

АННОТАЦИЯ

Выпускная квалификационная работа изложена на 104 страницах, содержит 34 рисунка, 6 таблиц, 26 источников, 2 приложения.

ЦИФРОВОЙ АВАТАР, ГЕНЕРАЦИЯ ТЕКСТА, СИНТЕЗ РЕЧИ, ДИАЛОГОВАЯ СИСТЕМА

Данная выпускная квалификационная работа посвящена разработке и экспериментальной апробации веб-сервиса, ориентированного на создание и персонализацию виртуальных собеседников — цифровых аватаров, способных поддерживать диалог в текстовой форме с последующим синтезом речи. Целью исследования являлось проектирование архитектурного и программного решения, обеспечивающего возможность загрузки пользовательских материалов для модификации поведения аватара и адаптации языковой модели под индивидуальные особенности коммуникации.

В качестве метода построения системы использован модульный подход: её компоненты разделены на отдельные подсистемы, отвечающие за обработку пользовательских данных, хранение структурированной и мультимедийной информации, а также организацию обмена сообщениями между элементами системы. Пользовательский интерфейс спроектирован с акцентом на интерактивность, что обеспечивает низкое время отклика и удобство при взаимодействии с цифровым собеседником.

Генерация текстовых реплик осуществляется с применением русскоязычной языковой модели, подстраиваемой под стилистические особенности конкретного пользователя. Синтез речи реализован с использованием заранее обученной голосовой модели, в которую на этапе генерации передаются пользовательские аудиофрагменты для имитации индивидуальных голосовых характеристик. Таким образом, система позволяет воспроизводить реплики, близкие по тембру и интонации к оригинальному голосу пользователя.

Практическая значимость разработки заключается в построении минимально жизнеспособного прототипа, демонстрирующего возможность создания персонализированных цифровых аватаров с текстовым и голосовым взаимодействием. Основной сервис был развёрнут в локальной среде и включает

расширенный функционал для пользовательской настройки поведения аватара. Отдельные компоненты, такие как прототип генерации речи, разрабатывались с использованием облачных вычислительных платформ. Полученные результаты могут быть применимы в сферах образования, цифровой гуманистики и интерактивных мультимедийных приложений. Новизна работы состоит в объединении средств кастомизации как речевого, так и языкового поведения в рамках единого пользовательского интерфейса.

ABSTRACT

The Bachelor's thesis has 104 pages, 34 figures, 6 tables, 26 references, 2 appendices.

DIGITAL AVATAR, TEXT GENERATION, SPEECH SYNTHESIS, CHAT SYSTEM

This bachelor's thesis is devoted to the development and experimental evaluation of a web-based service aimed at the creation and personalization of virtual interlocutors—digital avatars capable of conducting text-based dialogues with subsequent speech synthesis. The goal of the research was to design an architectural and software solution that enables users to upload their own materials in order to modify avatar behavior and adapt a language model to individual communication styles.

A modular approach was used in building the system: its components are organized into distinct subsystems responsible for processing user data, storing structured and multimedia content, and coordinating message exchange between elements. The user interface is designed with a focus on interactivity, ensuring low response time and convenient engagement with the digital interlocutor.

Text generation is carried out using a Russian-language language model that can be adapted to the stylistic features of a specific user. Speech synthesis is implemented using a pre-trained voice model, which, during the generation stage, receives user audio samples to imitate individual vocal characteristics. As a result, the system can produce speech that closely resembles the user's original tone and intonation.

The practical significance of the project lies in the development of a minimum viable prototype that demonstrates the feasibility of creating personalized digital avatars capable of both textual and vocal interaction. The main service was deployed in a local environment and includes extended functionality for customizing avatar behavior. Certain components, such as the speech generation prototype, were developed using cloud-based computing platforms. The results obtained may find application in the fields of education, digital humanities, and interactive multimedia systems. The novelty of this work lies in the integration of both speech and language personalization tools within a unified user interface.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	10
ВВЕДЕНИЕ	11
1 Структурный и системный анализ исследуемого объекта	13
2 Обзор научно-технических источников информации	16
2.1 Концептуальные и архитектурные решения	16
2.1.1 Общий архитектурный паттерн. Микросервисный и монолитный подходы	16
2.1.2 Паттерн взаимодействия микросервисов. Хореография и оркестрация	17
2.1.3 Модель взаимодействия компонентов. Асинхронность, параллельность и конкурентность	18
2.1.4 Модель хранения данных	19
2.1.5 Паттерн клиентской архитектуры. Модели МРА и SPA . .	20
2.2 Программные средства и технологии реализации	21
2.2.1 Выбор языка программирования	21
2.2.2 Веб интерфейс	22
2.2.3 Серверная часть. Оркестратор	23
2.2.4 Серверная часть. Модуль работы с языковой и звуковой моделью	24
2.2.5 Серверная часть. Технология распознавания речи	27
2.2.6 Серверная часть. Брокер сообщений	28
2.2.7 Инфраструктура и развертывание	29
2.2.8 Итог по архитектуре	29
3 Постановка задачи	32
4 Сущность решения задачи	33
4.1 Компоненты решения	33
4.1.1 Веб интерфейс	33
4.1.2 Бекэнд	33
4.1.3 Реляционная база данных	33

4.1.4	ML сервис	34
4.1.5	Брокер сообщений	34
4.1.6	S3 хранилище	34
4.2	Алгоритм взаимодействия	34
4.2.1	Процедуры регистрации и аутентификации пользователя .	34
4.2.2	Процедура обучения	35
4.2.3	Процедура взаимодействия с аватаром	37
5	Построение модели решения задачи	41
5.1	Цели и ограничения	41
5.2	Модуль генерации текста	42
5.2.1	Данные	42
5.2.2	Выбор базовой модели	43
5.2.3	Метрики	44
5.2.4	Токенизация	44
5.2.5	Обучение	44
5.2.6	Генерация текста	45
5.3	Модуль генерации звука	45
5.3.1	Выбор базовой модели	46
5.3.2	Подготовка данных	47
5.3.3	Гиперпараметры дообучения модели	47
5.3.4	Процесс дообучения и отслеживание параметров	48
5.3.5	Оценка качества генерации и используемые метрики . . .	50
6	Программная реализация	51
6.1	Веб-интерфейс	51
6.1.1	Гостевая страница	51
6.1.2	Идентификация, аутентификация и авторизация	51
6.1.3	Основная страница приложения	53
6.1.4	Страница работы с аватаром	55
6.1.5	Интерфейс общения с аватаром	56
6.1.6	Интерфейс обучения аватара	58
6.2	Backend-сервис	60
6.2.1	Взаимодействие с веб-интерфейсом	61
6.2.2	Взаимодействие с базой данных	64
6.2.3	Взаимодействие с ML-сервисом	67

6.2.4	Взаимодействие с S3-хранилищем	72
6.3	Сервис генерации звука	74
6.3.1	Обработка событий брокера сообщений и инициализация очереди генерации	74
6.3.2	Этапы синтеза речи	75
6.3.3	Формат обратной связи через брокер сообщений	77
6.4	Сервис генерации текста	78
6.4.1	Пайплайн обучения модели	78
7	Результаты	89
7.1	Инициализация системы	89
7.2	Регистрация и создание аватара	89
7.3	Загрузка обучающих материалов	89
7.4	Обучение аватара	89
7.5	Диалог и синтез речи	90
ЗАКЛЮЧЕНИЕ		91
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		92
Приложение А Интерфейс		95
Приложение Б Отчёты о работе сервиса		99

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе применяются следующие сокращения и обозначения:

- ВКР — выпускная квалификационная работа
- ИИ — искусственный интеллект
- ОС — операционная система
- ПО — программное обеспечение
- СУБД — система управления базами данных
- API — application programming interface
- JWT — json web token
- SPA — single page application
- DOM — document object model
- CRUD — create, read, update, delete

ВВЕДЕНИЕ

С ростом интереса к персонализированным цифровым технологиям моделирование речи, поведения и стиля мышления конкретных людей приобретает всё большую значимость. Цифровые аватары, способные воспроизводить облик, голос и характерную манеру общения конкретной личности, становятся частью инфраструктуры новых медиаформатов и пользовательского взаимодействия. В условиях стремительного развития генеративных моделей и широкого внедрения искусственного интеллекта в повседневную коммуникацию технологии создания интеллектуальных двойников находят применение в самых разных сферах — от сохранения цифровой памяти до коммерческих виртуальных ассистентов и инфлюенсеров, не привязанных к физическому носителю.

Особую актуальность такие решения приобретают в контексте работы с публичными, культурными или научными фигурами. Возможность сохранить их речевую манеру и поведенческую логику в виде доступного цифрового интерфейса позволяет не только продлить культурное присутствие личности, но и сделать его интерактивным и образовательным. Кроме того, в условиях распространенного запроса на персонализированные сервисы и форматы «виртуального присутствия», технология создания интеллектуальных аватаров может найти применение в разработке цифровых консультантов, обучающих систем, а также в области цифрового наследия — как инструмент архивирования опыта, взглядов и коммуникативной стилистики конкретного человека.

Цель данной выпускной квалификационной работы — разработка сервиса под названием «Bishop», предназначенного для общения пользователей с цифровыми двойниками или «аватарами» реальных личностей. Под «аватаром» в данном контексте понимается интеллектуальная компьютерная модель, способная воспроизводить стиль речи, манеру ведения беседы и эмоциональные реакции конкретного человека.

Идея работы заключается в том, чтобы предоставить пользователю не просто «чат-бота», а максимально близкого к реальному собеседнику. При этом формат взаимодействия выходит за рамки текстовых сообщений. Сервис, помимо текстового чата, обеспечивает генерацию аудиоответов, что создаёт иллюзию живой речи. Такая функциональность востребована в самых разных сценариях: от сохранения культурного и научного наследия выда-

ющихся лекторов и артистов до интерактивных образовательных платформ, где диалог с «виртуальным преподавателем» может способствовать более глубокому усвоению материала.

Важной отличительной чертой сервиса является возможность обучения аватара на широком спектре материалов. Речь идёт не только об изначальном корпусе данных (тексты, аудио и видео с участием реального прототипа), но и о регулярном пополнении этих данных через пользовательский интерфейс. Например, в учебных целях можно загрузить дополнительную лекцию или интервью, что расширит «знания» цифрового двойника и углубит способность имитировать манеру общения конкретной личности. В результате пользователь получает впечатление общения с «живым» человеком, обладающим определённой индивидуальностью и способным рассуждать на разные темы в присущей ему манере.

1 Структурный и системный анализ исследуемого объекта

Рассмотрим популярные платформы, которые предоставляют функционал по созданию информационных аватаров для общения. В последние годы интерес к цифровым двойникам и аватарам значительно вырос, что привело к появлению множества сервисов, предлагающих пользователям возможность создавать виртуальные образы реальных или вымышленных личностей. Несмотря на разнообразие решений, концептуально платформы во многом близки по духу, предоставляя пользователям функционал, основанный на конфигурационных промптах, которые описывают общее поведение и возможные сценарии общения аватара с человеком. Сравнительный анализ основных характеристик и возможностей, которые предоставляются пользователям бесплатно на самых распространённых сервисах, представлен в таблице 1.

Таблица 1 — Сравнение сервисов для работы с аватарами

Параметры \ Сервис	character.ai	sakura.fm	talkie-ai.com
Общее описание аватара	+	+	+
Описание личности аватара	-	+	+
Описание сценария общения	-	+	+
Дополнительные инструкции для модели	-	+	+
Выбор пола для аватара	-	+	+
Выбор озвучки аватара	+	+	-
Создание озвучки из данных	+	-	-

Общая проблема представленных выше сервисов заключается в том, что настройка поведения аватара ограничивается конфигурационным промптом, суммарный размер которого не превышает 10000 знаков. Подобный подход обеспечивает возможность быстро задать общий паттерн поведения и получить качественную имитацию общения. Тем не менее, он недостаточно гибок,

чтобы полностью раскрыть индивидуальность конкретного человека, передать тонкие стилистические особенности его речи, его эмоциональный отклик, нюансы восприятия и конкретные взгляды на различные темы. В результате данные сервисы демонстрируют высокую эффективность прежде всего в бизнес-контексте, где важно быстро и эффективно автоматизировать

тические задачи коммуникации, но совершенно недостаточны для задач, где необходима глубокая и реалистичная передача личности собеседника.

Однако на рынке существуют также сервисы с альтернативным подходом к созданию виртуальных аватаров. Такие платформы, как правило, предлагают крайне минималистичный интерфейс, практически не предоставляющий пользователю возможностей тонкой настройки, и весь процесс персонализации аватара полностью переносится на сторону разработчика модели. Это приводит к тому, что пользователь либо просто наблюдает за генерируемой речью аватара, либо взаимодействует с уже обученной моделью, не имея возможности повлиять на её развитие и поведение.

Примером такого подхода может служить проект infiniteconversation.com, создатель которого сумел имитировать диалог между философом Славоем Жижеком и режиссёром Вернером Херцогом, воспроизведя их узнаваемый стиль публичных выступлений и ведения дискуссий. По словам единственного разработчика, Джакомо Мичели, в основу данного проекта, который был создан ещё в доисторические времена по меркам развития индустрии, легла GPT-2, а массив данных для обучения состоял из примерно 600 записей с выступлений для каждого из говорящих, размерами по 250 слов каждая.

Другим примером служит модель «Жириновский», представляющая собой виртуальный аватар известного российского политика, который способен вести диалог с пользователями посредством стороннего чат-бота. Из открытых источников [1] известно, что размер модели для генерации составил около 48 миллионов параметров, а корпус для обучения, после предварительной обработки, составил 50 тысяч записей в форме вопрос-ответ.

Ключевая задача, решаемая в рамках данной выпускной квалификационной работы, состоит в разработке сервиса, который предоставит пользователям возможность не только коммуницировать, но и полноценно обучать интеллектуального аватара. Это позволит существенно улучшить глубину и точность воспроизведения манеры общения реальной личности, включая её эмоциональные реакции и специфику речи.

Дополнительным подтверждением высокой актуальности выбранной темы является патентный анализ, представленный на рисунке 1. Как следует из анализа, количество патентов, связанных с созданием и использованием интеллектуальных аватаров, стабильно возрастает год от года. Это свидетельствует о постоянном росте интереса к данной сфере и подтверждает

достаточную изученность темы для того, чтобы приступить к реализации практического решения.

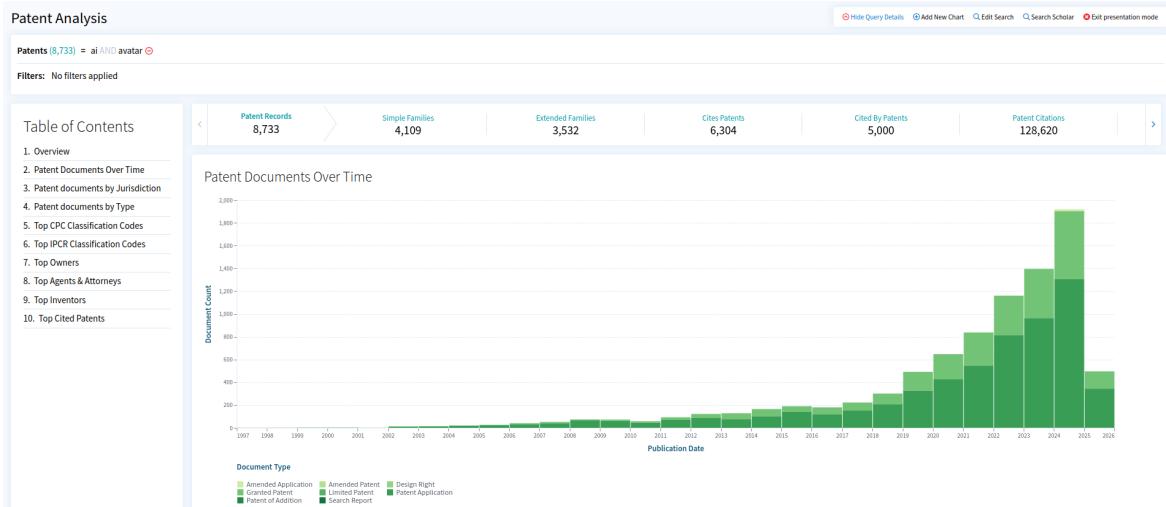


Рисунок 1 — Патентный анализ

2 Обзор научно-технических источников информации

Реализация программного сервиса, связанного с обучаемыми цифровыми аватарами, невозможна без опоры на современные научно-технические достижения в области веб-разработки, распределённых вычислений, хранения и обработки мультимедийных данных, а также технологий машинного обучения. В данном разделе рассматриваются ключевые архитектурные и инфраструктурные решения, а также программные компоненты, позволяющие реализовать сервис на практике.

2.1 Концептуальные и архитектурные решения

2.1.1 Общий архитектурный паттерн. Микросервисный и монолитный подходы

В качестве базового архитектурного паттерна для дипломного проекта был выбран микросервисный подход. Данный выбор обусловлен спецификой разрабатываемого приложения, которое представляет собой комплексный сервис с явно выраженным отдельными функциональными модулями: интерфейсом взаимодействия с пользователями, модулем генерации текстовых и аудиоответов, процессом обучения моделей, а также системой хранения и управления данными.

Микросервисная архитектура позволяет эффективно распределить обязанности между компонентами, разрабатывать их независимо друг от друга и проводить параллельную работу над проектом в рамках команды.

Кроме того, такой подход значительно упрощает процессы масштабирования. При росте нагрузки можно увеличивать ресурсы только для тех компонентов, которые в этом нуждаются, не затрагивая остальную систему.

В качестве альтернативы можно было бы рассмотреть монолитную архитектуру, предполагающую разработку всех функций сервиса в рамках единого приложения. Несмотря на относительную простоту реализации на начальных этапах, монолитное приложение имеет ряд существенных ограничений: усложняется внесение изменений, требующих частых релизов, и масштабирование становится менее гибким, так как масштабируется весь монолит целиком, а не отдельные его части.

Кроме того, использование микросервисов упрощает интеграцию различных технологий и фреймворков, что особенно важно в контексте дипломного проекта, в котором задействованы разнообразные инструменты и подходы, такие как NLP и TTS модели, брокеры сообщений и несколько видов баз данных.

Таким образом, микросервисная архитектура обеспечивает необходимую гибкость, отказоустойчивость, возможность масштабирования отдельных компонентов и параллельную разработку в рамках команды, что делает её оптимальным выбором в рамках дипломной разработки.

2.1.2 Паттерн взаимодействия микросервисов. Хореография и оркестрация

При построении взаимодействия между микросервисами в рамках дипломного проекта был выбран паттерн оркестрации. Он предполагает наличие централизованного компонента — оркестратора, который управляет потоками данных и координирует вызовы между отдельными модулями системы.

Такой подход хорошо подходит для систем, в которых важна наблюдаемость, чёткое управление последовательностью операций, а также необходимость логирования состояния на каждом этапе обработки.

Альтернативным решением могла бы быть хореография, где каждый микросервис действует автономно, реагируя на события, опубликованные другими сервисами, и самостоятельно принимая решения о своей логике работы. Хореография более гибка и масштабируема в условиях loosely coupled-сервисов, однако она увеличивает сложность отладки, снижает прозрачность процессов и требует высокой зрелости всей архитектуры.

В контексте данного дипломного проекта, где важны управляемость процессов, централизованная маршрутизация запросов, а также простота расширения и логирования, паттерн оркестрации оказался более подходящим.

Он позволяет точно определять, в какой момент вызывается тот или иной модуль, а также централизованно обрабатывать ошибки и реализовывать управление зависимостями между компонентами.

Таким образом, выбор в пользу оркестрации обеспечивает необходимый баланс между контролем, гибкостью и надёжностью при построении архитектуры сервиса в рамках дипломной работы.

2.1.3 Модель взаимодействия компонентов. Асинхронность, параллельность и конкурентность

Одним из ключевых архитектурных решений в рамках дипломной разработки стало использование асинхронного взаимодействия между компонентами системы. Данный подход был выбран с учётом природы выполняемых задач, в частности генерации текстов и синтеза речи, которые требуют значительных вычислительных ресурсов и занимают заметное время на выполнение.

Асинхронная модель позволяет выполнять такие задачи в фоновом режиме, не блокируя основной поток обработки, и даёт возможность пользователю продолжать взаимодействие с системой, в том числе с другими аватарами, не дожидаясь завершения ресурсоёмких операций.

