference: месяц ограничен диапазоном 1–12.

4. Уникальность. В таблице reliability запрещено дублирование оценки достоверности для одной и той же сущности (UNIQUE (entity_table, entity_id, column_name)).

5. Геопространственная целостность. Поле geometry в map_content типизировано (используется SRID 4326 – WGS 84), что предотвращает запись некорректных геоданных.

8.5 Схема данных

На рисунке 5 представлен фрагмент логической схемы базы данных.

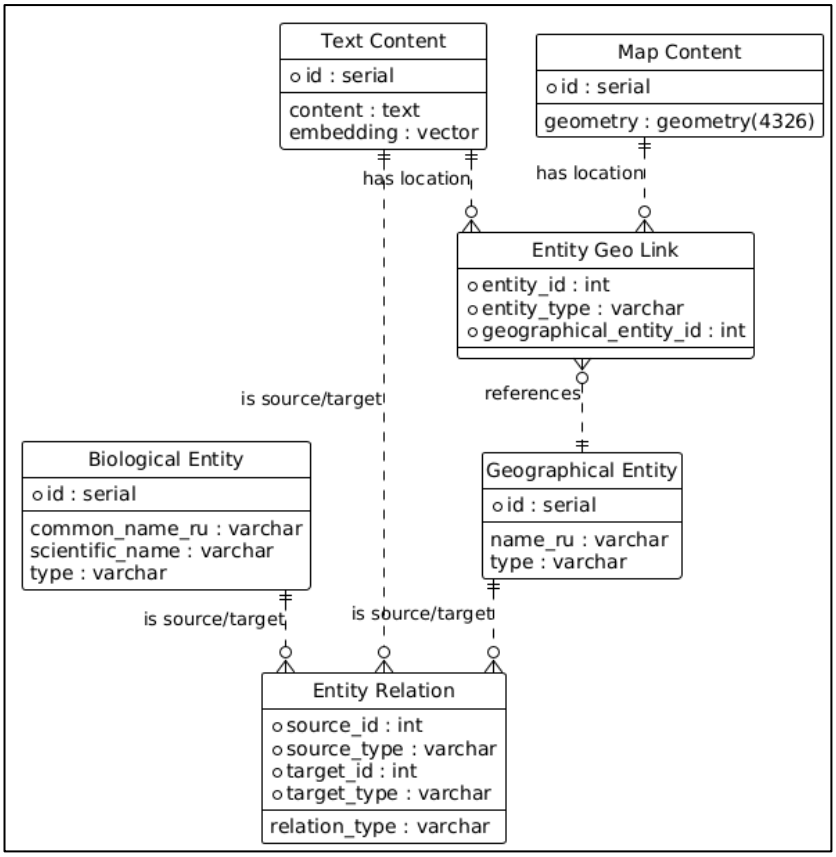

Рисунок 5 – Фрагмент логической схемы базы данных (основные сущности и полиморфные связи)

9 Проектирование пользовательского интерфейса

В связи с реализацией системы на платформе Telegram, пользовательский интерфейс относится к классу Conversational UI (Разговорный интерфейс). В отличие от классических графических интерфейсов (GUI), навигация здесь осуществляется не через переходы по страницам, а через обмен сообщениями и изменение состояний интерактивных элементов (клавиатур).

Проектирование интерфейса выполнялось с учетом ограничений и гайдлайнов Telegram Bot API. Основной упор сделан на минимизацию текстового ввода там, где это возможно, и предоставление наглядного мультимедийного контента.

9.1 Верхнеуровневое определение экранов (Состояний диалога)

«Экраны» сгруппированы по трем функциональным разделам: Навигация и настройки, Информационный поиск и Обработка неопределенностей. В таблице 2 представлено описание экранов (состояний) пользовательского интерфейса.