Для лучшего понимания выбора архитектурной модели необходимо кратко рассмотреть отличия между асинхронностью, параллельностью и конкурентностью. Асинхронность — это способ организации кода, при котором выполнение может быть приостановлено и возобновлено позже, что особенно эффективно в условиях операций ввода-вывода (I/O). Параллельность же подразумевает физическое исполнение нескольких задач одновременно, как правило — на разных ядрах процессора. Конкурентность — более широкое понятие, обозначающее возможность выполнения нескольких задач, не обязательно параллельно, но с чередованием выполнения или асинхронным управлением.

Асинхронное исполнение часто реализуется в виде событийного цикла, в то время как параллельное — через многопоточность или многопроцессность. Последние требуют более сложной координации и синхронизации, что затрудняет масштабирование и отладку, особенно в распределённой архитектуре.

В отличие от синхронного исполнения, где каждый компонент ожидает завершения предыдущего действия, асинхронный подход обеспечивает более высокую отзывчивость и масштабируемость сервиса. Синхронные вызовы проще в реализации, но в условиях высоких задержек или большого количества параллельных запросов они ведут к блокировке потоков и снижению общей производительности.

Параллельное исполнение с использованием многопоточности могло бы частично компенсировать блокировки, однако такая реализация сложнее в обслуживании и, забегая вперёд, стоит отметить, что в случае с Python (язык будет подробнее рассмотрен в следующем разделе) данный подход дополнительно осложняется ограничениями, связанными с GIL (Global Interpreter Lock), характерным для языка программирования Python, технический выбор которого будет обоснован в последующих разделах. Кроме того, в контексте работы с языковыми и звуковыми моделями важно учитывать, что генерация занимает значительное время на GPU, и в условиях параллельного запуска таких операций быстро наступает исчерпание доступных ресурсов, что приводит либо к очередям, либо к отказам в обработке.

Асинхронная модель позволяет более эффективно распределять ресурсы между задачами, обрабатывая запросы по мере готовности моделей к генерации, а не блокируя систему в ожидании ответа от перегруженного компонента.

Таким образом, выбор асинхронного взаимодействия в рамках дипломного проекта продиктован особенностями предполагаемой нагрузки: интенсивные операции генерации и обучения, непредсказуемое время отклика, высокая параллельность пользовательских запросов и необходимость устойчивой обработки в условиях сетевой неопределённости и ограниченных вычислительных ресурсов.

2.1.4 Модель хранения данных

В рамках дипломной разработки рассматриваемый сервис взаимодействует с принципиально разными типами данных, что требует осознанного выбора подходящих систем хранения. Ключевым источником анализа стал труд [2], в котором систематизированы подходы к выбору хранилищ в зависимости от характера данных и типов нагрузки.

Первым типом данных являются структурированные сущности: информация о пользователях, профилях и параметрах аватаров, чат-сессиях, настройках и доступах. Для этих целей была выбрана реляционная база данных, которая обеспечивает консистентность, связность данных и мощный язык запросов (SQL), необходимый для аналитических задач и построения сложных фильтров. Альтернативой могли бы выступать документо-ориентированные

СУБД, например MongoDB, однако в данном случае иерархическая структура с чёткими отношениями между таблицами оказалась более подходящей.

Вторым важным типом данных являются большие бинарные объекты — аудио- и видеоматериалы, загружаемые пользователями для обучения аватаров. Хранение таких файлов в реляционной базе данных представляется неэффективным с точки зрения производительности и масштабирования. Для этих целей было выбрано объектное хранилище, поддерживающее интерфейс S3, что обеспечивает совместимость с индустриальными инструментами и удобство масштабируемого хранения. Возможными альтернативами могли бы быть файловые хранилища или blob-хранилища в рамках SQL-СУБД, однако они уступают по скорости доступа, гибкости маршрутизации и отказоустойчивости.

Наконец, особую категорию составляют внутренние технические данные — журналы событий, логи ошибок, статистика работы сервисов. Эти данные имеют потоковую природу и требуют быстрого добавления и поиска по ключу. Для этих целей используется key-value-хранилище, оптимизированное под работу с большим объёмом логов в режиме реального времени. Альтернативой могла бы быть файловая система или хранение логов в PostgreSQL, но это создаёт избыточную нагрузку и снижает общую отзывчивость системы.

Таким образом, комбинация трёх типов хранилищ — реляционного, объектного и ключ-значение — обеспечивает логическую и техническую оптимальность решения задачи хранения в условиях разнородных данных и разнообразных сценариев их использования.

2.1.5 Паттерн клиентской архитектуры. Модели МРА и SPA

В качестве модели клиентской части приложения был выбран паттерн SPA (Single Page Application), который предполагает однократную загрузку основного интерфейса и последующую динамическую подгрузку данных через API без полной перезагрузки страницы.

Выбор данного подхода обусловлен необходимостью обеспечить высокую отзывчивость и непрерывность пользовательского взаимодействия, особенно в условиях частых обращений к функциональности, связанной с управлением аватарами и обменом сообщениями в режиме реального времени. Такой сценарий требует гибкой клиентской логики и быстрого отклика ин-

терфейса, что достигается за счёт исключения циклов полной перерисовки страницы.

Фронтенд построен с опорой на современные принципы реактивного взаимодействия, реализованные средствами, не требующими использования тяжёлых фреймворков. Это позволило упростить архитектуру, сохранить единство технологического стека и сократить затраты времени на разработку — особенно актуальные факторы в условиях ограниченных ресурсов дипломного проекта.

Альтернативой SPA могла бы стать модель МРА (Multi Page Application), при которой каждое взаимодействие пользователя приводит к загрузке новой страницы с серверным рендерингом содержимого. Несмотря на простоту реализации, данный подход менее эффективен в сценариях, требующих высокой интерактивности и поддержки многосессионного взаимодействия.

Таким образом, архитектура SPA была выбрана как наиболее подходящая, обеспечивая баланс между интерактивностью пользовательского интерфейса и эффективностью разработки.

2.2 Программные средства и технологии реализации

2.2.1 Выбор языка программирования

Ключевым техническим решением в рамках дипломной разработки стал выбор языка программирования Python в качестве основного инструмента для реализации серверной логики.

Python был выбран по совокупности следующих причин: широкая поддержка асинхронного программирования, богатая экосистема библиотек для работы с данными, интеграция с фреймворками машинного обучения, а также высокая читаемость и скорость разработки, что особенно важно в условиях ограниченного времени и ресурсов.

Дополнительно, наличие развитых средств работы с языковыми и звуковыми моделями (таких как библиотеки HuggingFace, Transformers, Torchaudio и др.) делает Python естественным выбором для построения сервисов, основанных на обработке естественного языка и синтезе речи.

В качестве возможных альтернатив могли бы рассматриваться языки с более высокой производительностью — например, Go или Rust, однако в кон-

тексте дипломного проекта ключевым критерием является не максимальная эффективность исполнения, а возможность быстрой реализации, прототипирования и интеграции с существующими ML-инструментами.

Кроме того, Python активно используется в академической среде, что облегчает повторяемость, демонстрацию и возможное масштабирование решения в дальнейшем, в том числе на инфраструктуре облачных сервисов, таких как Google Colab, Kaggle.

Выбор Python, таким образом, органично вписывается в общую архитектуру разрабатываемого сервиса: его асинхронная модель исполнения сочетается с выбранным серверным фреймворком и системой обмена сообщениями, а гибкость и выразительность языка позволяют эффективно реализовать как управляющую логику, так и интеграцию с компонентами машинного обучения.

2.2.2 Веб интерфейс

При выборе технологии для реализации веб-интерфейса в рамках настоящей выпускной квалификационной работы решающее значение имели два основных аспекта: ограниченный опыт участников команды с использованием современных JavaScript-фреймворков и стремление к технологической унификации проекта с использованием единого языка программирования.

Исходя из этих условий, было принято решение о выборе библиотеки FastHTML, написанной на языке Python, которая предоставляет средства для генерации HTML-разметки и интеграции с библиотекой HTMX. Последняя представляет собой лёгкую JavaScript-библиотеку, позволяющую динамически обновлять отдельные элементы страницы посредством асинхронных HTTP-запросов, исключая необходимость полной перезагрузки веб-страницы.

Подобное техническое решение позволило обеспечить интерактивность пользовательского интерфейса без привлечения более сложных и ресурсоёмких клиентских фреймворков, что существенно упростило процессы разработки и последующего сопровождения. Минималистичный подход к реализации веб-интерфейса особенно оправдан в контексте проекта, где основное внимание уделяется реализации серверной логики, интеграции моделей машинного обучения и обработки данных.

В качестве альтернативы рассматривалась возможность использования современных клиентских фреймворков, таких как React, Vue или Angular. Несмотря на их обширные функциональные возможности и развитую экосистему, применение данных инструментов требовало бы освоения специфических технологий (например, JSX, шаблоны Vue), создания дополнительной инфраструктуры сборки и управления зависимостями, что существенно усложнило бы архитектуру и повысило нагрузку на разработчиков.

Дополнительным недостатком использования таких решений стало бы неизбежное разделение проекта на две отдельные части (клиентскую и серверную), требующие различных инструментов, языков и процессов интеграции, что негативно повлияло бы на простоту сопровождения и целостность проекта в условиях ограниченных ресурсов и учебной специфики работы.

Таким образом, выбор технологии FastHTML в сочетании с HTMX представляется наиболее целесообразным в рамках данной работы, обеспечивая оптимальное сочетание интерактивности интерфейса и простоты разработки и сопровождения.

2.2.3 Серверная часть. Оркестратор

В основе серверной архитектуры дипломного проекта лежит компонент-оркестратор, реализованный на базе библиотеки FastAPI [3]. Выбор FastAPI был обусловлен рядом преимуществ, ключевыми среди которых являются высокая производительность и полная поддержка асинхронного исполнения, что обеспечивает эффективную обработку большого количества одновременных запросов в условиях интенсивной сетевой и вычислительной нагрузки.

В качестве альтернативных решений рассматривались популярные серверные фреймворки Django REST Framework и Flask. Однако оба решения ориентированы преимущественно на синхронную модель исполнения, что делает их менее подходящими в контексте проекта, где значительная часть операций связана с взаимодействием между микросервисами и внешними асинхронными источниками данных.

В частности, Flask отличается простотой и минимализмом, но требует дополнительных усилий при построении масштабируемой архитектуры, а также не обладает встроенными средствами интеграции с асинхронной обра-

боткой, что ограничивает его применимость в высоконагруженных распределённых системах.

Django REST Framework, в свою очередь, является мощным и функциональным инструментом, но его асинхронные возможности находятся на этапе развития и не позволяют достичь той же степени гибкости и производительности, которую предоставляет FastAPI.

Кроме того, важным преимуществом FastAPI является отсутствие жёстких ограничений на структуру проекта, что позволяет свободно организовывать взаимодействие между модулями, интегрировать внешние хранилища, брокеры сообщений и другие инфраструктурные компоненты без необходимости следовать строго заданной архитектурной модели.

Таким образом, использование FastAPI в качестве основного инструмента реализации оркестратора представляется наиболее обоснованным решением, соответствующим требованиям производительности, масштабируемости и гибкости, предъявляемым к современным микросервисным приложениям в условиях дипломного проектирования.

2.2.4 Серверная часть. Модуль работы с языковой и звуковой моделью

Одним из ключевых компонентов серверной архитектуры является модуль генерации, отвечающий за формирование текстовых и аудиоответов в ходе взаимодействия пользователя с аватаром. Модуль функционирует по асинхронной модели, обеспечивая выполнение ресурсоёмких операций вне основного цикла обработки и получая команды от оркестратора посредством брокера сообщений, реализующего принцип FIFO (first in, first out).

В качестве основной технологической базы для построения языковых и звуковых моделей были выбраны открытые библиотеки и модели из экосистемы HuggingFace [4]. Данный выбор обусловлен высоким уровнем зрелости инструментов, поддержкой современного стека NLP и TTS-технологий, а также наличием предварительно обученных моделей и инфраструктуры для дообучения и тонкой настройки.

Одним из важнейших преимуществ HuggingFace является открытость исходного кода и соблюдение свободных лицензий, что позволяет исключить зависимость от коммерческих поставщиков и обеспечивает возможность

развёртывания модели в полностью контролируемой среде. Кроме того, использование данных инструментов снижает финансовые издержки на этапе разработки и делает проект более устойчивым к изменениям внешней среды.

В качестве альтернативных решений могли бы рассматриваться облачные сервисы от крупных вендоров, таких как Google Cloud (Dialogflow, Text-to-Speech), Amazon Web Services (Polly, Lex), Microsoft Azure и др. Несмотря на высокую точность и удобство интеграции, эти решения обладают рядом проблем, которые включают в себя ограниченную доступность в отдельных регионах, зависимость от внешнего API, стоимость при масштабируемом использовании, а также ограниченные возможности кастомизации моделей под специфические задачи.

В контексте настоящей дипломной работы, где важны автономность, воспроизводимость и контроль над конфигурацией, использование локальных open-source решений представляется наиболее рациональным и перспективным подходом.

Одной из главных задач для реализации пайплайнов обучения и генерации является выбор языковой модели. Для оценки производительности различных вариантов LLM был выбран ряд общепринятых бенчмарков, которые позволяют объективно оценить способности моделей от задач на базовые знания и логику до качества диалога и мультиязычности. Определяющими стали:

- MMLU (Massive Multitask Language Understanding) — бенчмарк, который включает задачи из 57 различных академических дисциплин [5];
- Russian SuperGLUE — аналог англоязычного SuperGLUE адаптированный под русский язык, включающий в себя тесты на классификацию, парное сопоставление текстов, логические выводы и пр. [6];
- MERA (Multilingual Evaluation of Reasoning Abilities) — относительно новый бенчмарк, состоящий из 21 задания по разным навыкам модели. Он позволяет выявить, насколько хорошо модель справляется с выводами вне англоязычного контекста [7];
- Chatbot Arena — представляет собой систему парных сравнений моделей, где пользователи выбирают между ответами двух анонимных систем, что позволяет выявить предпочтения людей в реальном общении [8];

- LIBRA (Linguistically Informed Benchmark for Russian AI) — оценивает качество LLM именно в контексте русского языка, проверяя на синтаксический анализ, морфологию, семантику и другое [9].

Проанализировав общую картину по всем бенчмаркам было выделено три серии открытых языковых моделей, которые показали высокие показатели: LLaMA [10], Mistral [11], Qwen [12] и DeepSeek [13], имеющий наивысшие оценки. LLM семейства LLaMA, разработанные Meta AI, имеют хороший результат в задачах понимания контекста. Mistral обладает высокой эффективностью за счет своей компактности. Qwen, разработанная Alibaba Cloud, обладает высоким качеством генерации при относительно небольшом количестве параметров. Главным их преимуществом является открытая архитектура, что позволяет использовать их в пользовательских и научных проектах. Однако, важно отметить, что данные, на которых обучались эти модели, в основном ориентированы на английский язык, поэтому из-за наличия небольшого количества текста на других языках в их наборах обучающих данных их общая производительность имеет отрицательных эффект при генерации на русском. Отдельно стоит выделить модель DeepSeek, которую разработала китайская группа исследователей. Несмотря на то, что она в своем обучающем датасете имеет преимущественно китайские и английские текста, ее архитектура и объем позволяют ей достигать лидирующих позиций в большинстве бенчмарков. Русский язык является одним из наиболее ресурсоемких языков с точки зрения морфологии, синтаксиса и семантики, что требует от LLM языковой адаптации и богатого лингвистического представления в наборах данных. Для решения этой проблемы были разработаны MTS AI, GigaChat и YandexGPT, которые демонстрируют высокое качество генерации, превосходя своих флагманских конкурентов, ориентированных на английский язык, в том, что касается обработки и генерации текста на русском, но имеют закрытый исходный код, что делает невозможность их использование в рамках научного проекта. Поэтому высокой популярностью пользуются открытые модели, среди которых можно выделить серии Saiga [14], ruGPT [15], ruadapt [16] и Vikhr [17]. Они были дообучены на базе лидирующих англоязычных LLM и дополнительно настроены для улучшения генерации и понимания текста именно в русскоязычном контексте. В таблице 2 представлены средние значения по бенчмарку PingPong [18], оценивающим языковые модели в их способности поддерживать многоразовую беседу, сохраняя выбранную

роль и персону. Как из нее видно LLM семейства Saiga входят в число лидеров, что делает ее оптимальным выбором для задачи генерации текста в речевом стиле конкретного человека.

Таблица 2 — Сравнение моделей по бенчмарку PingPong

Модель \ PingPong бенчмарк	Average Score	Average Score
deepseek_v3_0324	4.79±0.04	4.93
deepseek_v3	4.65±0.06	4.79
sainemo_remix_12b	4.61±0.07	4.64
saiga_yandexgpt_8b	4.69±0.05	4.71
saiga_nemo_12b_v3	4.68±0.06	4.63
mistral_nemo_vikhr_dostoevsky_slerp_12b	4.55±0.06	4.59
saiga_gemma3_12b	4.73±0.07	4.74

2.2.5 Серверная часть. Технология распознавания речи

Для преобразования аудиоданных в текст в рамках пайплайна обучения использовалась модель автоматического распознавания речи Whisper, разработанная компанией Open AI [19]. Она является представителем open-source систем ASR (Automatic Speech Recognition), основанную на архитектуре трансформеров и обученную на большом объеме аудиоматериалов, включающих различные интервью, лекции, подкасты, видеоконтент. Благодаря этому Whisper хорошо справляется с переводом потенциальных источников информации для сбора датасета под конкретного человека. В качестве альтернативных решений рассматривались облачные сервисы, такие как Google Speech-to-Text, однако их применение ограничено лицензионными условиями и необходимостью постоянного интернет-соединения. Кроме того, была проанализирована модель Wav2Vec 2.0 [20], разработанная Facebook AI Research, которая демонстрирует высокую точность на английском языке, но требует дополнительного дообучения для русского, что значительно усложняет ее интеграцию в проект и увеличивает требования к вычислительным ресурсам. Для извлечения аудиодорожки из видеофайлов выбрана библиотека MoviePy, которая имеет удобный и простой интерфейс, что упрощает процесс программной реализации данного этапа пайплайна. Данный модуль является удобной надстройкой над библиотекой FFmpeg, упрощая функции

нарезки, конкатенации, преобразования форматов, обеспечивая высокий уровень абстракции.

2.2.6 Серверная часть. Брокер сообщений

Для организации взаимодействия между микросервисами в архитектуре дипломного проекта применяется брокер сообщений, играющий ключевую роль в обеспечении асинхронной коммуникации и устойчивости системы к пиковым нагрузкам.

В качестве основного инструмента был выбран Apache Kafka [21], широко распространённая распределённая платформа для обработки потоков данных в реальном времени. Kafka обеспечивает высокую пропускную способность, гарантированную доставку сообщений, а также масштабируемость за счёт распределённой архитектуры, что делает её оптимальным решением для сервисов, требующих надёжной и непрерывной передачи данных между компонентами.

Выбор Kafka также обоснован её зрелостью, широкой поддержкой в сообществе и богатой документацией, а также наличием опыта работы с данной системой в команде, что позволило сократить время на внедрение и конфигурацию.

В качестве альтернатив могли бы быть использованы RabbitMQ — брокер с поддержкой расширенных шаблонов маршрутизации сообщений, или Memphis — более молодой, но активно развивающийся инструмент с удобным интерфейсом и низким порогом входа. Тем не менее, в условиях проекта, ориентированного на обработку ресурсоёмких задач и высокую частоту межсервисных взаимодействий, Kafka продемонстрировала наибольшую пригодность благодаря своей отказоустойчивости и способности обрабатывать большие объёмы сообщений в режиме реального времени.

Таким образом, применение Apache Kafka представляется технически обоснованным решением, соответствующим требованиям надёжности, масштабируемости и производительности, необходимым для устойчивой работы распределённого сервиса в рамках дипломного проекта.

2.2.7 Инфраструктура и развертывание

Микросервисная архитектура, выбранная в рамках дипломной разработки, предполагает наличие большого числа взаимосвязанных компонентов, каждый из которых выполняет строго определённую функцию и может развиваться независимо от остальных. Такая модульность увеличивает гибкость и масштабируемость системы, но одновременно усложняет процессы развертывания, настройки и сопровождения.

Для минимизации издержек на конфигурацию окружения и унификацию подходов к развертыванию различных сервисов было принято решение использовать технологию контейнеризации на основе Docker [22]. Docker предоставляет изолированную среду выполнения для каждого из компонентов, позволяя гарантировать одинаковое поведение приложений вне зависимости от целевой платформы.

Дополнительно применяется инструмент Docker Compose, обеспечивающий декларативное описание всей системы и автоматизацию процессов сборки, настройки и запуска сервисов в едином командном интерфейсе. Такой подход значительно упрощает работу с инфраструктурой на этапе разработки, тестирования и демонстрации, что особенно важно в условиях дипломного проектирования, где критичны воспроизводимость и предсказуемость поведения среды.

В качестве альтернативы контейнеризации могло бы рассматриваться использование виртуальных машин. Однако данный подход предполагает более высокие накладные расходы, меньшую гибкость в масштабировании и требует отдельной настройки каждой инстанции системы, что затрудняет управление и автоматизацию развёртывания.

Таким образом, применение Docker и связанных с ним инструментов позволило выстроить устойчивую и удобную в сопровождении инфраструктуру, оптимально подходящую для микросервисной архитектуры, реализуемой в рамках дипломного проекта.

2.2.8 Итог по архитектуре

Таблица 3 — Архитектурные решения и обоснование выбора

Область	Выбрано	Альтернативы	Ключевые аргументы выбора
Архитектурный паттерн	Микросервисы	Монолит	Гибкое масштабирование по компонентам. База для параллельной работы команды. Упрощённая интеграция разнородных технологий
Взаимодействие микросервисов	Оркестрация	Хореография	Требуется центральный контроль, единый маршрутизатор запросов, упрощённый контроль точек отказов
Модель исполнения	Асинхронность	Синхронность или многопоточность	Высокая отзывчивость при I/O-нагрузке. Нет затрат на синхронизацию потоков.
Модель хранения	RDBMS + S3 Object Store + Key–Value	Только SQL. Только NoSQL. Локальная файловая система.	Выбор нативно продиктован формой нагрузки и видом данных для хранения. Структурные данные в SQL. Крупные бинарные объекты — в S3. Логи и временные статусы — в KV-хранилище.
Клиентская архитектура	SPA	MPA	Непрерывный UX, частые AJAX-запросы для диалогов.

Таблица 4 — Выбранный технологический стек и рассмотренные альтернативы

Компонент	Технология	Альтернативы	Причины выбора
Язык программирования	Python	Go	Широкая экосистема как для веб-разработки так и для ML
Оркестратор (API)	FastAPI	Django REST (DRF), Flask	Полноценный async. Минимальные требования к структуре проекта от фреймворка
Модуль NLP / TTS	HuggingFace Transformers / TTS	Google Cloud TTS, Azure Cognitive, ElevenLabs	Открытый код библиотек, бесплатный доступ к необходимому спектру облачного функционала
Брокер сообщений	Apache Kafka	RabbitMQ; Memphis	Опыт работы с платформой, так как в контексте работы от брокера требуется базовый функционал
Веб-интерфейс	FastHTML + HTMX	React, Vue, Angular	Единый язык проекта. Отсутствие опыта у команды в работе с JS-фреймворками
Развёртывание	Docker + Docker Compose	Виртуальные машины	Унификация интерфейса для локальной разработки, упрощенные условия для потенциального деплоя продукта

3 Постановка задачи

Основной целью разрабатываемого сервиса является предоставление пользователям удобного и понятного интерфейса для взаимодействия с цифровыми двойниками реальных личностей — интеллектуальными аватарами. Сервис должен позволять не только вести диалог с аватаром посредством текстового и аудио общения, но и иметь возможность полноценного обучения аватара с использованием широкого спектра материалов, таких как тексты, аудио и видео. В результате аватар должен максимально точно воспроизводить индивидуальные особенности манеры общения и эмоционального отклика выбранного человека.

Для достижения поставленной цели необходимо реализовать следующие технические требования к минимальной конфигурации сервиса:

- Разработка веб-интерфейса, предоставляющего пользователям возможность взаимодействовать с созданными аватарами, задавать им вопросы и получать текстовые и аудио ответы в реальном времени.
- Реализация удобного и интуитивного интерфейса для процесса обучения аватаров, который позволит пользователям загружать материалы различных форматов (тексты, аудио и видеозаписи) для последующей обработки и обучения моделей.
- Обеспечение стабильности и надежности процесса обучения языковых и звуковых моделей, который является наиболее ресурсоёмкой частью работы сервиса и требует эффективного управления вычислительными ресурсами.
- Поддержка возможности интеграции дополнительных источников данных и сторонних сервисов, таких как социальные сети и мессенджеры, для получения дополнительных материалов, обогащающих процесс обучения аватаров.
- Реализация механизма хранения и управления данными разного типа и формата, включая метаданные аватаров, пользовательские данные, текстовые и аудио материалы для обучения, а также системные логи.

Таким образом, итоговая архитектура должна обеспечивать гибкость, производительность и простоту использования, позволяя пользователям не только взаимодействовать с виртуальными аватарами, но и активно участвовать в процессе их создания и улучшения.

4 Сущность решения задачи

Описанные выше технические требования нашли своё отражение в сущности решения задачи предоставления удобного сервиса для создания, обучения и общения с цифровыми двойниками. Решение было дифференцировано на несколько независимых сущностей, каждая из которых была материализована в соответствующие компоненты системы.

4.1 Компоненты решения

4.1.1 Веб интерфейс

Веб интерфейс отвечает за взаимодействие с пользователем и именно с его помощью пользователь будет посыпать запросы внутрь системы. Так же этот компонент системы будет отвечать за то, чтобы потребитель получил результат работы сервиса в удобном и интуитивно понятном формате.

4.1.2 Бекэнд

Бекэнд необходим для запуска запрошенных процедур со стороны пользователя через веб интерфейс. Помимо этого компонент занимается оркестрацией всего сервиса, первичной предобработкой данных, полученных от пользователя и возврата промежуточных и итоговых результатов работы сервиса.

4.1.3 Реляционная база данных

Реляционная база данных предоставляет среду для хранения метаданных пользователя и аватаров, состояния системы, а так же быстрого поиска, вставки и модификации этих данных. Так же для корректной работы системы необходимо, чтобы данные оставались валидными в условиях конкурентного исполнения, что так же является зоной ответственности этого компонента.

4.1.4 ML сервис

ML сервис занимается непосредственным обучением моделей и генерацией данных для создания ответа и его озвучки на запрос пользователя с помощью ранее обученных моделей. Причем после обучения модели для конкретного цифрового двойника этот сервис должен обеспечить возможность быстрого взаимодействия без повторного обучения модели при последующем обращении. Поэтому данный компонент контролирует сохранение и выгрузку весов моделей, полученных в результате процедуры обучения на пользовательских данных.

4.1.5 Брокер сообщений

Брокер сообщений является сердцем всей системы, так как он предоставляет отказоустойчивую и отзывчивую среду для организации межкомпонентного взаимодействия системы.

4.1.6 S3 хранилище

S3 хранилище обеспечивает хранение, обновление и удаление больших по объему данных, которые не укладываются или их хранение является неэффективным в контексте реляционных баз данных.

4.2 Алгоритм взаимодействия

4.2.1 Процедуры регистрации и аутентификации пользователя

Нельзя забывать о том, что мы работаем с данными пользователя, значит, нельзя забывать и о конфиденциальности. Поэтому помимо непосредственного решения поставленной задачи необходимо позаботиться о защищенности данных. Самый простой и надежный способ сделать это - добавить функционал аутентификации пользователей. Таким образом сервис сможет гарантировать, что доступ к данным будет оставаться только у владельца соответствующей учетной записи. На диаграмме процесса регистрации и аутентификации (рис. 2) описаны шаги, которые пользователь проходит от момента

перехода на веб страницу сервиса до перехода в личный кабинет, который предоставляет интерфейс по управлению аватарами.

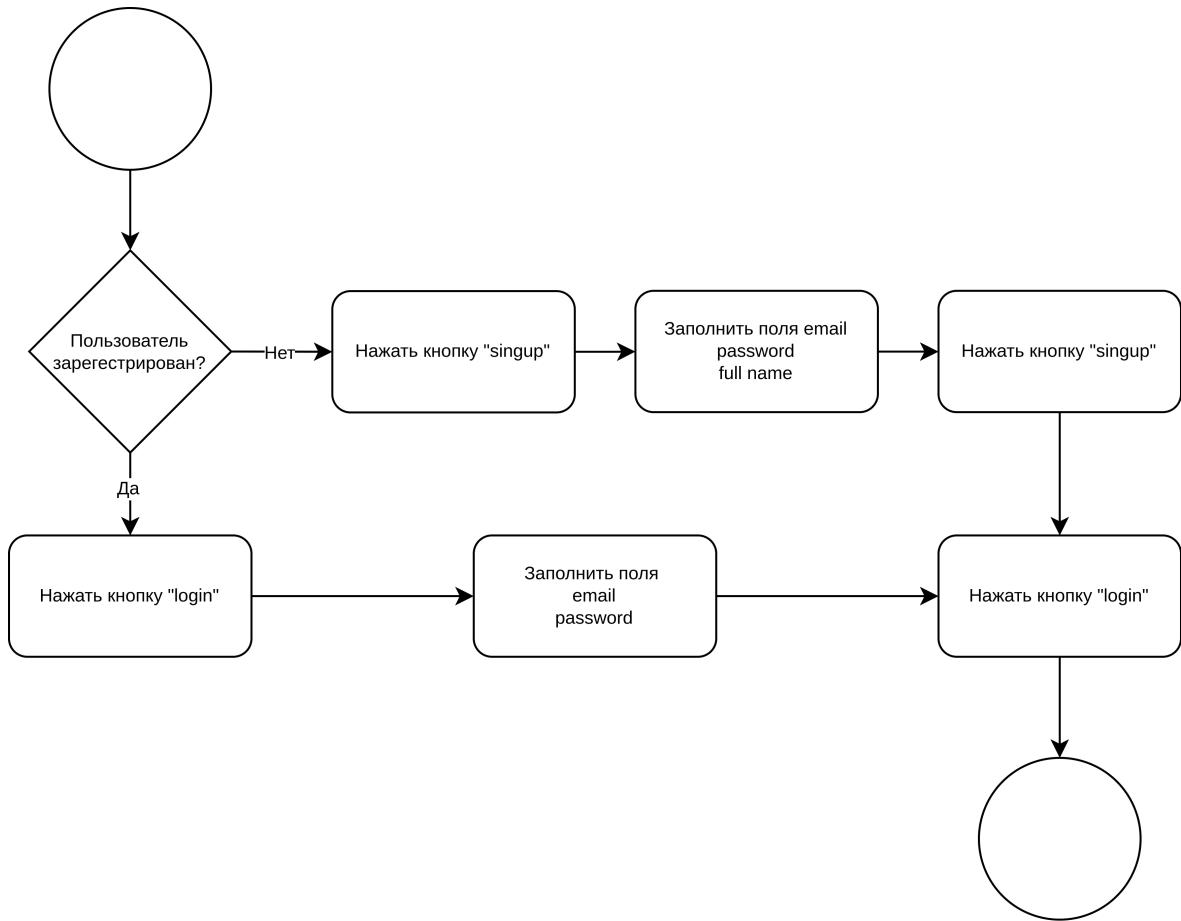


Рисунок 2 — User flow диаграмма процесса регистрации или авторизации

4.2.2 Процедура обучения

Решение имеет микросервисную архитектуру. Это значит, что весь сервис разбит на ряд независимых с точки зрения физического расположения компонент, а взаимодействие между ними происходит через сеть с помощью брокера сообщений. На диаграмме обучения аватара (рис. 3) представлены основные сообщения и события, которые происходят в системе для получения желаемого результата. Так же на user flow диаграмме (рис. 4) показано какие именно действия производит пользователь для обучения существующего или создания и обучения нового аватара.

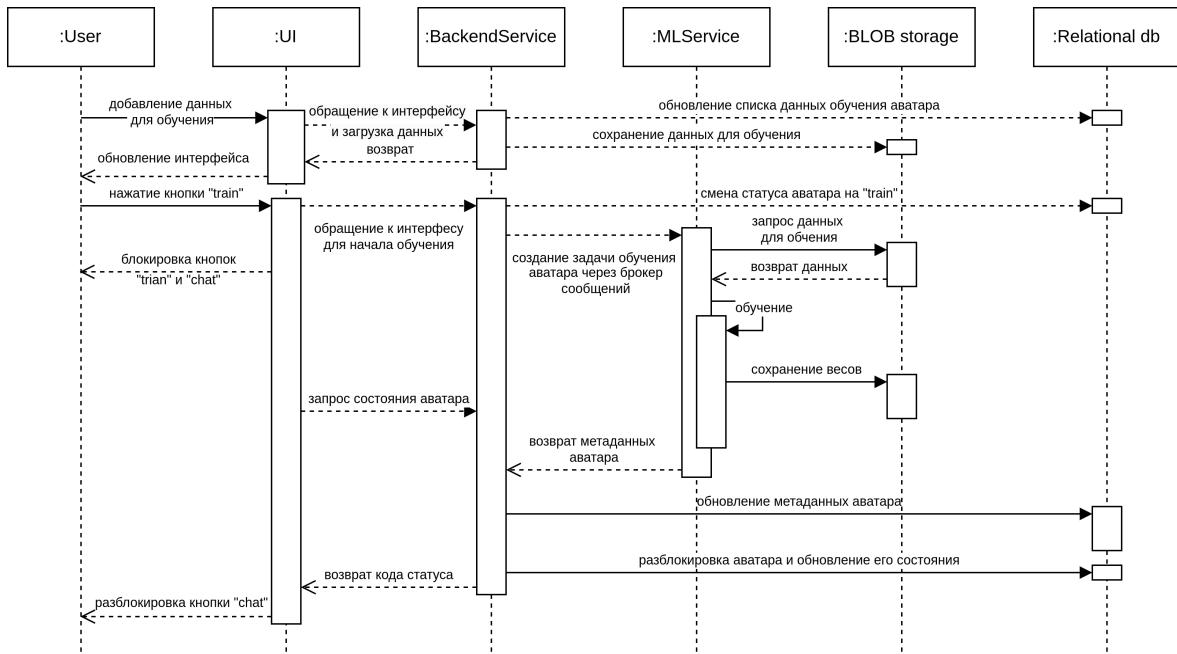


Рисунок 3 — UML-диаграмма процесса обучения

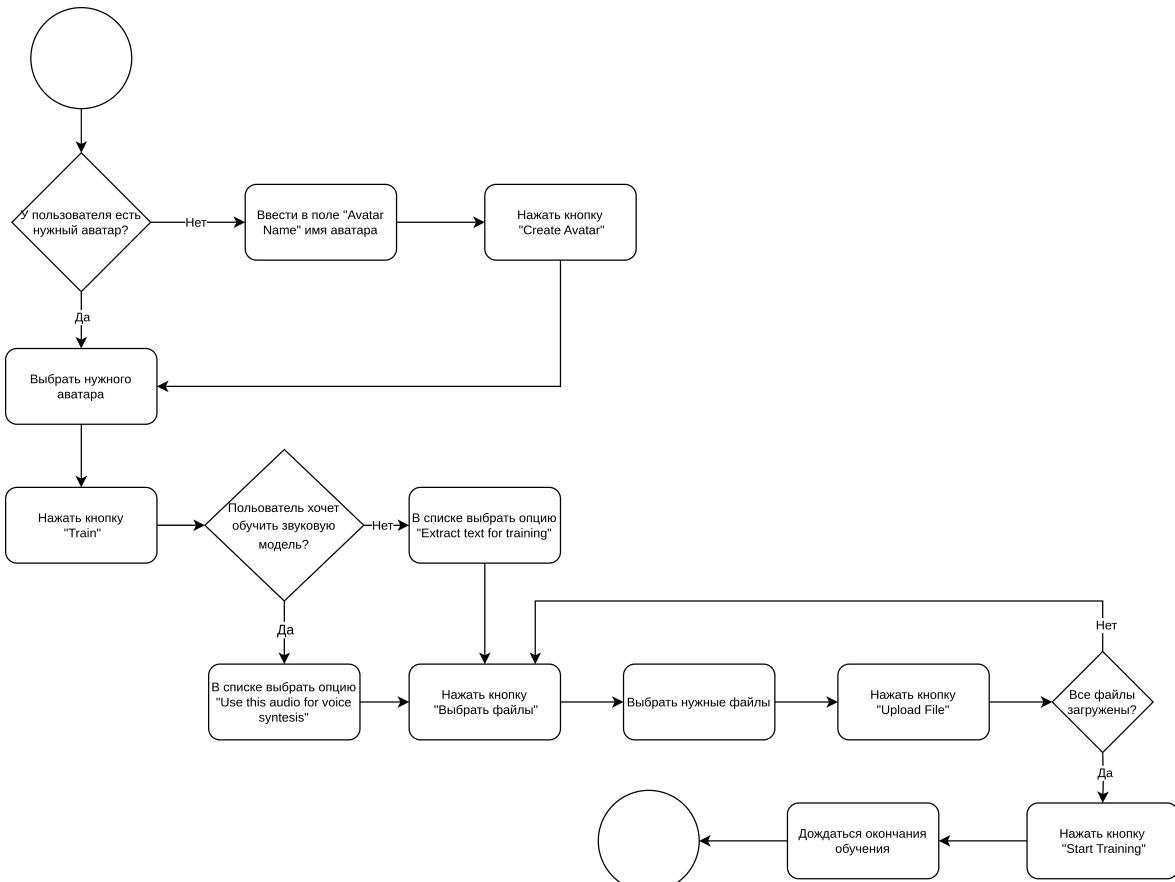


Рисунок 4 — User flow диаграмма процесса обучения (пользователь уже выполнил вход)

4.2.3 Процедура взаимодействия с аватаром

Аналогично процессу обучения на диаграмме процесса генерации (рис. 5) отражены основные сообщения и события, через которые проходит система для предоставления пользователю сгенерированного текстового ответа и его озвучки на запрос. Аналогично представлен путь, через который проходит пользователь для создания запроса и получения ответа на user flow диаграмме (рис. 6)