Таблица 2 – Описание экранов (состояний) пользовательского интерфейса

№	Краткое название (Name)	Поля ввода / Валидация (Input/Validation)	Описание и поведение (Behavior)	Состояния (States)
1.0	Экран приветствия (Start)	Команда /start	Точка входа. Бот приветствует пользователя, объясняет свои возможности и предлагает открыть меню настроек.	Default
1.1	Настройки и режимы	Ввод: Нажатие Inline-кнопок. Валидация: Обработка callback_data.	Меню, прикрепленное к сообщению. Позволяет переключать режимы (Rasa / GigaChat), включать/выключать фоллбэк и стоп-лист. Кнопки меняют вид (галочки) при нажатии.	Settings_Active
1.2	Инлайн-поиск (Search)	Ввод: Символ @ + текст. Валидация: Поиск совпадений в локальном списке names.txt.	Всплывающее меню Telegram (native). Предлагает варианты автодополнения названий видов. При выборе отправляет	Inline_Query

			готовую команду боту.	
2.0	Основной ввод запроса	Ввод: Текстовое сообщение. Валидация: 1. Проверка на длину. 2. NLU-анализ (определение интента и сущностей).	Пользователь пишет запрос (например, «Покажи нерпу»). Бот отображает статус «печатает...» (ChatAction), пока идет запрос к бэкенду или LLM.	Processing
2.1	Карточка объекта (Текст/Фото)	Ввод: нет (Read-only) или кнопки пагинации. Валидация: Нет.	Ответ системы. Содержит фото (или галерею), отформатированный текст (Markdown) и, опционально, аудио/видео файл.	Response_Success
2.2	Интерактивная карта	Ввод: Кнопка-ссылка (URL). Валидация: Нет.	Бот присылает статичное превью карты (изображение) и Inline-кнопку «🌍 перейти на карту», ведущую на веб-версию геоинформационной системы.	Map_View
3.0	Уточнение (Disambiguation)	Ввод: Выбор из списка кнопок. Валидация: Callback-data с индексом варианта.	Возникает, если запрос неоднозначен (например, «Ива» — какая именно?). Бот предлагает список кнопок с конкретными видами.	Clarification_Wait
3.1	Фоллбэк (GigaChat)	Ввод: Нет. Валидация: Проверка флага настроек.	Если в БД нет ответа, но включен режим фоллбэка, выводится сгенерированный текст с пометкой «Ответ от GigaChat».	Fallback_Active

9.2 Карта экранов и состояний

Логика переходов между состояниями интерфейса представлена на Рисунке 6 в виде диаграммы состояний. Она демонстрирует циклический характер взаимодействия: после получения результата система возвращается в состояние ожидания нового запроса, сохраняя при этом контекст диалога.