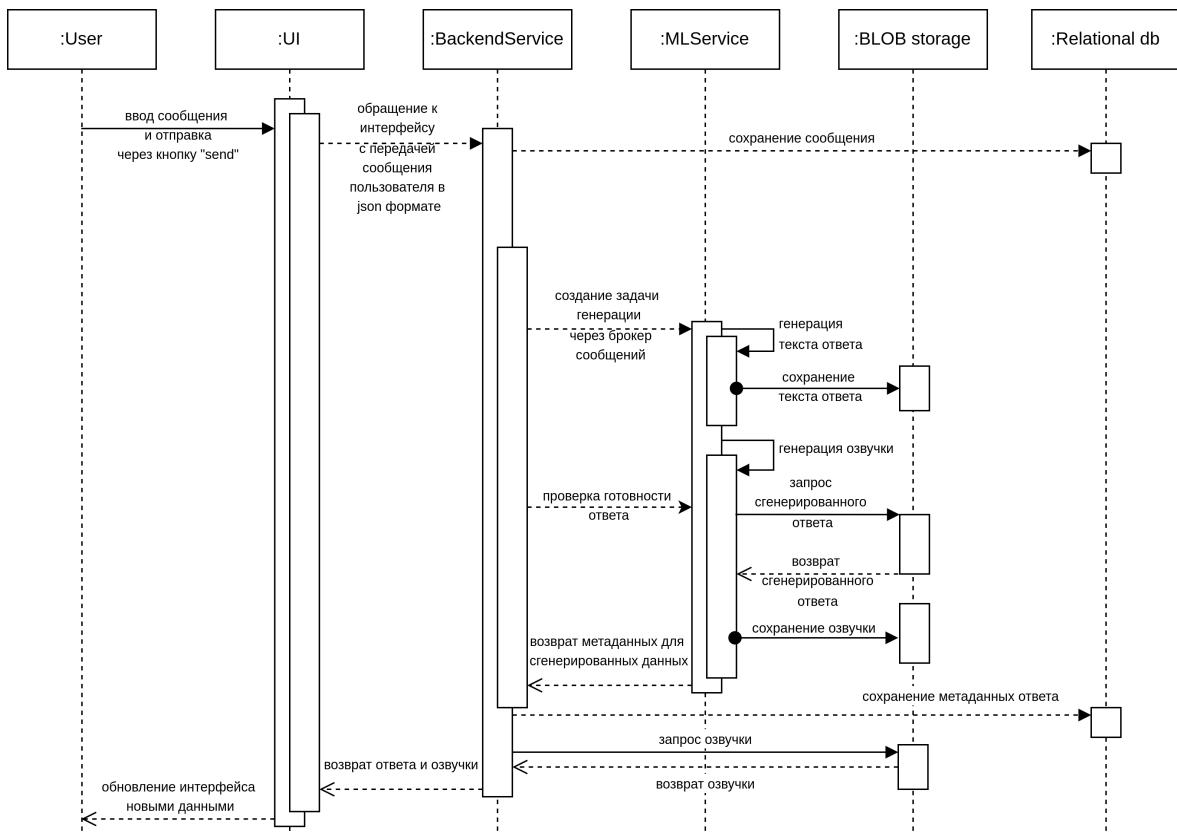


Рисунок 5 — UML-диаграмма процесса генерации

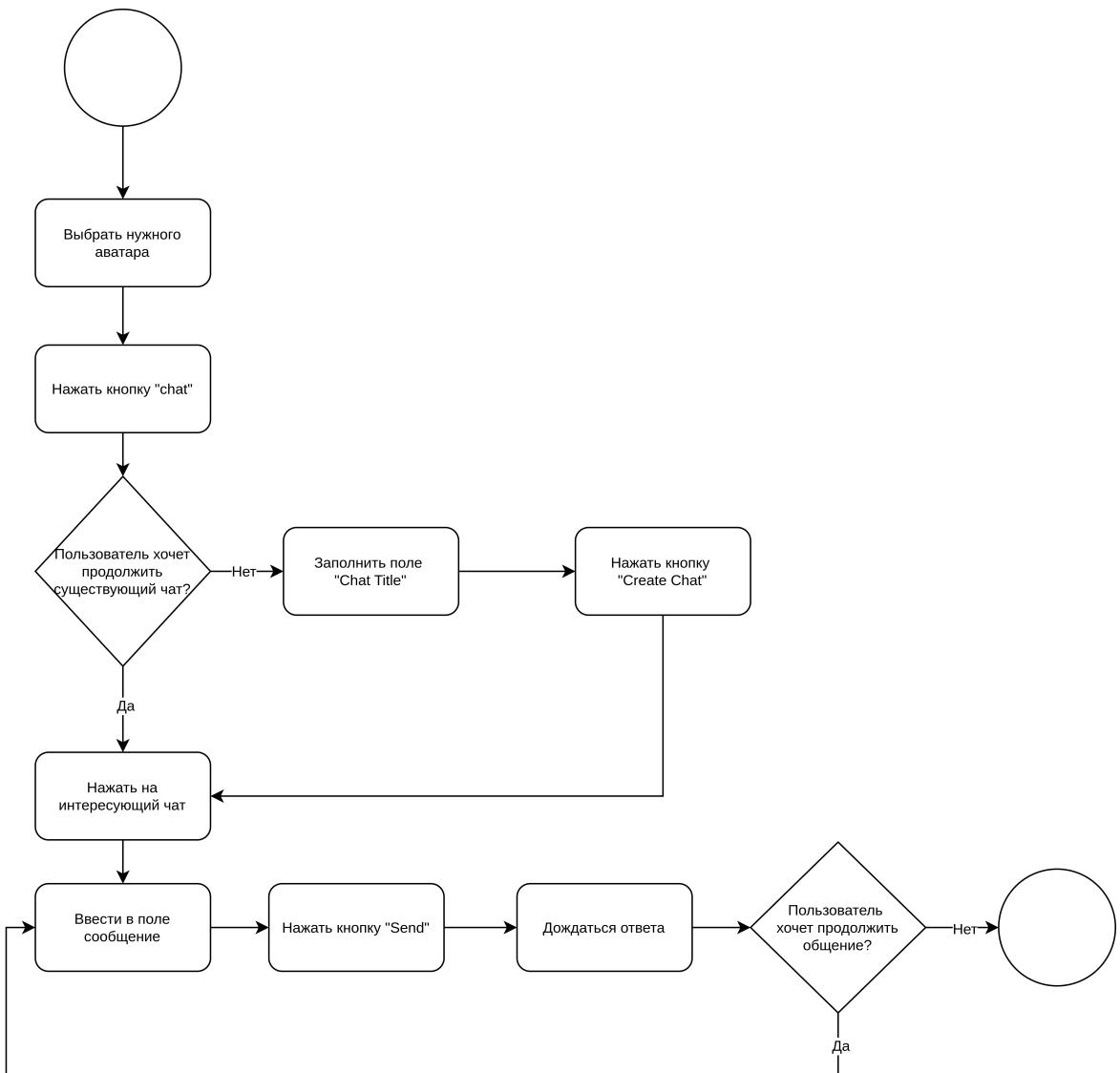


Рисунок 6 — User flow диаграмма процесса взаимодействия с аватаром (пользователь уже выполнил вход)

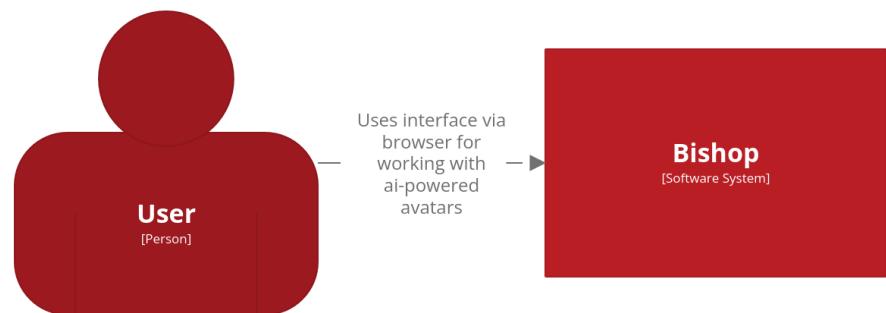


Рисунок 7 — C4 диаграмма для сервиса «Bishop»

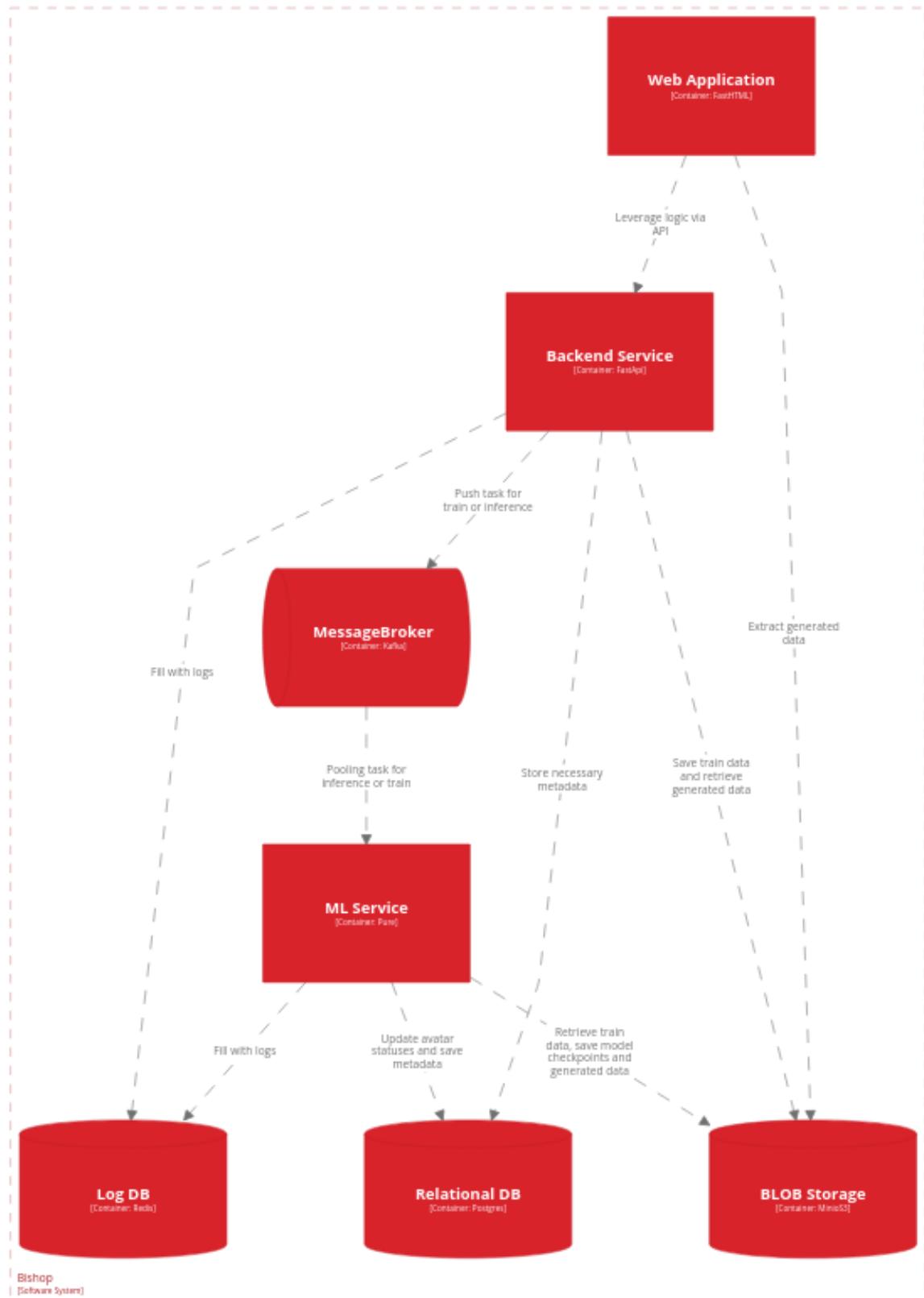


Рисунок 8 — C4 диаграмма для сервиса «Bishop»

5 Построение модели решения задачи

Концепция сервиса опирается на интенсивное использование аппарата машинного обучения. Диаграммы процесса генерации **5** и обучения **3** дают общее понимание процесса работы данного модуля. Далее будут рассмотрены как теоретические аспекты работы с моделями, так и результаты проведённых экспериментов.

5.1 Цели и ограничения

Основная цель данного модуля заключается в организации эффективного взаимодействия пользователя с аватаром через последовательный обмен сообщениями с последующей озвучкой сгенерированных текстовых сообщений. Ключевой особенностью является реалистичная имитация речевого поведения конкретного человека.

Описанная задача общения аватара с пользователем включает генерацию текста с учётом особенностей личности (persona-based text generation) в контексте домена обработки естественного языка (NLP). Дополнительно решается задача синтеза речи с имитацией голоса реального человека (voice imitation). Также в процессе формирования обучающего корпуса решается вспомогательная задача извлечения текста из аудио- и видеоматериалов (text extraction from audio/video).

Необходимо учитывать, что вычислительные ресурсы для генерации текста и аудио намного меньше, чем ресурсы, необходимые для обучения моделей. В целях демонстрации функционала, общение с аватаром будет осуществляться при помощи заранее обученных моделей, размещённых на бесплатных вычислительных мощностях таких сервисов, как Kaggle и Colab. Для демонстрации процесса обучения будут использованы модели небольших размеров с умеренными параметрами, чтобы снизить нагрузку на вычислительные мощности.

5.2 Модуль генерации текста

5.2.1 Данные

Формирование обучающего корпуса для аватара требует сбора и обработки значительных объёмов данных, характеризующих речевое поведение человека. Корпус включает три основных типа данных: текст, аудио и видео.

Каждый указанный формат поддерживается интерфейсом сервиса с возможностью загрузки популярных расширений файлов. Эти файлы преобразуются в исходные тексты, которые составляют первый тип данных в обучающем корпусе.

Вторым важным источником данных являются социальные сети, представляющие собой естественный источник коммуникации человека в современном мире. Данные социальных сетей имеют структуру вопрос-ответ и требуют персонального согласия пользователя на обработку. Реализована интеграция с мессенджером Telegram для автоматического сбора текстовых, аудио и видеосообщений из выбранных пользователем диалогов.

Итого, обучающий корпус включает исходные тексты, а также диалоговые сообщения из социальных сетей.

После получения данных в пайплайн обучения формируется структурированный датасет для дообучения модели генерации. Видеофайлы конвертируются в аудиофайлы формата WAV с помощью библиотеки moviepy. Затем начинается этап распознавания речи с помощью модели Whisper, аналогичный для полученных аудиофайлов, позволяющий получить текстовую транскрипцию с хорошей точностью. На следующем шаге преобразованные и полученные текстовые файлы проходят стадию предобработки, в процессе которой удаляются лишние символы, технические вставки. Кроме того, слишком длинные предложения разбиваются на короткие, разбивая в основном на концах предложения, чтобы сохранить читаемость и связность. В завершение все данные объединяются в единый текстовый корпус, готовый для запуска обучения.

5.2.2 Выбор базовой модели

Выбор языковой модели для задачи генерации текста в речевом стиле конкретного человека основывался на необходимости запуска обучения и генерации на бесплатных вычислительных ресурсах Kaggle, обеспечивая лучший результат в рамках данных ограничений. Преимуществом использования платформы Kaggle для реализации пайплайнов является предоставление доступа к GPU ресурсам (NVIDIA Tesla P100 или T4), которых, несмотря на ограниченность, вполне достаточно для проведения экспериментов в научном проекте. Также немаловажным фактором в пользу использования данного облачного сервиса стало наличие официального API, который позволяет интегрировать использование вычислительных ресурсов в процесс обучения и генерации. На основании анализа научно-технической литературы и сравнительных оценок LLM было принято решение использовать модель из семейства Saiga. Основным критерием выбора послужили высокие результаты в бенчмарке PingPong, ориентированном на оценку в ролевых диалогах, одним из ключевых критериев которых является способность модели сохранять заданную роль. Наличие в обучающих корпусах этой серии данных, которые содержат разнообразные диалоговые и ролевые сценарии, что дополнительно подтверждает пригодность этих моделей для выбранной задачи. Кроме того, линейка моделей Saiga включает в себя версии различной сложности и размеров, что позволило подобрать оптимальный вариант. Модель saiga_mistral_7b_lora соответствует имеющимся ограничениям по вычислительным ресурсам, обладая хорошим уровнем качества генерации текста. В рамках ограниченных ресурсов ее применение возможно благодаря технологии QLoRA, при которой модель заранее квантуется до 4-битного представления и фиксирует неизменными часть слоев, что позволяет значительно сократить объем занимаемой памяти при незначительной потере точности. Поэтому ее можно использовать на видеокартах с объемом памяти 16 ГБ, которые доступны на платформе Kaggle. Также в описании модели на платформе Hugging Face представлен сравнительный тест, где saiga_mistral_7b_lora показала себя лучше, чем более крупная модель saiga2_13b_lora, основанная на архитектуре LLaMA 2, где при teste 243 ответа были признаны лучшими, против 31 у последней и при 141 ничейном результате. При этом в серии мо-

делей Saiga уступает более новой модели, основанной на архитектуре LLaMA 3, которая требует больших вычислительных ресурсов.

5.2.3 Метрики

На этапе выбора базовой модели для оценки качества генерации текста в речевом стиле конкретного человека главным элементом выступала *human evaluation* (человеческая экспертиза). Тем не менее учитывались следующие автоматические метрики:

- Perplexity — стандартная метрика, оценивающая степень «неопределённости» модели в предсказании следующего слова;
- BLEU (Bilingual Evaluation Understudy) — оценивает сходство сгенерированного текста с эталонными образцами;
- ROUGE (Recall-Oriented Understudy for Gisting Evaluation) — оценивает качество по наличию ключевых слов и выражений в сгенерированном тексте;

Однако численные метрики носили вспомогательный характер, а основное внимание уделялось качественной оценке результатов: насколько текст реалистичен, естественен, и соответствует интонациям и стилевым особенностям пользователя.

5.2.4 Токенизация

Для токенизации текста используется алгоритм Byte-Pair Encoding (BPE), который позволяет эффективно работать с большим словарём слов, сохраняя баланс между объёмом данных и скоростью обработки. Использование BPE помогает эффективно представлять редкие слова и специализированные термины, что особенно важно для персонализированной генерации текста.

5.2.5 Обучение

Процесс дообучения модели реализован в рамках чистой архитектуры, что обеспечило высокую модульность и отделение бизнес-логики от технических деталей. Все ключевые компоненты пайплайна — загрузка данных,

проверка ресурсов, запуск обучения и сохранение результатов — организованы в виде изолированных модулей с интерфейсами. Такой подход позволяет быстро адаптировать архитектуру под новые требования в условиях стремительного развития LLM. Дообучение может запускаться как на локальной машине, так и на бесплатных вычислительных ресурсах Kaggle. Выбор режима настраивается в конфигурационном файле. По умолчанию создается и запускается Kaggle-ноутбук с предварительно загруженным датасетом. Изменением одной переменной можно переключиться на локальное выполнение, что особенно полезно в условиях ограниченного доступа к аппаратным ресурсам. Запуск пайплайна осуществляется в отдельном процессе, что позволяет оставлять сервис в активном состоянии. Однако новые запросы на обучение игнорируются, пока работает хотя бы одна задача обучения. Конечная модель сохраняется в S3-хранилище, откуда потом используется для пайплайна генерации текста.

5.2.6 Генерация текста

Генерация текста построена по аналогичной модульной логике: в зависимости от настроек может использоваться локальный или удаленный режим. Ранее дообученная модель загружается из облачного хранилища для генерации ответа. Благодаря изолированной структуре, интеграция новой модели для генерации требует минимальных изменений — достаточно заменить соответствующий компонент, не затрагивая остальную часть пайплайна. Этот подход сохраняет гибкость системы и упрощает ее масштабирование при работе с различными моделями и окружениями.

5.3 Модуль генерации звука

Для синтеза речи используется короткий аудиофрагмент продолжительностью порядка 15 секунд, достаточный для точной имитации тембра и артикуляции голоса пользователя. Прикладная направленность задача звукового модуля задаёт акцент на практической пригодности решения и рациональном использовании вычислительных ресурсов, без разработки принципиально новых архитектур.

5.3.1 Выбор базовой модели

При выборе модели для синтеза речи ключевым критерием была способность точно воспроизводить голос пользователя на основе короткого аудиофрагмента. В процессе исследования различных решений, доступных на платформах GitHub и Hugging Face, было рассмотрено несколько подходов.

Одним из первых протестированных решений стала реализация **Real-Time Voice Cloning** от CorentinJ [23]. Эта модель демонстрирует хорошие результаты в задачах клонирования голоса и может работать в реальном времени. Однако, как отмечает сам автор, существуют более современные и качественные решения в области синтеза речи. В частности, он рекомендует обратить внимание на проекты, представленные на **paperswithcode**, а также на такие решения, как **CoquiTTS** [24] и **MetaVoice-1B** [25].

Следует отметить, что, несмотря на то, что проект **CoquiTTS** более не поддерживается его первоначальной командой разработчиков, он продолжает развиваться силами сообщества и остаётся актуальным благодаря регулярным обновлениям и исправлениям. В ходе тестирования CoquiTTS продемонстрировал стабильную работу, хорошую адаптацию к русскому языку и возможность кастомизации тембра речи, что сделало его оптимальным выбором в качестве основной модели синтеза речи в рамках проекта.

Параллельно с этим была предпринята попытка интеграции более актуальной архитектуры **LLASA** (LLaMA-based Speech Synthesis) [26]. Эти модели представляют собой современные решения в области синтеза речи, основанные на архитектуре LLaMA и интегрированные с кодеком XCodec2. Особенностью моделей LLASA является их способность к высококачественному синтезу речи и поддержка многозадачного обучения.

Семейство LLASA включает модели с различным количеством параметров: 1, 3 и 8 миллиардов. Однако полное переобучение даже минимальной версии **HKUSTAudio/Llasa-1B** требует значительных вычислительных ресурсов и большого объёма обучающих данных, что выходит за рамки доступных возможностей. Поэтому было принято решение использовать предобученную версию **Llasa-1B-Multilingual**, обладающую следующими преимуществами:

- Высокая естественность синтезируемой речи «из коробки» и унифицированная BPE-токенизация без необходимости в G2P-преобразовании.

- Компактный объём (около 1 миллиарда параметров), позволяющий проводить тонкую настройку модели на одной GPU с объёмом памяти 24 GB.

Основным недостатком выбранной модели является отсутствие русского корпуса в изначальной предобученной выборке, что требует дополнительного дообучения на соответствующих данных.

5.3.2 Подготовка данных

Для обучения и валидации использован открытый корпус **Russian Open Speech To Text**. Из него сформированы тематические подкорпуса суммарным объёмом около 460 GB WAV, охватывающие бытовую речь, подкасты, лекции, радиопередачи и ролики YouTube. Подготовка аудио-текстовых пар включала четыре этапа. Сначала исходные записи проходили базовую процедуру нормализацию: обрезку тишины, выравнивание уровня громкости и преобразование частоты дискретизации к 16 кГц. Затем текст каждой реплики токенизировался встроенной ВРЕ-моделью LLaSA, обеспечивая единый словарь для всех языков. Далее аудиодорожки дискретизировались в компактные токены кодеком XCodec2. Наконец, пары «текст–звук» сохранялись в формате **Hugging Face Datasets**, что позволяло без дополнительной конвертации использовать их в пайплайне дообучения и сохранить больше вычислительных мощностей для переобучения модели, а также сохранить целостность сформированных датасетов.

5.3.3 Гиперпараметры дообучения модели

Для успешного переноса модели на русский язык важное значение имеет выбор гиперпараметров обучения. Корректная настройка параметров оптимизации, регуляризации и аппаратного обеспечения обеспечивает достижение стабильной сходимости и приемлемого качества генерации. В таблице 6 приведены ключевые гиперпараметры, использованные в ходе финального цикла дообучения модели.

Таблица 5 — Подкорпуса Russian Open Speech To Text, отобранные для дообучения

Префикс	Объём, GB
asr_calls_2_val	2
buriy_audiobooks_2_val	1
public_youtube700_val	2
public_lecture_1	0.7
public_series_1	1.9
asr_public_stories_1	4.1
asr_public_stories_2	9
asr_public_phone_calls_1	22.7
public_youtube1120_hq	31
asr_public_phone_calls_2	66
public_youtube700	75
tts_russian_addresses_rhvoice_4voices	80.9
radio_2	154
public_youtube1120	237

Таблица 6 — Ключевые гиперпараметры дообучения

Параметр	Значение
Модель	HKUSTAudio/Llasa-1B-Multilingual
Тип данных	bfloat16, Flash-Attention 2
Оптимизатор	8-bit Adam (bitsandbytes)
Learning Rate	$5 \cdot 10^{-5}$ (cosine schedule)
Warmup Ratio	0.03
Weight Decay	0.01
Batch Size	16 аудио-сегментов
Эпох	3
Gradient Acc.	1
Аппаратная база	одна NVIDIA A100 40 GB

5.3.4 Процесс дообучения и отслеживание параметров

Эффективное управление процессом обучения требует постоянного мониторинга показателей, таких как функция потерь и другие вспомогатель-

ные метрики. Для обеспечения прозрачности экспериментов, удобства визуализации динамики и последующего анализа была выбрана платформа Weights & Biases. Данная платформа позволяет фиксировать следующие параметры и характеристики обучения:

- графики функции потерь на обучении и валидации;
- изменения значений MOS после завершения дообучения;
- контрольные аудио-сэмплы, позволяющие оценить промежуточные результаты;
- хэш коммита и конфигурацию гиперпараметров для воспроизведимости экспериментов.

На рисунках 9 и 10 представлены примеры визуализации основных параметров, т.е. функции потерь в данном случае, полученные при дообучении модели.

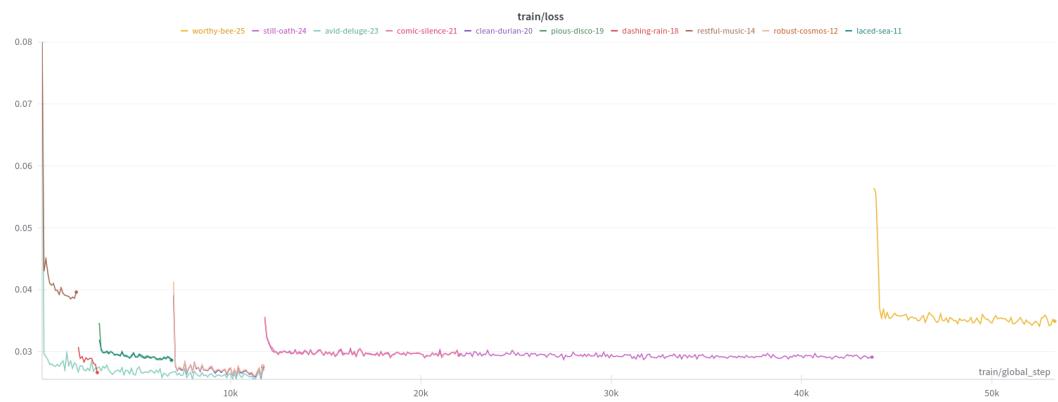


Рисунок 9 — Динамика функции потерь на обучении

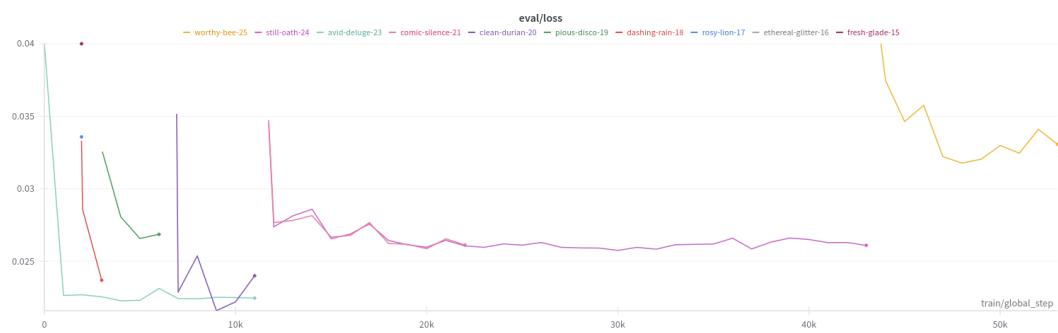


Рисунок 10 — Динамика функции потерь на валидации

5.3.5 Оценка качества генерации и используемые метрики

Итоговая оценка качества синтезированной речи представляет собой важнейший этап, подтверждающий практическую пригодность разработанной модели. Особое значение в этом контексте имеет субъективное восприятие конечного пользователя — естественность, эмоциональность и узнаваемость голоса. Для финальной проверки качества планировалось использовать метрику **Mean Opinion Score (MOS)**, которая широко применяется в задачах оценки качества синтетической речи.

Методика предполагала генерацию набора из 20 «слепых» аудио-сэмплов на фиксированном контрольном тексте, с последующим прослушиванием и оценкой каждого сэмпла группой из пяти независимых слушателей по шкале от 1 до 5. На основе этих оценок рассчитывались среднее значение и дисперсия MOS, что позволило бы количественно зафиксировать степень естественности и узнаваемости полученной речи.

Однако на практике полная реализация данного подхода оказалась невозможной из-за ограниченных вычислительных ресурсов. Провести полноценное обучение модели в запланированном объеме не удалось — была реализована лишь часть обучающих итераций, в результате чего качество сгенерированной речи оставалось далеко от ожидаемого. В таких условиях применение MOS стало неинформативным: оценки слушателей варьировались случайным образом, не отражая объективных различий в качестве, а результирующие значения средней оценки не позволяли сделать достоверные выводы. Таким образом, при текущем уровне реализации метрика MOS оказалась неприменимой и не может использоваться для формальной оценки итогового качества.

В связи с вышеописанными ограничениями, для получения качественных аудио-сэмплов было принято решение использовать готовую модель из библиотеки **Coqui TTS**.

6 Программная реализация

6.1 Веб-интерфейс

Веб-интерфейс является важнейшим элементом пользовательского взаимодействия с системой, обеспечивая доступ ко всему спектру функциональных возможностей приложения Bishop. В данной главе будет рассмотрена структура интерфейса, описаны отдельные страницы и реализуемые ими задачи, а также представлен детальный анализ технических решений, используемых при разработке.

6.1.1 Гостевая страница

Первая страница веб-интерфейса, доступная пользователю, представляет собой гостевую посадочную страницу [12](#) (landing page). Она разработана с целью ознакомления пользователей с проектом Bishop и не требует предварительной авторизации.

Страница выполнена в минималистичном стиле, что позволяет сфокусировать внимание пользователя на основном содержании. В верхней части расположено название проекта «Bishop Ai Application», а справа в шапке страницы находятся две активные ссылки: «Login» и «Signup», позволяющие перейти к авторизации или регистрации соответственно.

Центральная часть страницы содержит краткое приветствие пользователя («Welcome to BishopApp») и детальное описание основной идеи и функционала приложения.

Нижний колонтитул страницы содержит информацию об авторских правах, что завершает композицию интерфейса и подчеркивает официальность ресурса.

6.1.2 Идентификация, аутентификация и авторизация

Сервис Bishop предусматривает возможность идентификации и авторизации пользователей через формы входа [13](#) («Login») и регистрации [14](#) («Signup»).

Страница входа содержит поля для ввода электронной почты и пароля. После отправки данных производится запрос к backend API, который проверяет подлинность введенных данных. В случае успешного входа система переадресует пользователя на личную страницу, при этом отображая уведомление с текстом «*login is successful!*». Если данные введены неверно, интерфейс отображает сообщение об ошибке («*Incorrect email or password*»), что существенно улучшает пользовательский опыт, четко информируя о причинах невозможности авторизации.

Регистрация

Форма регистрации аналогична форме входа, однако требует от пользователя дополнительной информации в виде полного имени. Введенные данные валидируются как на уровне фронтенда (проверка формата email и минимальной длины пароля), так и на стороне бэкенда (проверка уникальности email). Ошибки ввода или проверки также сопровождаются информативными уведомлениями:

- *value is not a valid email address* — если указан некорректный адрес электронной почты.
- *String should have at least 8 characters* — при слишком коротком пароле.
- *The user with this email already exists in the system* — если email уже зарегистрирован.

Техническая реализация и безопасность данных

Для реализации процесса авторизации используется стандарт JWT. После успешной аутентификации токен сохраняется в сессии пользователя и добавляется в заголовки каждого запроса к backend API, что обеспечивает надежную защиту и конфиденциальность данных пользователя:

```
def get_auth_headers(sess):  
    jwt_keys = ["token_type", "access_token"]  
    for key in jwt_keys:  
        if key not in sess:  
            return None
```

```

    token_type = sess["token_type"]
    token = sess["access_token"]
    headers = {"Authorization": f"{token_type} {token}"}

    return headers

```

Дополнительным слоем безопасности является проверка действительности JWT-токена при каждой попытке доступа к защищённым страницам посредством специального промежуточного обработчика jwt_before:

```

async def jwt_before(req, sess):
    jwt_keys = ["token_type", "access_token"]
    for key in jwt_keys:
        if key not in sess:
            return Redirect("/")

    auth_hdrs = {"Authorization": f"{sess['token_type']} {sess['access_token']}"} 

    async with httpx.AsyncClient(headers=auth_hdrs) as cli:
        res = await cli.post(BACKEND_URL + "/login/test-
                               token")

    if res.status_code != 200:
        return Redirect("/")

```

Данные подходы позволяют обеспечить высокую степень защиты персональных данных и контролируемость доступа к ресурсам приложения.

6.1.3 Основная страница приложения

После успешной авторизации пользователь перенаправляется на главную страницу **15**, являющуюся единственной страницей в архитектуре SPA. Все дальнейшие действия пользователя инициируют частичное обновление элементов DOM, не приводя к полной перезагрузке страницы. Такой подход значительно повышает отзывчивость и удобство использования приложения.

Главная страница состоит из следующих элементов:

- Информация о пользователе
- Функциональные элементы для менеджмента аватаров

В верхней части страницы располагается блок с персональными данными авторизованного пользователя, включающий его имя и адрес электронной почты, а также кнопку выхода («Sign Out») для завершения текущей сессии.

Ниже блока с информацией пользователя находится основной функционал, предназначенный для работы с цифровыми аватарами:

- **Создание аватара:** Пользователь вводит имя нового аватара в соответствующее текстовое поле и нажимает кнопку Create Avatar.

После успешного создания аватара выводится уведомление (Avatar created successfully!) в виде toast-сообщения.

- **Список аватаров:** Все созданные пользователем аватары отображаются в виде кнопок, расположенных в специальном блоке. По клику на аватар открываются дополнительные элементы управления (редактирование, удаление, чат и обучение), при этом не происходит полной перезагрузки страницы.

Технически реализация создания аватара представлена следующим образом:

```
@dataclass
class AvatarCreateInfo:
    name: str

@rt("/avatar-create")
async def avatar_create(avatar_info: AvatarCreateInfo, sess):
    :
    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as cli:
        res = await cli.post(
            f"{BACKEND_URL}/avatars/",
            json=asdict(avatar_info)
        )

        if res.status_code == 200:
            add_toast(sess, "Avatar created successfully!",
                      "info")
            return Redirect("/index")
```

Использование асинхронных запросов (httpx.AsyncClient) и JWT-токенов в заголовках запроса гарантирует безопасность операций и соответствие высоким стандартам защиты пользовательских данных.

6.1.4 Страница работы с аватаром

Страница работы с конкретным аватаром [16](#) представляет собой ключевую часть веб-интерфейса Bishop, позволяющую получить доступ ко всему основному функционалу, связанному с взаимодействием и управлением аватаром. Эта страница является динамически загружаемым компонентом основного SPA-приложения и отображается при выборе конкретного аватара из списка.

На странице представлена карточка с именем аватара и набор кнопок, каждая из которых предоставляет доступ к определенному функционалу:

- **Train** — переход к интерфейсу загрузки новых материалов для обучения модели аватара и управления процессом её обучения.
- **Chat** — переход к разделу общения с аватаром, где пользователь может взаимодействовать с виртуальной персоной в режиме диалога.
- **Change Name** — позволяет изменить имя выбранного аватара через простой и удобный интерфейс.
- **Delete** — реализует удаление аватара, сопровождаемое подтверждением действия со стороны пользователя, что предотвращает случайные ошибки.
- **To avatars list** — возвращает пользователя обратно к общему списку аватаров.

CRUD-функционал реализован асинхронными вызовами к backend API, что повышает скорость взаимодействия и удобство использования приложения.

Пример кода реализации переименования аватара:

```
@rt("/avatar/{avatar_id}/change_name", methods=["POST"])
async def change_name(avatar_id: str, name: str, sess):
    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as client:
        response = await client.put(
            f"{BACKEND_URL}/avatars/{avatar_id}",
            json={"name": name}
        )
    if response.status_code == 200:
        add_toast(sess, "Avatar renamed successfully!", "success")
    else:
```

```

        add_toast(sess, "Failed to rename avatar.", "error")

    return Redirect(f"/index")

```

Пример кода реализации удаления аватара:

```

@rt("/avatar/{avatar_id}/delete", methods=["POST"])
async def delete_avatar(avatar_id: str, sess):
    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as client:
        response = await client.delete(
            f"{BACKEND_URL}/avatars/{avatar_id}"
        )

    if response.status_code == 200:
        add_toast(sess, "Avatar deleted!", "success")
    else:
        add_toast(sess, "Failed to delete avatar.", "error")

    return Redirect("/index")

```

Взаимодействие с backend происходит через запросы с авторизационными JWT-токенами, что гарантирует сохранность и защищенность данных пользователя.

6.1.5 Интерфейс общения с аватаром

Один из центральных элементов веб-интерфейса — это возможность вести живой диалог с цифровым аватаром. Для реализации этого функционала предусмотрен отдельный интерфейс чата 18. Пользователь может создать несколько отдельных чатов для каждого аватара на отдельно странице для менеджмента чатов 17, например, на различные темы. Каждый чат имеет своё название, задаваемое пользователем при его создании.

После выбора чата пользователю открывается интерфейс переписки, состоящий из:

- поля для ввода текста сообщения;
- кнопки отправки сообщения (Send);
- кнопки возврата к списку чатов (Back to Chats).

Особенностью взаимодействия является то, что аватар не только генерирует текстовые ответы, но и озвучивает их. После отправки сообщения

пользователь видит свой вопрос и автоматически отображающееся сообщение о том, что аватар формирует ответ (Avatar is typing...).

Как только модель сформирует ответ, сообщение заменяется текстовым ответом аватара и аудиопроигрывателем, позволяющим прослушать ответ. Это обеспечивает эффект живого диалога и существенно улучшает восприятие взаимодействия.

Для реализации подобного поведения использованы асинхронные запросы с механизмом polling-запросов (периодические запросы к backend API для получения статуса готовности ответа):

```
@rt("/avatar/{avatar_id}/chat/{chat_id}/poll_response/{  
    rsp_msg_id}/")  
async def poll_response(avatar_id: str, chat_id: str,  
    rsp_msg_id: str, sess):  
    async with httpx.AsyncClient(headers=get_auth_headers(  
        sess)) as client:  
        for _ in range(3):  
            await asyncio.sleep(2)  
  
            msg_url = (  
                f"{BACKEND_URL}/avatars/{avatar_id}/chat/{  
                    chat_id}"  
                f"/msgs/{rsp_msg_id}/response/"  
            )  
            res = await client.get(msg_url)  
            if res.status_code != 200:  
                continue  
  
            msg = res.json()  
            audio_url = (  
                f"{BACKEND_URL}/avatars/{avatar_id}/chat/{  
                    chat_id}"  
                f"/msgs/{rsp_msg_id}/response/dub/"  
            )  
            audio_res = await client.get(audio_url)  
  
            if audio_res.status_code == 200:  
                b64 = base64.b64encode(audio_res.content).  
                    decode("ascii")  
            return Div(  
                P(f"Avatar: {msg['text']}"),  
                Audio(
```

```

        controls=True ,
        autoplay=False ,
        preload="auto" ,
        src=f"data:audio/x-wav;base64,{b64}"
    ) ,
    cls="chat-message-bot"
)

```

Таким образом, интерфейс обеспечивает интерактивное взаимодействие с аватаром, совмещающая текстовую переписку с живым голосовым сопровождением, что делает диалог максимально близким к естественной коммуникации.

6.1.6 Интерфейс обучения аватара

Также ключевым элементом функционала является раздел обучения аватара **19**. Пользователю доступен расширенный набор действий, позволяющий совершенствовать аватар посредством добавления новых обучающих материалов и управления процессом обучения.

На странице обучения пользователь может загрузить материалы двух основных типов:

- **Аудио для синтеза голоса:** Короткие аудио-файлы формата .wav, используемые для создания реалистичной модели голоса аватара.
- **Материалы для извлечения текста:** Аудио, видео или текстовые документы, из которых будет извлечён текстовый контент для обучения языковой модели аватара.

Пользовательский интерфейс загрузки материалов представлен формой с возможностью выбора типа материалов и загрузки нескольких файлов одновременно:

- Выпадающий список выбора типа загружаемых данных (Extract text или Voice synthesis).
- Поле выбора файлов и кнопка загрузки.

После загрузки файлы отображаются в списке текущих материалов для обучения, где для каждого файла указывается тип и ссылка для скачивания.

Пример реализации загрузки материалов на уровне frontend:

```
@rt("/avatar/{avatar_id}/train", methods=["POST"])
```

```

async def proxy_upload(avatar_id: str, request: Request,
sess):
    form = await request.form()
    uploaded_files = form.getlist("files")
    type_value = form.get("type")

    file_data = []
    for file in uploaded_files:
        content = await file.read()
        file_data.append((
            "file",
            (file.filename, content, file.content_type)
        ))

    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as client:
        res = await client.post(
            f"{BACKEND_URL}/avatars/{avatar_id}/train/",
            data={"type": type_value},
            files=file_data
        )

    if res.status_code == 200:
        add_toast(sess, "File uploaded successfully!", "info")
    else:
        add_toast(sess, "File upload failed!", "error")

    return await avatar_train_widget(avatar_id, sess)

```

Пользователь может непосредственно управлять процессом обучения, используя две основные кнопки:

- **Start Training:** Инициирует процесс обучения модели аватара с использованием всех загруженных материалов. После старта обучения статус модели изменяется на `training`.
- **Stop Training:** Позволяет принудительно остановить процесс обучения. После остановки статус модели меняется обратно на `available`, и пользователю вновь предоставляется возможность запуска нового обучения.