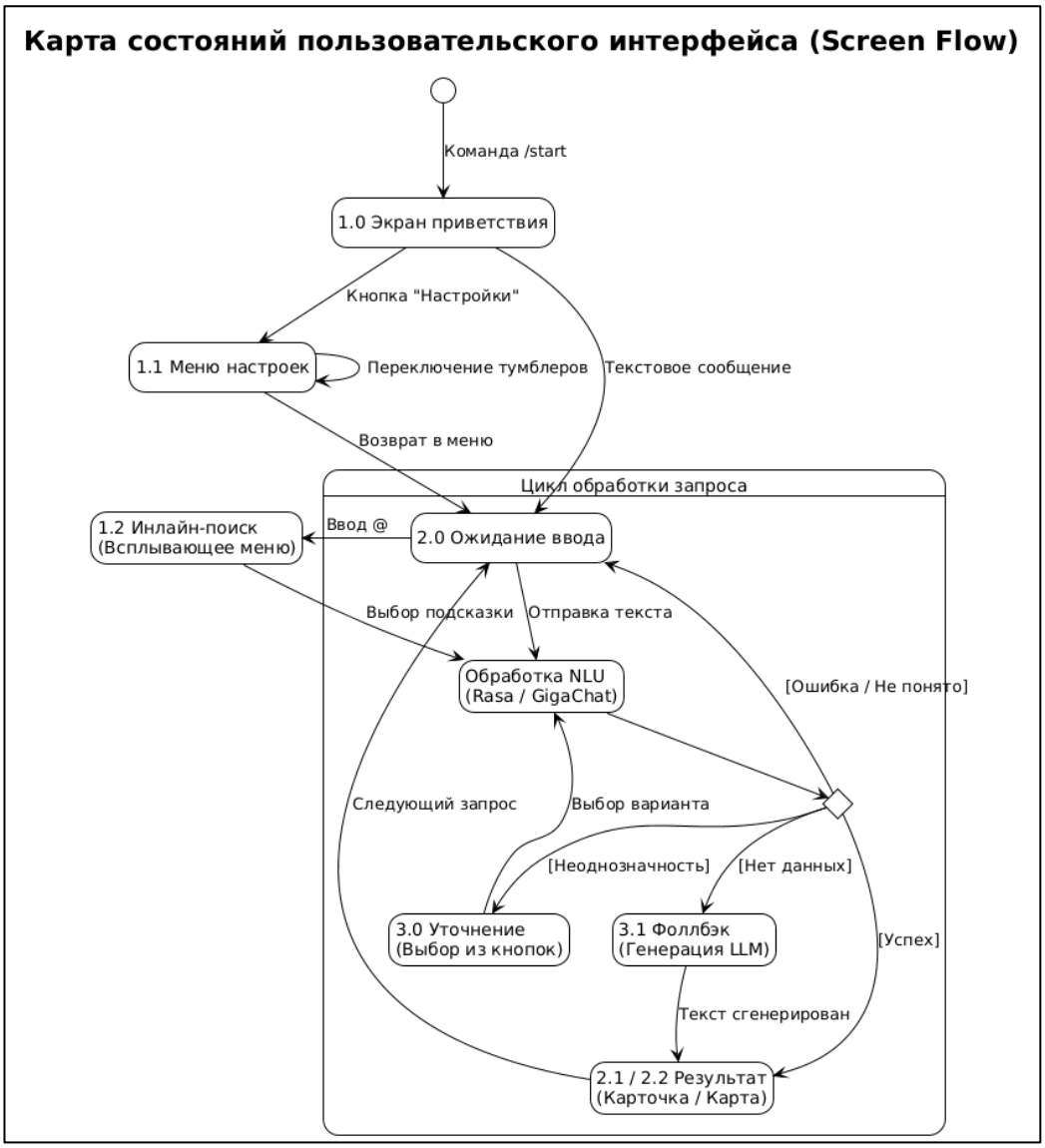

Рисунок 6 – Карта навигации пользовательского интерфейса

9.3 Особенности реализации UI в Telegram

Поскольку Telegram предоставляет жестко заданный фреймворк для отображения информации, проектирование интерфейса сводилось к эффективному использованию доступных нативных элементов.

9.3.1 Использование клавиатур

В проекте используются два типа клавиатур для разных сценариев:

1. Inline Keyboards (Кнопки под сообщением): Основной инструмент управления. Используются для настроек (тумблеры вкл/выкл), ссылок на внешние карты и выбора вариантов при уточнении. Они не засоряют историю чата, так как привязаны к конкретному контексту (сообщению) и могут динамически обновляться.

2. Reply Keyboards (Кнопки под полем ввода): используются только для глобальных действий, таких как «⚙ Настройки» или «Помощь». Эти кнопки должны быть доступны всегда, независимо от контекста диалога.

9.3.2 Форматирование и медиа

Для улучшения читаемости (UX) используются:

1. MarkdownV2: для выделения ключевых сущностей жирным шрифтом (например, названия видов), оформления списков и цитат.
2. Вложения: Фотографии и карты отправляются не ссылками, а как нативные объекты PhotoSize, что позволяет просматривать их во встроенной галерее Telegram без перехода в браузер.
3. Chat Actions: Бот отправляет статус «typing» (печатает) или «upload_photo» (загружает фото) во время длительной обработки запроса (НФТ-2), чтобы пользователь понимал, что система работает, а не зависла.

9.3.3 Цветовая палитра и стили

Цветовое решение интерфейса диктуется темой, установленной в клиенте пользователя (Светлая/Темная/Системная). Разработка велась с учетом адаптивности:

1. Используются стандартные цвета кнопок Telegram (серый прозрачный).
2. Изображения карт генерируются бэкендом в нейтральной цветовой гамме, обеспечивающей контрастность как на светлом, так и на темном фоне мессенджера.
3. В текстовых ответах используются эмодзи (🌿, 🗺️, 📖) в качестве визуальных якорей для быстрой идентификации типа контента (например, 🌿 — для флоры, 🗺️ — для географии).

10 Реализация и итоги работы

В результате выполнения курсовой работы был спроектирован и программно реализован программный комплекс «Эко-ассистент Байкала». Система успешно развернута в контейнеризированной среде и прошла функциональное тестирование.

Ниже представлены ключевые результаты реализации, демонстрирующие работоспособность архитектуры и пользовательского интерфейса.

10.1 Реализация пользовательского интерфейса

Взаимодействие с пользователем осуществляется через клиент Telegram. Реализован адаптивный интерфейс, поддерживающий работу с медиа-контентом и интерактивными элементами.