Реализация управления статусом обучения:

```
@rt("/avatar/{avatar_id}/train/start", methods=["POST"])
```

```

async def avatar_train_start(avatar_id: str, sess):
    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as client:
        await client.post(
            f"{BACKEND_URL}/avatars/{avatar_id}/train/start"
        )

    return await avatar_train_widget(avatar_id, sess)

@rt("/avatar/{avatar_id}/train/stop", methods=["POST"])
async def avatar_train_stop(avatar_id: str, sess):
    async with httpx.AsyncClient(headers=get_auth_headers(
        sess)) as client:
        await client.post(
            f"{BACKEND_URL}/avatars/{avatar_id}/train/stop"
        )

    return await avatar_train_widget(avatar_id, sess)

```

Интерфейс динамически отражает текущее состояние аватара, показывая статус модели (`training`, `available`), список загруженных материалов и соответствующие уведомления, обеспечивая высокий уровень прозрачности и удобства для пользователя.

Таким образом, Bishop предоставляет удобный и интуитивно понятный интерфейс для полноценного управления обучением аватаров, поддерживая оперативный контроль за текущим состоянием и эффективное взаимодействие с системой.

6.2 Backend-сервис

Backend-сервис является центральным узлом системы, обеспечивающим взаимодействие между всеми ключевыми компонентами:

- Веб-интерфейс
- База данных
- ML-сервис
- S3-хранилище

В следующих разделах мы подробно разберем ключевые особенности имплементации, связанные с взаимодействием backend-сервиса с каждым конкретным узлом системы.

6.2.1 Взаимодействие с веб-интерфейсом

Ключевой библиотекой, которая используется в backend-сервисе для работы с веб-интерфейсом является FastApi, которая позволяет быстро и эффективно описывать структурированный API, поддерживает асинхронный режим работы, а так же автоматически генерирует документацию через OpenApi(Swagger).

Группировка HTTP методов и соответствующих модулей

При проектировании API использовался устоявшийся в индустрии метод, согласно которому маршруты разделялись по HTTP-методам. Такой метод позволяет упростить и стандартизировать взаимодействие между backend-сервисом и веб-интерфейсом. Так же это может упростить и сделать API понятным для сторонних разработчиков.

Для удобства и простоты поддержки маршруты дополнительно были сгруппированы в отдельные модули:

- users - управление пользователями
- train – загрузка данных для обучения
- msgs – управление сообщениями
- chats – управление чатами
- avatars – управление аватарами

Ниже приведены примеры маршрутов, используемые для создания ресурсов и их краткое описание:

- POST /api/v1/avatars – тело запроса состоит из json-объекта, единственным полем которого является строка. Стока передает в систему имя аватара, которого хочет создать пользователь.

Структура json-объекта:

```
{  
    "name": "Avatar's name"  
}
```

- POST /api/v1/avatars/{avatar_id}/chat – тело запроса вновь состоит из строки, которая несет в себе имя для чата, который хочет создать пользователь. В отличие от первого примера URI содержит в себе параметр {avatar_id}, отвечающий за идентификацию конкретного аватара пользователя.

Структура json-объекта:

```
{
    "title": "Chat's title"
}
```

- POST /api/v1/avatars/{avatar_id}/chat/{chat_id}/msgs – несет в себе полезную нагрузку в виде строки, которую набрал пользователь при отправке очередного сообщения. URI содержит {avatar_id} и {chat_id}.

Структура json-объекта:

```
{
    "message": "User's message"
}
```

- POST /api/v1/users – содержит json объект с полями email, isSuperuser, isActive, fullName, password. Все эти поля используются для создания записи о пользователе в базе данных.

Структура json-объекта:

```
{
    "email": "user@example.com",
    "isSuperuser": false,
    "isActive": true,
    "fullName": "User's fullname",
    "password": "secret"
}
```

- POST /api/v1/avatars/{avatar_id}/train – в теле запроса несет бинарное представление файла пользователя и его метаданные такие как имя файла и формат. URI содержит {avatar_id}.

Пример описания обработчика из модуля users

Ниже приведен пример исходного кода функции, которая занимается обработкой запросов на создание нового пользователя:

```
@router.post(
    "/",
    dependencies=[Depends(get_current_active_superuser)],
    response_model=UserPublic
)
async def create_user(
    *,
    session: SessionDep,
    user_in: UserCreate
) -> User:
    """
    Create new user.
    """

    user = await user_repository.get_user_by_email(session=session,
                                                    email=user_in.email)
    if user:
        raise HTTPException(
            status_code=400,
            detail="The user with this email already exists
                   in the system.",
        )

    user = await user_repository.create_user(session=session,
                                              user_create=user_in)

    if settings.emails_enabled and user_in.email:
        email_data = generate_new_account_email(
            email_to=user_in.email, username=user_in.email,
            password=user_in.password
        )
        send_email(
            email_to=user_in.email,
            subject=email_data.subject,
            html_content=email_data.html_content,
        )
    return user
```

Представленный код демонстрирует общий подход к написанию обработчиков в backend-сервисе.

Ключевые архитектурные элементы:

- Pydantic-модели - обеспечивают строгую типизацию и автоматическую валидацию
- Система зависимостей (Depends) - инкапсулируют повторяющуюся логику
- Разделение ответственности - для работы с БД и другими сущностями обработчики вызывают функции из отдельных модулей
- Декораторы - стандартизируют HTTP-метод, путь и формат ответа
- Обработка ошибок - единый стиль обработки через HTTPException

6.2.2 Взаимодействие с базой данных

Основные сведения о реализации

В backend сервисе для работы с базой данных используется библиотека SQLAlchemy. В исходном коде сервиса активно используются следующие преимущества этой библиотеки:

- Работа с записями БД как с объектами Python
- Встроенная валидация данных на основе аннотаций типов
- Поддержка асинхронных операций.

Все сущности, которые семантически относятся к записям в базе данных или содержат в себе информацию, которая должна попасть в базу данных были описаны с помощью специальных классов, которые в данном контексте принято называть моделями. Эти модели можно разделить на два типа:

- Модели БД – описывают структуру таблиц в базе данных
- Pydantic-модели (прокси-классы), которые используются для:
 - Валидации входящих json-данных
 - Сериализации ответов API
 - Предварительной обработки данных

Пример реализации

Рассмотрим на примере сущности User организацию моделей для обработки запросов и работы с БД:

```
class UserBase(SQLModel):
    email: EmailStr = Field(unique=True, index=True,
                           max_length=255)
    is_active: bool = True
    is_superuser: bool = False
    full_name: str | None = Field(default=None, max_length
                                  =255)

class UserCreate(UserBase):
    password: str = Field(min_length=8, max_length=40)

class User(UserBase, table=True):
    __tablename__ = "user"
    id: uuid.UUID = Field(default_factory=uuid.uuid4,
                          primary_key=True)
    hashed_password: str
    avatars: list["Avatar"] = Relationship(
        back_populates="user", cascade_delete=True)
```

Модели организованы по принципу наследования:

- UserBase - базовый класс, содержащий общие поля, которые присутствуют во всех производных моделях. Служит для избежания дублирования кода.
- UserCreate - модель для валидации входящих данных при создании пользователя. Если json-объект в теле запроса не соответствует этой модели, сервер автоматически возвращает ошибку 422 Unprocessable Entity с детальным описанием проблем валидации, что избавляет разработчика от необходимости писать рутинный код для проверки входных данных.
- User - основная модель, которая описывает структуру таблицы БД, наследует все функциональные возможности SQLAlchemy и используется для непосредственных запросов к БД

Рассмотрим преимущества такой структуры кода на примере, когда API получает запрос на создание нового пользователя и вызывает функцию, которая инкапсулирует в себе обращение к БД для создания новой записи:

```
async def create_user(*, session: AsyncSession, user_create:  
    UserCreate) -> User:  
    db_obj = User.model_validate(  
        user_create, update={  
            "hashed_password": get_password_hash(user_create.  
                password)})  
    session.add(db_obj)  
    await session.commit()  
    await session.refresh(db_obj)  
    return db_obj
```

В этом примере стоит обратить внимание на следующие пункты, которые раскрывают удобство описываемого подхода:

- Входящие данные автоматически валидируются моделью UserCreate
- Готовый объект User сохраняется в БД
- Происходит преобразование UserCreate → User с:
 - Автоматическим заполнением недостающих полей
 - Игнорированием нерелевантных данных

Структура модулей для работы с разными таблицами в БД

Для каждой сущности из базы данных (рис. 11) реализован отдельный модуль, содержащий:

- CRUD-операции – базовые создание, чтение, обновление и удаление записей
- Специфическую логику – например, загрузка бинарных данных для обучения в S3 хранилище, с последующим сохранением URL в базу данных

Такое архитектурное решение предоставляет ряд преимуществ:

- Четкое разделение ответственности – каждый модуль отвечает только за свою сущность
- Масштабируемость – добавление новых сущностей не ломает существующий код

- Упрощение поддержки – изменения в одной сущности не затрагивают другие

- Удобство тестирования – модули можно проверять изолированно

На примере модуля, отвечающего за операции с пользователями, рассмотрим стандартные CRUD-функции и их назначение:

- Получение списка пользователей

```
async def get_users(*, session: AsyncSession,
                    skip: int = 0, limit: int = 100) -> list[User]:
```

- Создание пользователя

```
async def create_user(*, session: AsyncSession,
                      user_create: UserCreate) -> User:
```

- Обновление пользователя

```
async def update_user(*, session: AsyncSession,
                      db_user: User, user_in: UserUpdate) -> Any:
```

- Поиск пользователя по email

```
async def get_user_by_email(*, session:
                            AsyncSession, email: str) -> User | None:
```

- Аутентификация пользователя

```
async def authenticate(*, session: AsyncSession,
                      email: str, password: str) -> User | None:
```

На данном примере можно отметить ряд особенностей, которые выполняются для всех подобных модулей:

- Типизация – все функции строго типизированы
- Единообразный подход в формировании сигнатур
- Соответствие названий функций семантике

6.2.3 Взаимодействие с ML-сервисом

Взаимодействие backend с ML-сервисом осуществляется через брокер сообщений Kafka. Для работы с брокером используется библиотека aiokafka.

Чтобы обеспечить удобство поддержки и разделение ответственности, логика работы с брокером была инкапсулирована в две отдельные сущности. Критерием разделения стало направление сообщений:

- KafkaMessageProducer — отвечает за формирование и отправку сообщений в ML-сервис

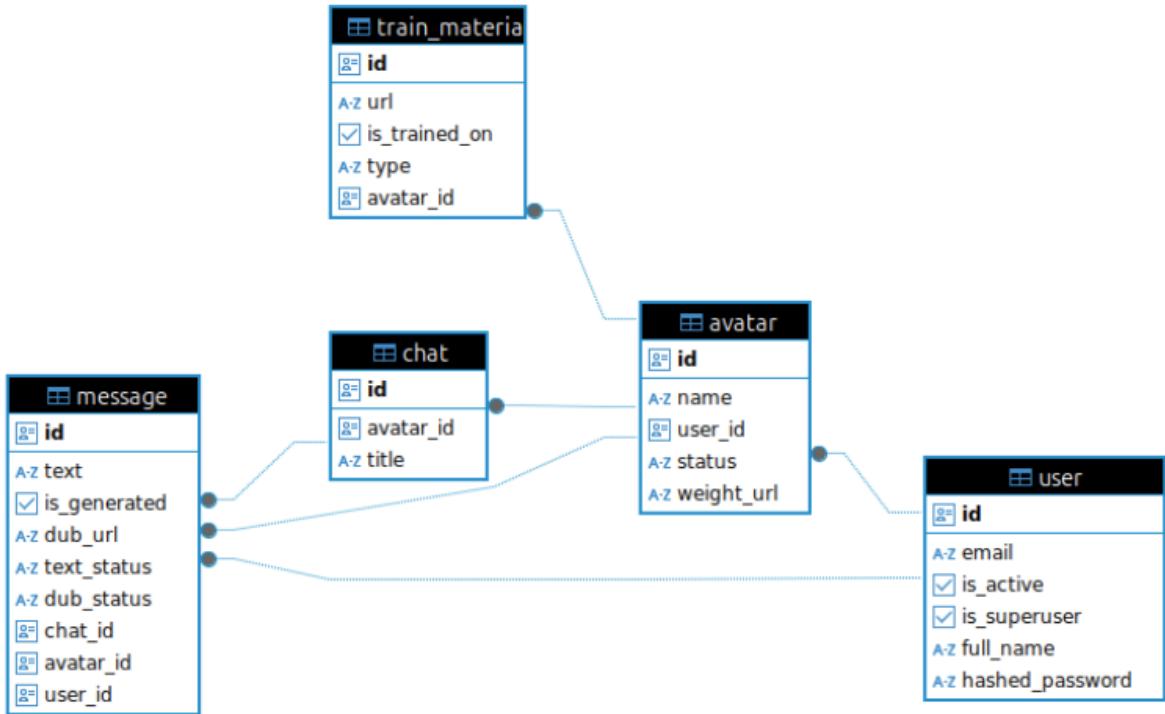


Рисунок 11 — Структура базы данных

- MessageManagerConsumer — обрабатывает входящие сообщения от ML-сервиса

Такое разделение позволяет упростить масштабирование, тестирование и модификацию каждого компонента в будущем, а также не накладывает ограничений на возможность параллельной разработки.

Формирование и передача запросов к ML-сервису

KafkaMessageProducer представляет собой клиентский модуль, отвечающий за отправку заранее сформированных сообщений в Kafka. Для каждого типа сообщения реализована отдельная функция, которая использует интерфейс клиента для передачи данных в брокер.

Типы сообщений:

- start_train — запускает обучение аватара
- stop_train — останавливает процесс обучения;
- inference_response — инициирует генерацию текстового ответа на сообщение пользователя;

- sound_inference — запускает синтез озвучки для сгенерированного текста.

1. Сообщение start_train

```
{
  "event": "train_start",
  "avatar_id": avatar_id,
  "train_materials": [
    {
      "id": material.id,
      "type": material.type,
      "url": material.url
    }
  ]
}
```

Описание полей:

- event - тип события
- avatar_id - уникальный идентификатор аватара
- train_materials - список объектов для обучения:
 - id - уникальный идентификатор материала
 - type - тип материала
 - url - путь в S3

2. Сообщение stop_train

```
{
  "event": "train_stop",
  "avatar_id": avatar_id
}
```

Описание полей:

- event - тип события
- avatar_id - уникальный идентификатор аватара

3. Сообщение inference_response

```
{
  "event": "inference_response",
  "message_id": message_id,
  "text": user_message
}
```

Описание полей:

- event - тип события
- message_id - уникальный идентификатор сообщения пользователя

- text - текст сообщения пользователя

3. Сообщение sound_inference

```
{
    "event": "sound_inference",
    "storage_url": storage_url,
    "base_voice_url": base_voice_url,
    "message_id": message_id,
    "text": gen_message,
}
```

Описание полей:

- event - тип события
- storage_url - путь в S3, где нужно будет сохранить результат
- base_voice_url - путь в S3, откуда брать голос для озвучки
- message_id - уникальный идентификатор сообщения, для которого требуется озвучка
- text - текст сообщения, которое нужно озвучить

Получение и обработка ответов от ML-сервиса

Как было сказано в начале раздела - для обработки сообщений от ML-сервиса был реализован отдельный класс, MessageManagerConsumer принимает и обрабатывает все входящие сообщения. В текущей архитектуре сервис поддерживает всего два типа сообщений:

- save_response - сохранение сгенерированного ответа
- save_response_dub - сохранение сгенерированной озвучки

Учитывая небольшое количество сообщений и отсутствие планов по расширению модели сообщений в направлении от ML-сервиса к backend-сервису, а также большую схожесть в процессе обработки, было решено инкапсулировать всю логику работы внутрь единого класса. Такой подход предоставил ряд преимуществ:

- Отсутствие избыточного разделения кода
- Единая точка отказа для данного направления
- Прозрачность потока данных

```
async with db_context() as session:
    message_id = task_data["message_id"]
```

```

        db_message = await message_repository.
            get_message_by_id(
                session=session,
                message_id=message_id
            )
        if db_message is None:
            logger.warning(f"Message with ID {message_id}
                            } not found.")
            return

        if event == "save_response":
            db_message.text = task_data.get(
                "generated_text", db_message.text
            )
            db_message.text_status = "ready"

        elif event == "save_response_dub":
            db_message.text = task_data.get(
                "generated_text", db_message.text
            )
            db_message.text_status = "ready"
            db_message.dub_url = task_data.get(
                "dub_url", db_message.dub_url
            )
            db_message.dub_status = "ready"

        session.add(db_message)
        await session.commit()
    
```

1. Обработка сообщения save_response

В этом сообщении присутствует 3 поля:

- event - тип события
- message_id - уникальный идентификатор сообщения
- generated_text - сгенерированный текст

По message_id производится поиск нужной записи в БД, после чего обновляется поле text_status, которое является идентификатором готовности ответа, и generated_text, которое отвечает за хранение сгенерированного ответа.

2. Обработка сообщения save_response_dub

В этом сообщении присутствует 4 поля:

- event - тип события

- message_id - уникальный идентификатор сообщения
- generated_text - сгенерированный текст
- dub_url - путь в S3, где хранится озвучка

Аналогично предыдущему пункту по message_id производится поиск нужной записи в БД, после чего обновляется поле text_status, generated_text и dub_url, а также dub_status

6.2.4 Взаимодействие с S3-хранилищем

Для работы с объектным хранилищем в проекте используется MinIO — высокопроизводительная S3-совместимая платформа для хранения данных. Взаимодействие реализовано через класс-клиент, который предоставляется библиотекой. Через него можно производить базовые операции над файлами:

- Загрузка
- Скачивание
- Удаление

Загрузка данных пользователя

Загрузка данных в объектное хранилище инкапсулирована в отдельную функцию:

```
async def upload_to_s3(
    file: UploadFile,
    user_id: uuid.UUID,
    avatar_id: uuid.UUID,
    type: str,
    session: SessionDep
) -> str:
    file_ext = file.filename.split('.')[-1]
    file_type = detect_file_type(file_ext)
    file_id = uuid.uuid4()
    object_name = f"users/{user_id}/avatars/{avatar_id}/{file_type}/{file_id}.{file_ext}"

    logger.info(f"Uploading file to MinIO: {object_name}")
    logger.info(f"File type detected: {file_type}")
    logger.info(f"File type requested: {type}")
    logger.info(f"File extension: {file_ext}")
```

```

        if type == TRAINIGN_MATERIAL_TYPE.voice_syntesis and
            file_ext != "wav":
            logger.error(
                f"Invalid file type for voice synthesis: {file_type}. Expected wav."
            )
            raise HTTPException(
                status_code=400,
                detail="Invalid file type for voice synthesis. "
                "Expected audio file with wav extension for
                better quality."
            )

    try:
        file_data = await file.read()
        file_size = len(file_data)

        kwargs = dict(
            bucket_name=settings.MINIO_BUCKET,
            object_name=object_name,
            data=BytesIO(file_data),
            length=file_size,
            content_type=file.content_type,
        )
        await run_in_threadpool(minio_client.put_object, **kwargs)
    except S3Error as exc:
        raise RuntimeError(f"Failed to upload to MinIO: {exc}")

    if type == TRAINIGN_MATERIAL_TYPE.voice_syntesis and
        file_ext == "wav":
        await avatar_repository.update_avatar_voice_url(
            session=session,
            avatar_id=avatar_id,
            voice_url=f"{
                settings.MINIO_URL}/{settings.MINIO_BUCKET}
                /{object_name}"
        )

    return f"{settings.MINIO_URL}/{settings.MINIO_BUCKET}/{object_name}"

```

Стоит обратить внимание, что все файлы для обучения сохраняются по этому пути:

users/{user_id}/avatars/{avatar_id}/{file_type}/{file_id}.{file_ext}

Преимущества такой организации файлов:

- Консистентность между хранилищем и API - пути в S3 зеркалируют структуру маршрутов
- Единая логика доступа - упрощает построение ментальной модели, что помогает увеличить скорость разработки
- Прозрачность взаимодействия - иерархия путей сохраняет отношения между сущностями
- Масштабируемость - позволяет легко добавлять новые уровни вложенности

6.3 Сервис генерации звука

Сервис реализован как отдельное приложение, функционирующее в режиме фонового процесса и обрабатывающее сообщения из системы обмена сообщениями Kafka. В зависимости от параметров задачи он способен использовать как основную нейросетевую модель синтеза речи (xtts_v2), так и альтернативную — LLASA. Хранение аудиофайлов реализовано через объектное хранилище MinIO.

6.3.1 Обработка событий брокера сообщений и инициализация очереди генерации

После старта приложение инициализирует Kafka-консюмера и подписывается на два ключевых топика: **sound_inference** (обработка задач озвучки) и **health_check** (мониторинг работоспособности). Эта инициализация происходит в точке входа:

```
consumer = KafkaMessageConsumer(  
    bootstrap_servers=settings.KAFKA_BROKER_URL,  
    topics=[  
        settings.KAFKA_HEALTH_CHECK_TOPIC,  
        settings.KAFKA_TOPIC_SOUND_INFERENCE,  
    ],  
    group_id=settings.KAFKA_GROUP_ID
```

```
)  
consumer.run()
```

Консюмер постоянно опрашивает брокер Kafka, проверяя наличие новых сообщений, и при получении валидной задачи вызывает функцию обработки:

```
def execute_task_pipeline(task_data):  
    curr_event = task_data["event"]  
  
    if curr_event == "sound_inference":  
        process_inference_task(task_data)
```

6.3.2 Этапы синтеза речи

При поступлении задачи с типом `sound_inference` запускается пайплайн, использующий Coqui TTS (модель `xtts_v2`). Модель загружается при первом обращении и используется для синтеза речи на основе текста и выбранного голосового профиля.

В зависимости от параметров задачи, может быть использован как кастомный голос, загружаемый из MinIO, так и предустановленный:

```
if base_voice:  
    tts.tts_to_file(  
        text=text,  
        language='ru',  
        speaker_wav=temp,  
        pipe_out=wrapper  
    )  
else:  
    tts.tts_to_file(  
        text=text,  
        language='ru',  
        speaker='Craig Gutsy',  
        pipe_out=wrapper  
    )
```

Полученный аудиофайл сохраняется в MinIO. Для этого используется обёртка над BytesIO и функция загрузки:

```
def save_dub_to_s3(dub_url: str, buffer: io.BytesIO) -> str:  
    buffer.seek(0)  
    minio_client.put_object(
```

```

        bucket_name=settings.MINIO_BUCKET ,
        object_name=dub_url ,
        data=buffer ,
        length=buffer.getbuffer().nbytes ,
        content_type='audio/wav'
    )
    return dub_url

```

Перед генерацией, если используется кастомный голос, он временно скачивается в локальное хранилище:

```

def download_minio_file_to_local(object_name: str, suffix:
    str = ".wav") -> str:
    response = minio_client.get_object(
        bucket_name=settings.MINIO_BUCKET ,
        object_name=object_name ,
    )
    with tempfile.NamedTemporaryFile(delete=False, suffix=
        suffix) as temp_file:
        temp_file.write(response.read())
    return temp_file.name

```

По завершении — временный файл удаляется:

```

def delete_local_file(file_path: str) -> None:
    if os.path.exists(file_path):
        os.remove(file_path)

```

После успешного завершения обработки, результат отправляется обратно в Kafka:

```

def send_update_message_state(producer, message_id,
    generated_text, dub_url):
    payload = {
        "event": "save_response_dub",
        "message_id": str(message_id),
        "dub_url": dub_url,
        "generated_text": generated_text,
    }
    producer.send(topic=settings.
        KAFKA_TOPIC_SAVE_RESPONSE_DUB, data=payload)

```

Альтернативный режим работы используется, если в задаче явно указан режим llasa. В этом случае применяется модель LLASA в связке с декодером Codec2. Генерация выполняется в несколько этапов:

- Формирование промпта с использованием специальных токенов;

- Токенизация текста с помощью LLASA;
- Генерация модели и извлечение токенов;
- Декодирование в аудиосигнал с помощью Codec2.

```

chat = [
{
    "role": "user",
    "content": "Convert the text to speech: <|
        TEXT_UNDERSTANDING_START |>...<|
        TEXT_UNDERSTANDING_END |>"
},
{
    "role": "assistant",
    "content": "<| SPEECH_GENERATION_START |>"
}
]
in_ids = tokenizer.apply_chat_template(chat, tokenize=True,
    return_tensors='pt')
outputs = model.generate(...)
```

6.3.3 Формат обратной связи через брокер сообщений

После завершения генерации, итоговое сообщение, отправляемое обратно в Kafka, имеет унифицированный JSON-формат:

```
{
    "event": "save_response_dub",
    "message_id": "e4aa6ae7-d6d6-4b1a-88fc-6d79a50a645f",
    "dub_url": "avatars/.../dubs/1ddf...wav",
    "generated_text": "some text"
}
```

Поля сообщения включают:

- **event** — тип события (save_response_dub);
- **message_id** — уникальный идентификатор запроса;
- **dub_url** — путь к аудиофайлу в хранилище;
- **generated_text** — сгенерированный текст ответа.

Сервис генерации озвучки представляет собой устойчивое и расширяемое звено архитектуры Bishop. Он позволяет не только получить текстовый ответ, но и воспроизвести его голосом, делая взаимодействие с системой

ближе к естественному. Гибкость реализации позволяет легко адаптировать сервис под новые задачи и модели.

6.4 Сервис генерации текста

Сервис реализует два основных пайплайна: пайпайн генерации текста и пайпайн обучения модели. Оба пайплайна построены из независимых, переиспользуемых компонентов, в соответствии с принципами чистой архитектуры, что позволяет гибко адаптировать процессы под изменение базовой модели и требований к системе. Архитектура сервиса во многом схожа с архитектурой сервиса генерации звука: взаимодействие между компонентами осуществляется посредством сообщений, поступающих через Kafka. Сервис ожидает команды на запуск соответствующего пайплайна, а после завершения работы формирует и отправляет результат обратно через Kafka.

6.4.1 Пайпайн обучения модели

Обучение модели является ресурсозатратной и длительной по времени операцией, поэтому запуск реализован как отдельный процесс. Благодаря этому он остается доступным и может сообщать другим компонентам системы о текущем состоянии обучения, не блокируя основной поток.

Для контроля состояния процесса и исключения повторного запуска одновременно нескольких экземпляров пайплайна, используется механизм хранения информации о процессе в Redis. Это реализовано через класс RedisManager на основе библиотеки redis, который предоставляет методы для записи, чтения и удаления информации о текущем обучении. Таким образом, если процесс уже запущен, повторный запуск будет отклонен, потому что предполагается, что все ресурсы системы в это время заняты.

Для запуска и контроля отдельного процесса используются функции на основе библиотеки multiprocessing, выделенные в отдельный инфраструктурный менеджер, реализующие запуск задачи в отдельном процессе, проверка есть ли запущенный процесс и возможность принудительного завершения процесса:

```
def run_task_in_process(func, *args):
    if is_process_running():
```

```

        logger.warning("A process is already running.
                         Rejecting new request.")
        return

    logger.info("Launching task in a separate process")
    p = Process(target=func, args=args)
    p.start()

    redis_manager.set_running_process_id(p.pid)
    logger.info(f"Started task with PID {p.pid}")

def is_process_running() -> bool:
    process_id = redis_manager.get_running_process_id()
    if process_id:
        logger.info(f"Process with PID {process_id} is
                    already running.")
        return True
    return False

def terminate_process(pid: int) -> bool:
    try:
        os.kill(pid, signal.SIGTERM)
        logger.info(
            f"Successfully sent termination signal to
            process with PID {pid}")
        return True
    except ProcessLookupError:
        logger.warning(f"Process with PID {pid} not found.")
        return False
    except Exception as e:
        logger.error(f"Error while trying to terminate
                     process {pid}: {e}")
        return False

```

Этап подготовки данных

После запуска отдельного процесса обучения первым этапом является обработка данных, включающая в себя три этапа:

- Загрузка всех файлов из S3-хранилища:

```

def download_data(
    s3_urls: List[Path],

```

```

        output_dir: Path = Path(settings.RAW_DATA_DIR
            ),
    ) -> List[Path]:
        logger.info("Processing dataset...")
        local_files = []

        for s3_url in tqdm(s3_urls, desc="Downloading
            files"):
            try:
                local_path = s3_storage.download_file(
                    s3_url)
                local_files.append(local_path)
            except Exception as e:
                logger.error(f"Error downloading {s3_url
                    }: {e}")

        logger.info(f"Processing dataset complete. Files
            saved to {output_dir}")
        return local_files

```

- Извлечение аудио и транскрипция. Для этого используется класс Transcribator, основанный на библиотеке moviepy для извлечения аудио из видео и модели Whisper от OpenAI для транскрипции звуковых дорожек в текст:

```

class Transcribator:
    def __init__(self, model_type: str = "base"):
        self.model_type = model_type
        self.model = whisper.load_model(self.
            model_type)

    def transcribe_audio(
        self,
        audio_path: Path,
        output_dir: Path = Path(settings.
            INTERIM_DATA_DIR),
    ) -> Path:
        output_dir.mkdir(parents=True, exist_ok=True)
        result = self.model.transcribe(str(audio_path
            ))
        output_path = output_dir / (audio_path.stem +
            ".txt")
        with output_path.open('w', encoding='utf-8'):
            as f:

```

```

        f.write(result['text'])
    return str(output_path)

def transcribe_video(
    self,
    video_path: Path,
    output_dir: Path = Path(settings.
        INTERIM_DATA_DIR),
) -> Path:
    output_dir.mkdir(parents=True, exist_ok=True)
    temp_audio = output_dir / "temp_audio.wav"
    try:
        video = VideoFileClip(str(video_path))
        video.audio.write_audiofile(str(
            temp_audio), codec='pcm_s16le', logger=
            None)
        video.close()
        result = self.model.transcribe(str(
            temp_audio))
        output_path = output_dir / (video_path.
            stem + ".txt")
        with output_path.open('w', encoding='utf
            -8') as f:
            f.write(result['text'])
    return str(output_path)
finally:
    if temp_audio.exists():
        temp_audio.unlink()

```

- Формирование финального датасета. Происходит очистка текста от символов разметки и разделение длинных строк на разные:

```

def _clean_text(text: str) -> str:
    text = re.sub(r"<[^>]+>", "", text)
    text = normalize("NFKC", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()

def _split_into_chunks(text: str, max_len: int =
    1000, min_len: int = 50) -> list[str]:
    text = text.replace("\r\n", "\n").replace("\r", "
        \n")

```

```

rough_chunks = re.split(r"\n{2,}|\.\s", text)
refined_chunks = []

for chunk in rough_chunks:
    chunk = chunk.strip()
    if len(chunk) < min_len:
        continue

    if not re.search(r"[.,!?\s]", chunk):
        i = 0
        while i < len(chunk):
            max_rand_len = min(max_len, len(chunk) - i)
            if max_rand_len < min_len:
                break
            rand_len = random.randint(min_len,
                                       min(max_len, len(chunk) - i))
            part = chunk[i:i + rand_len].strip()
            if len(part) >= min_len:
                refined_chunks.append(part)
            i += rand_len
        continue

    while len(chunk) > max_len:
        split_idx = chunk.rfind(" ", 0, max_len)
        if split_idx == -1:
            split_idx = max_len
        part = chunk[:split_idx].strip()
        if len(part) >= min_len:
            refined_chunks.append(part)
        chunk = chunk[split_idx:].strip()
    if len(chunk) >= min_len:
        refined_chunks.append(chunk)

return refined_chunks

def build_dataset(
    input_dir: Path = Path(settings.INTERIM_DATA_DIR),
    ,
    output_file: Path = Path(settings.
        PROCESSED_DATA_DIR) / "dataset.txt",
) -> Path:

```

```

        output_file.parent.mkdir(parents=True, exist_ok=True)
        text_files = list(input_dir.glob("*.txt"))

        all_chunks = []

        for file in text_files:
            raw_text = file.read_text(encoding="utf-8")
            if not raw_text.strip():
                continue
            cleaned = _clean_text(raw_text)
            chunks = _split_into_chunks(cleaned)
            all_chunks.extend(chunks)

        unique_chunks = list(dict.fromkeys(all_chunks))

        with output_file.open("w", encoding="utf-8") as f_out:
            for chunk in unique_chunks:
                f_out.write(chunk + "\n")

        logger.info(f"Cleaned dataset with {len(unique_chunks)} samples saved at {output_file}")
    )
    return output_file

```

Этап запуска обучения

Следующим этапом после формирования датасета является обучение модели, реализованного на библиотеках datasets и transformers. В зависимости от конфигурации, обучение может происходить локально либо на платформе Kaggle, где процесс контролируется и отслеживается через API. Управление этим шагом реализовано с помощью класса KaggleManager, который инкапсулирует всю логику взаимодействия с Kaggle API, включая аутентификацию, загрузку датасета, запуск Python-скрипта, отслеживание его выполнения и скачивание результатов:

```

class KaggleManager:

    def __init__(self) -> None:
        self._api = KaggleApi()

```

```

def authenticate(self) -> None:
    try:
        self._api.authenticate()
    except Exception as e:
        raise KaggleAuthenticationError("Failed to
                                         authenticate with Kaggle API") from e

def upload_dataset(self, dataset_dir: Path) -> None:

    dataset_config_path = self._generate_dataset_config(
        dataset_dir)

    try:
        self._api.dataset_create_version(
            folder=dataset_dir,
            version_notes="A universal version of the
                           dataset for launching training",
        )
        logger.info(f"Dataset uploaded successfully from
                    {dataset_dir}."
                    f"Waiting 7 seconds for Kaggle to
                     update the data.")
        time.sleep(7)

        dataset_config_path.unlink(missing_ok=True)
        logger.info("Dataset metadata file deleted.")
    except Exception as e:
        raise KaggleManagerError(f"Failed to create a
                                  new version of the dataset from {dataset_dir
                                  }.") from e

def run_script(self, py_script_path: Path) -> str:
    self._generate_py_script_config(py_script_path)

    try:
        logger.info(f"Uploading python script {
                    py_script_path} to Kaggle...")
        response = self._api.kernels_push(
            folder=py_script_path.parent,
            timeout=settings.KAGGLE_KERNEL_RUN_TIMEOUT,
        )

```

```

        logger.info(f"Python script {py_script_path}\nuploaded successfully.")

        py_script_path.unlink(missing_ok=True)
        logger.info(f"Kernel metadata deleted.")

    kernel_ref = f"{settings.KAGGLE_AUTH_NAME}/{\n        settings.KAGGLE_KERNEL_TITLE}"
    return kernel_ref
except Exception as e:
    raise KaggleManagerError(f"Failed to upload\n        python script {py_script_path} to Kaggle.")
from e

def track_execution(self, kernel_ref: str) -> None:
    while True:
        status_response = self._api.kernels_status(
            kernel_ref)
        status = status_response.status.name

        if status == "COMPLETE":
            logger.info(f"Kernel '{kernel_ref}'\ncompleted successfully.")
            return
        elif status in (
            "ERROR",
            "CANCEL_REQUESTED",
            "CANCEL_ACKNOWLEDGED",
        ):
            error_msg = f"Kernel '{kernel_ref}' failed\n            with status: {status}"
            if failure_message := status_response.
                failure_message:
                    error_msg += f" | Message: {\n                        failure_message}"
            raise RuntimeError(error_msg)
        elif status == "QUEUED":
            logger.info(f"Kernel '{kernel_ref}' is in\n            queue...")
            time.sleep(5)
        elif status in (
            "RUNNING",
            "NEW_SCRIPT"

```

```

    ):

        logger.info(f"Kernel '{kernel_ref}' is in
                    progress...")
        time.sleep(3)

    def download_output(self, kernel_ref: str, output_dir: Path) -> List[Path]:
        if not output_dir.exists():
            output_dir.mkdir(parents=True, exist_ok=True)

        if not output_dir.is_dir():
            raise ValueError("Provided path must be a
                             directory.")

    try:
        logger.info(f"Downloading output of kernel '{
                    kernel_ref}' to '{output_dir}'...")
        outfiles, token = self._api.kernels_output(
            kernel=kernel_ref,
            path=output_dir,
            force=True
        )
        logger.info(f"Downloading output of kernel '{
                    kernel_ref}' to '{output_dir}' completed
                     successfully.")
    except Exception as e:
        raise RuntimeError(f"Failed to download output
                           for kernel '{kernel_ref}'") from e

    if not outfiles:
        logger.warning(f"No output files found for
                      kernel '{kernel_ref}'")
    else:
        logger.info(f"Downloaded {len(outfiles)} files
                     from kernel '{kernel_ref}'")

    return [Path(f) for f in outfiles]

def __generate_py_script_config(self, py_script_path: Path) -> None:
    if not py_script_path.is_file():
        raise ValueError(f"Provided python script path {
                         py_script_path} is not a valid file.")

```

```

kernel_metadata_file = py_script_path.parent / "kernel-metadata.json"

dataset_ref = f"{{settings.KAGGLE_AUTH_NAME}}/{{settings.KAGGLE_DATASET_TITLE}}"

kernel_metadata = {
    "id": f"{{settings.KAGGLE_AUTH_NAME}}/{{settings.KAGGLE_KERNEL_TITLE}}",
    "title": settings.KAGGLE_KERNEL_TITLE.replace("-",
        " ").capitalize(),
    "code_file": py_script_path.name,
    "language": "python",
    "kernel_type": "script",
    "is_private": True,
    "enable_gpu": True,
    "enable_tpu": False,
    "enable_internet": True,
    "dataset_sources": [dataset_ref],
    "competition_sources": [],
    "kernel_sources": [],
    "model_sources": [],
}

with open(kernel_metadata_file, 'w') as f:
    json.dump(kernel_metadata, f, indent=4)

logger.info(f"Kernel metadata created at {py_script_path}")

def _generate_dataset_config(self, dataset_dir: Path) -> Path:
    if not dataset_dir.is_dir():
        raise ValueError(f"Provided dataset folder path {dataset_dir} is not a valid directory.")

dataset_metadata_file = dataset_dir / "dataset-metadata.json"

dataset_metadata = {
    "id": f"{{settings.KAGGLE_AUTH_NAME}}/{{settings.KAGGLE_DATASET_TITLE}}",
}

```

```

        "title": settings.KAGGLE_DATASET_TITLE.replace(
            "-", " ")
            .capitalize(),
        "licenses": [{"name": "CC0-1.0"}],
        "is_public": False,
    }

    with open(dataset_metadata_file, 'w') as f:
        json.dump(dataset_metadata, f, indent=4)

    logger.info(f"Dataset metadata created at {
        dataset_metadata_file}")

    return dataset_metadata_file

```

Этап сохранения результатов

Завершающим этапом пайплайна обучения является сохранение результатов в S3 и отправка сообщения об окончании в Kafka. Обученная модель передаётся с помощью метода upload_file() класса S3StorageWithCache. Этот класс реализует работу с объектным хранилищем на базе библиотеки MinIO:

```

class S3StorageWithCache:
    def __init__(
        self,
        cache_dir: Path,
    ):
        self.client = minio_client
        self.bucket_name = settings.MINIO_BUCKET
        self.cache_dir = cache_dir
        self.cache_dir.mkdir(parents=True, exist_ok=True)

    def upload_file(self, local_path: Path, object_key: str):
        self.client.fput_object(
            bucket_name=self.bucket_name,
            object_name=object_key,
            file_path=str(local_path),
        )

    logger.info(f"Uploaded file {local_path}, with
        object_key {object_key}")

```

7 Результаты

Раздел посвящён краткому изложению основных результатов экспериментального тестирования системы. Каждый этап подтверждён логами и сопровождается иллюстрациями, представленными в приложении Б.

7.1 Инициализация системы

После запуска всех микросервисов система переходит в состояние полной готовности. Доступность компонентов подтверждается с помощью механизма периодического *health-check*, реализованного через брокер Kafka. Диагностические сообщения успешно обрабатываются всеми службами (см. рис. 20).

7.2 Регистрация и создание аватара

Пользователь успешно проходит регистрацию и авторизацию, после чего инициализируется пустой список аватаров. Создание нового аватара выполняется без ошибок, и он получает статус доступности (см. рис. 21, 22, 23).

7.3 Загрузка обучающих материалов

Текстовые и аудиофайлы проходят валидацию и сохраняются в объектное хранилище. Метаданные регистрируются и становятся доступными для дальнейшего использования (см. рис. 24, 25).

7.4 Обучение аватара

Процедура дообучения запускается после получения команды `train_start`. Компоненты обрабатывают данные, а по завершении аватар возвращается в состояние `available` (см. рис. 26, 27, 28, 29).

7.5 Диалог и синтез речи

Пользователь отправляет сообщение. Система формирует текстовый ответ, выполняет синтез речи и возвращает результат в интерфейс (см. рис. 30, 31, 32, 33, 34).