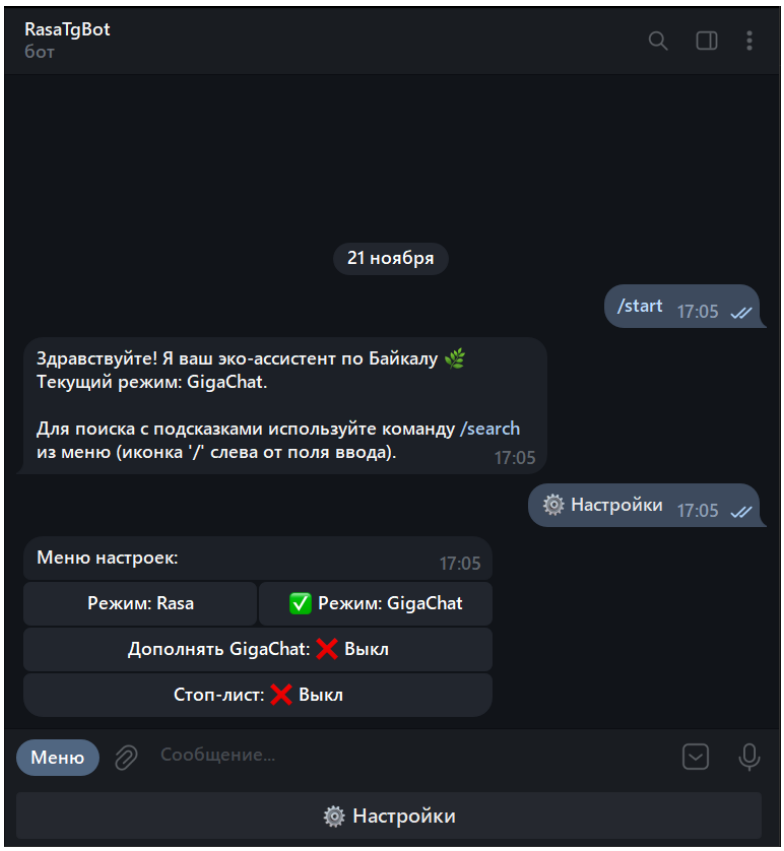

Рисунок 7 – Главное меню и настройки

На Рисунке 7 представлен экран настроек, позволяющий пользователю переключать NLU-конвейеры «на лету».

На Рисунке 8 продемонстрирована работа геоинформационного модуля: система распознала вид, нашла его координаты в PostGIS и сгенерировала карту ареала.

Рисунок 8 - Результат запроса с генерацией карты

10.2 Программная реализация логики маршрутизации

Одним из наиболее сложных архитектурных элементов стала реализация переключения режимов обработки (ФТ-8). В Листинге 1 приведен фрагмент кода главного роутера (bot.py), который определяет, в какой микросервис направить запрос, основываясь на профиле пользователя в Redis.

Листинг 1 – Фрагмент маршрутизации сообщений (Python, aiogram)

```
@dispatcher.message_handler(content_types=types.ContentTypes.TEX
T)
async def handle_message_by_mode(message: types.Message):
    # Игнорируем команды, начинающиеся со слэша
    if message.text.startswith('/'):
        return

    # Получаем настройки пользователя из Redis/FileLock
    user_id = str(message.from_user.id)
    user_settings = get_user_settings(user_id)
    mode = user_settings.get("mode", "gigachat") # Режим по
умолчанию

    # Маршрутизация потока
    if mode == "rasa":
        # Отправка в контейнер Rasa (Локальный NLU)
        await rasa_h.process_message(message)
    else:
        # Отправка в модуль LLM (GigaChat)
        await gigachat_h.process_message(message)
```

10.3 Реализация семантического анализа (LLM)

Для обработки неструктурированных запросов (режим GigaChat) был разработан модуль QueryAnalyzer. Он не просто передает текст нейросети, а использует технику Prompt Engineering для извлечения структурированных данных (JSON).

Листинг 2 – Структура системного промпта для анализа (фрагмент)

```
# Из файла logic/prompts_structure/prompts.py
("system", """
## РОЛЬ
Ты — высокоточный NLU-аналитик. Твоя задача — разобрать текущий
запрос пользователя...
и вернуть СТРОГО JSON с его полной структурой.

## ФОРМАТ ВЫВОДА (JSON)
{{
  "action": "...",  // describe, show_map, find_nearby
  "primary_entity": {{
    "name": "...",
    "type": "...", // Biological, GeoPlace, Infrastructure
    "category": "..."
```

```
  }},
  "attributes": {{ "season": "...", "habitat": "..." }}
}}
""")
```

Такой подход позволил превратить генеративную модель в детерминированный инструмент управления бизнес-логикой.

10.4 Реализация базы данных и геопространственных запросов

Хранилище данных спроектировано для работы с полиморфными связями и геометрий. В Листинге 3 представлен SQL-скрипт создания таблицы для хранения картографических данных с использованием расширения PostGIS.

Листинг 3 — DDL таблицы геоданных (SQL)

```
-- Создание таблицы для хранения геометрии
CREATE TABLE map_content (
    id SERIAL PRIMARY KEY,
    title VARCHAR(500) NOT NULL,
    description TEXT,
    -- Использование типа GEOMETRY из PostGIS с проекцией WGS 84
(SRID 4326)
    geometry GEOMETRY(Geometry, 4326) NOT NULL,
    feature_data JSONB
);

-- Создание пространственного индекса GiST для ускорения поиска
CREATE INDEX idx_map_geometry ON map_content USING GIST(geometry);
```

Использование индекса GIST обеспечивает выполнение пространственных запросов (например, «найти все объекты в радиусе 5 км») за миллисекунды, что удовлетворяет нефункциональному требованию НФТ-2 (производительность).

10.5 Аналитика обработки данных

В ходе импорта данных из внешних источников (датасет resources_dist.json) была реализована автоматическая генерация векторных эмбеддингов для текстового контента. Это позволяет в будущем реализовать семантический поиск (RAG).

Листинг 4 — Генерация векторных представлений (Python)

```
# Из файла postgres_adapter.py
def generate_embedding(self, text):
    """Генерация эмбеддинга для текста"""
    try:
        # Использование модели HuggingFace (Rubert-tiny2 или
аналог)
        embedding = self.embedding_model.embed_query(text)
```

```
        # Проверка размерности вектора (например, 312 или 768)
        if len(embedding) != self.embedding_dimension:
            print(f"Предупреждение: Неверная размерность
вектора")

        return embedding
    except Exception as e:
        print(f"Ошибка генерации эмбеддинга: {e}")
        return None
```

Заключение

В ходе выполнения курсовой работы был спроектирован и программно реализован программный комплекс «Эко-ассистент Байкала». Основная цель работы — создание доступного инструмента для популяризации научных знаний о природе региона среди широкой аудитории — была достигнута посредством разработки интеллектуального диалоговой системы.

В процессе реализации была построена отказоустойчивая система на основе микросервисной архитектуры. Вместо использования монолитного решения был применен подход с разделением логики обработки естественного языка на два независимых режима. Для выполнения стандартных команд и навигации задействован локальный модуль Rasa, обеспечивающий высокую скорость и предсказуемость работы. Для анализа сложных неструктурированных запросов интегрирована большая языковая модель GigaChat. Такое архитектурное решение позволило объединить экономичность классических алгоритмов с гибкостью современных генеративных моделей.

Особое внимание было уделено проектированию хранилища данных. Реализованная база данных на платформе PostgreSQL с расширением PostGIS позволяет системе оперировать не только текстовой информацией, но и геопространственными объектами, обеспечивая возможность динамической генерации карт ареалов обитания. Пользовательский интерфейс, развернутый на платформе Telegram, поддерживает работу с мультимедийными галереями, инлайн-поиском и интерактивными элементами управления, а использование асинхронных библиотек гарантирует высокую скорость отклика системы.

Разработанная архитектура обладает значительным потенциалом для дальнейшего масштабирования. В качестве приоритетного направления развития рассматривается полноценное внедрение технологии семантического поиска (RAG). На текущем этапе реализована векторизация текстового контента, что создает фундамент для поиска ответов на основе смыслового сходства, а не только по ключевым словам. Кроме того, перспективным представляется расширение картографического модуля функционалом построения туристических маршрутов и внедрение элементов геймификации, например, образовательных викторин.

Таким образом, созданный программный комплекс представляет собой функционально законченный продукт, решающий прикладную задачу доступа к верифицированным экологическим данным и обладающий необходимой гибкостью для дальнейшей модернизации.