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы была достигнута поставленная цель — разработан сервис, позволяющий создавать цифровых аватаров на основе пользовательских данных. В процессе реализации были решены следующие задачи: обеспечена возможность обучения аватаров на загружаемых аудио-, видео- и текстовых данных, реализована функция текстового взаимодействия с обученным аватаром, а также возможность озвучивания сообщений с использованием синтезированной речи.

До начала разработки была проведена аналитическая работа: изучены существующие решения на рынке, а также исследована патентная база, что позволило учесть текущие тренды и избежать дублирования. На основании анализа фреймворков и библиотек произведена архитектурная декомпозиция сервиса на независимые компоненты, что обеспечило гибкость и масштабируемость при разработке.

Результатом работы стал прототип полноценного сервиса с веб-интерфейсом, реализующий весь необходимый функционал.

В перспективе дальнейшего развития сервиса, при наличии более высоких вычислительных ресурсов, возможно использование более сложных языковых и акустических моделей, что позволит значительно повысить качество взаимодействия с цифровыми аватарами. Кроме того, увеличение вычислительных мощностей откроет возможность ускоренной обработки запросов и позволит рассматривать направление генерации видео-аватаров, что расширит функциональность системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. nanosemantics.ai. «Наносемантика» разработала для ЛДПР первый в мире политический алгоритм — нейросеть «Жириновский» // Nanosemantics Blog. — 2023. — URL: <https://nanosemantics.ai/blog/656>.
2. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. — O'Reilly Media, 2017.
3. Ramírez S. FastAPI: High-performance web framework for building APIs with Python. — 2018. — URL: <https://github.com/tiangolo/fastapi> ; Accessed: 2025-03-19.
4. Transformers: State-of-the-Art Natural Language Processing / T. Wolf [и др.] // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. — Association for Computational Linguistics. 2020. — C. 38–45. — URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
5. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark / Y. Wang [и др.] // arXiv preprint arXiv:2406.01574. — 2024. — arXiv: [2406.01574 \[cs.CL\]](https://arxiv.org/abs/2406.01574). — URL: <https://arxiv.org/abs/2406.01574>.
6. Russian SuperGLUE 1.1: Revising the Lessons not Learned by Russian NLP models / A. Fenogenova [и др.] // arXiv preprint arXiv:2202.07791. — 2022. — arXiv: [2202.07791 \[cs.CL\]](https://arxiv.org/abs/2202.07791). — URL: <https://arxiv.org/abs/2202.07791>.
7. MERA: A Comprehensive LLM Evaluation in Russian / A. Fenogenova [и др.] // arXiv preprint arXiv:2401.04531. — 2024. — arXiv: [2401.04531 \[cs.CL\]](https://arxiv.org/abs/2401.04531). — URL: <https://arxiv.org/abs/2401.04531>.
8. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference / W.-L. Chiang [и др.] // arXiv preprint arXiv:2403.04132. — 2024. — arXiv: [2403.04132 \[cs.AI\]](https://arxiv.org/abs/2403.04132). — URL: <https://arxiv.org/abs/2403.04132>.
9. Libra: Building Decoupled Vision System on Large Language Models / Y. Xu [и др.] // arXiv preprint arXiv:2405.10140. — 2024. — arXiv: [2405.10140 \[cs.CV\]](https://arxiv.org/abs/2405.10140). — URL: <https://arxiv.org/abs/2405.10140>.
10. The Llama 3 Herd of Models / A. Grattafiori [и др.] // arXiv preprint arXiv:2407.21783. — 2024. — arXiv: [2407.21783 \[cs.AI\]](https://arxiv.org/abs/2407.21783). — URL: <https://arxiv.org/abs/2407.21783>.

11. Mistral 7B / A. Q. Jiang [и др.] // arXiv preprint arXiv:2310.06825. — 2023. — arXiv: [2310.06825 \[cs.CL\]](https://arxiv.org/abs/2310.06825). — URL: <https://arxiv.org/abs/2310.06825>.
12. Qwen3 Technical Report / A. Yang [и др.] // arXiv preprint arXiv:2505.09388. — 2025. — arXiv: [2505.09388 \[cs.CL\]](https://arxiv.org/abs/2505.09388). — URL: <https://arxiv.org/abs/2505.09388>.
13. DeepSeek-AI. DeepSeek-V3 Technical Report // arXiv preprint arXiv:2412.19437. — 2025. — arXiv: [2412.19437 \[cs.CL\]](https://arxiv.org/abs/2412.19437). — URL: <https://arxiv.org/abs/2412.19437>.
14. Gusev I. rulm: Language Modeling and Instruction Tuning for Russian. — 2025. — URL: <https://github.com/IlyaGusev/rulm>; GitHub repository (archived on March 9, 2025). <https://github.com/IlyaGusev/rulm>.
15. A Family of Pretrained Transformer Language Models for Russian / D. Zmitrovich [и др.] // arXiv preprint arXiv:2309.10931. — 2023. — arXiv: [2309.10931 \[cs.CL\]](https://arxiv.org/abs/2309.10931). — URL: <https://arxiv.org/abs/2309.10931>.
16. Tikhomirov M., Chernyshev D. Facilitating Large Language Model Russian Adaptation with Learned Embedding Propagation // arXiv preprint arXiv:2412.21140. — 2024. — arXiv: [2412.21140 \[cs.CL\]](https://arxiv.org/abs/2412.21140). — URL: <https://arxiv.org/abs/2412.21140>.
17. Vikhr: The Family of Open-Source Instruction-Tuned Large Language Models for Russian / A. Nikolich [и др.] // arXiv preprint arXiv:2405.13929. — 2024. — arXiv: [2405.13929 \[cs.CL\]](https://arxiv.org/abs/2405.13929). — URL: <https://arxiv.org/abs/2405.13929>.
18. Gusev I. PingPong: A Benchmark for Role-Playing Language Models with User Emulation and Multi-Model Evaluation // arXiv preprint arXiv:2409.06820. — 2025. — arXiv: [2409.06820 \[cs.CL\]](https://arxiv.org/abs/2409.06820). — URL: <https://arxiv.org/abs/2409.06820>.
19. Whisper-PMFA: Partial Multi-Scale Feature Aggregation for Speaker Verification using Whisper Models / Y. Zhao [и др.] // arXiv preprint arXiv:2408.15585. — 2024. — arXiv: [2408.15585 \[cs.SD\]](https://arxiv.org/abs/2408.15585). — URL: <https://arxiv.org/abs/2408.15585>.
20. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations / A. Baevski [и др.] // arXiv preprint arXiv:2006.11477. — 2020. — arXiv: [2006.11477 \[cs.CL\]](https://arxiv.org/abs/2006.11477). — URL: <https://arxiv.org/abs/2006.11477>.

21. Apache Software Foundation. Apache Kafka. — Accessed: 2025-03-19. <https://kafka.apache.org/>.
22. Docker, Inc. Docker: Accelerated Container Application Development. — 2023. — URL: <https://www.docker.com/> ; Accessed: 2025-03-19.
23. Jemine C. Real-Time Voice Cloning : дис. . . .маг. / Jemine Corentin. — Liège, Belgique : Université de Liège, 2019. — URL: <https://matheo.uliege.be/handle/2268.2/6801> ; Unpublished master's thesis.
24. Gölge E., Team T. C. T. Coqui TTS: A deep learning toolkit for Text-to-Speech. — 2021. — URL: <https://github.com/coqui-ai/TTS> ; MPL-2.0 License.
25. MetaVoice Team. MetaVoice-1B: Foundational Model for Human-like, Expressive TTS. — 2024. — Accessed: 2025-05-12. <https://github.com/metavoiceio/metavoice-src>.
26. Llasa: Scaling Train-Time and Inference-Time Compute for Llama-based Speech Synthesis / Z. Ye [и др.] // arXiv preprint arXiv:2502.04128. — 2025. — arXiv: [2502.04128 \[eess.AS\]](https://arxiv.org/abs/2502.04128). — URL: <https://arxiv.org/abs/2502.04128>.

Приложение А

Интерфейс

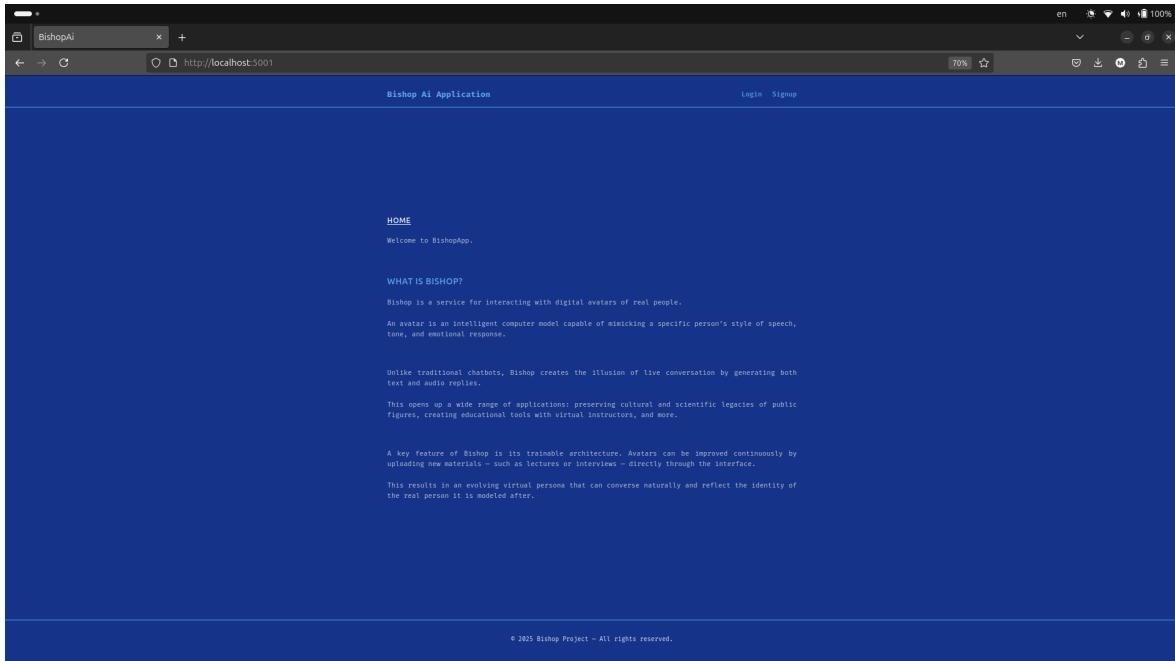


Рисунок 12 — Гостевая страница

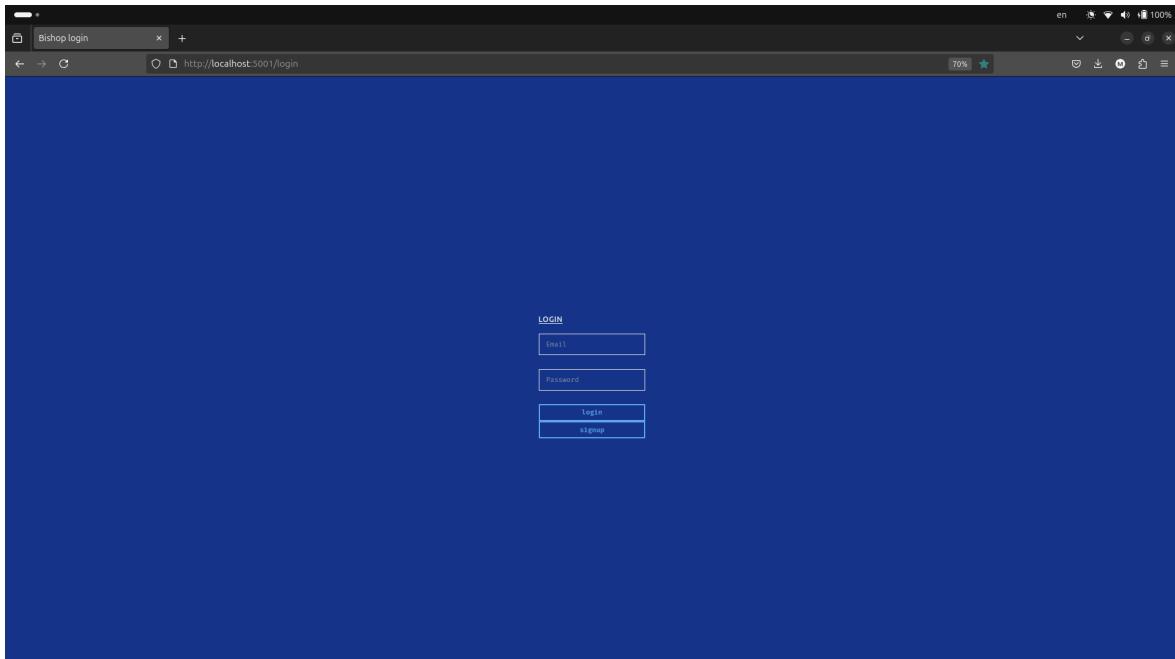


Рисунок 13 — Страница аутентификации

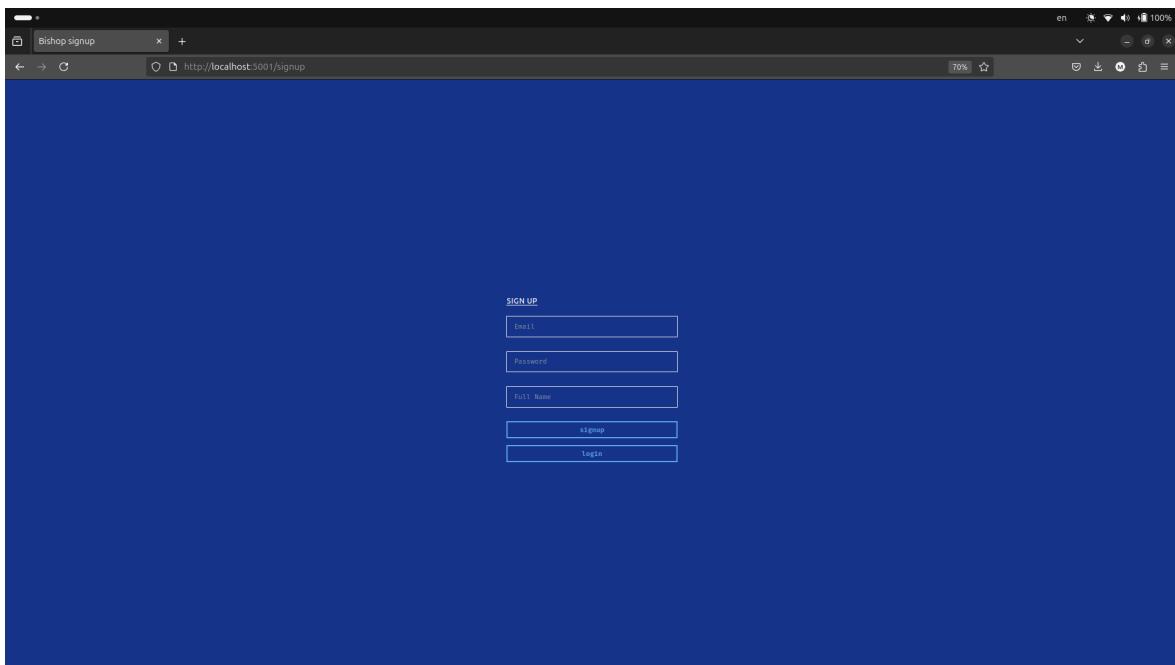


Рисунок 14 — Страница регистрации

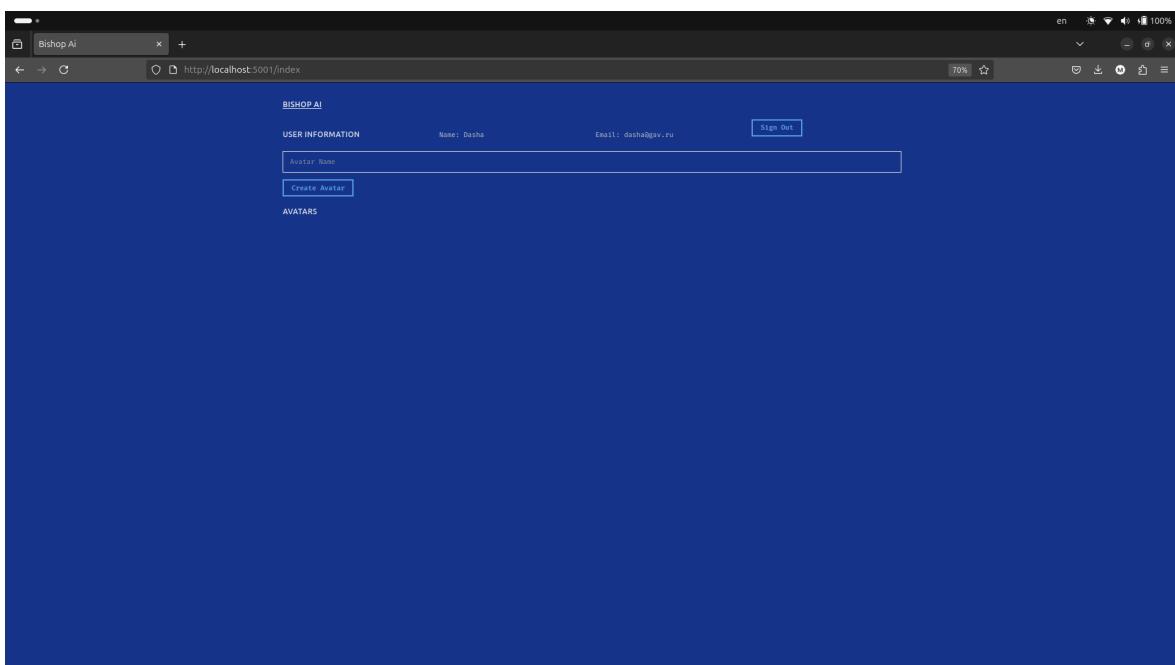


Рисунок 15 — Основная страница приложения

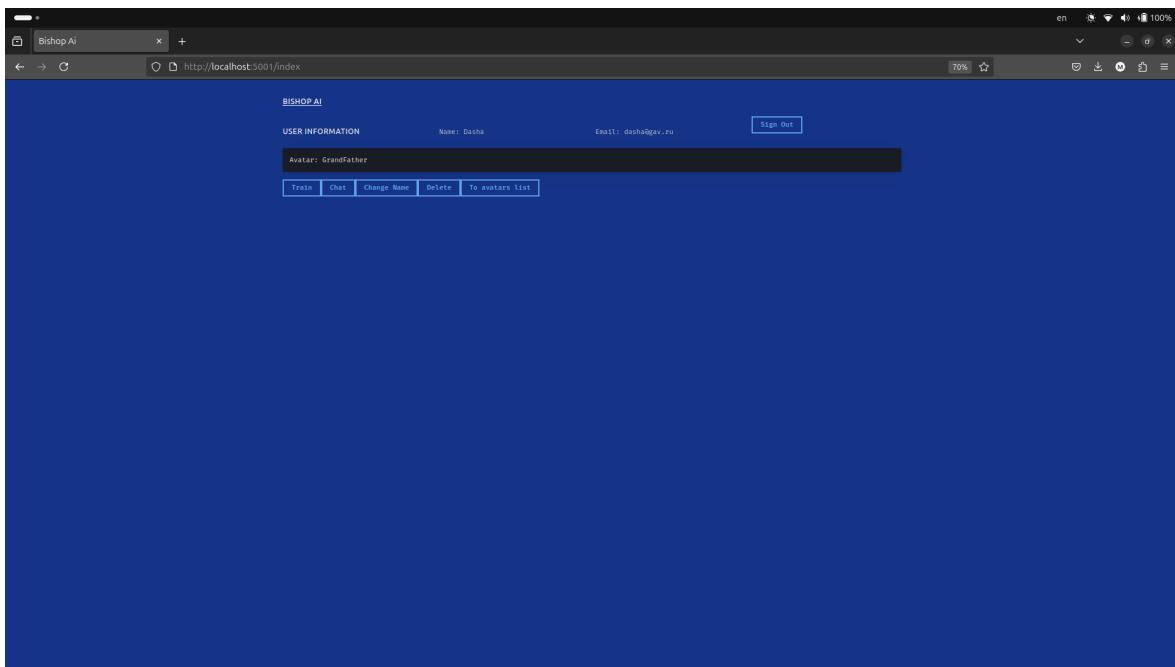


Рисунок 16 — Страница управления аватарами

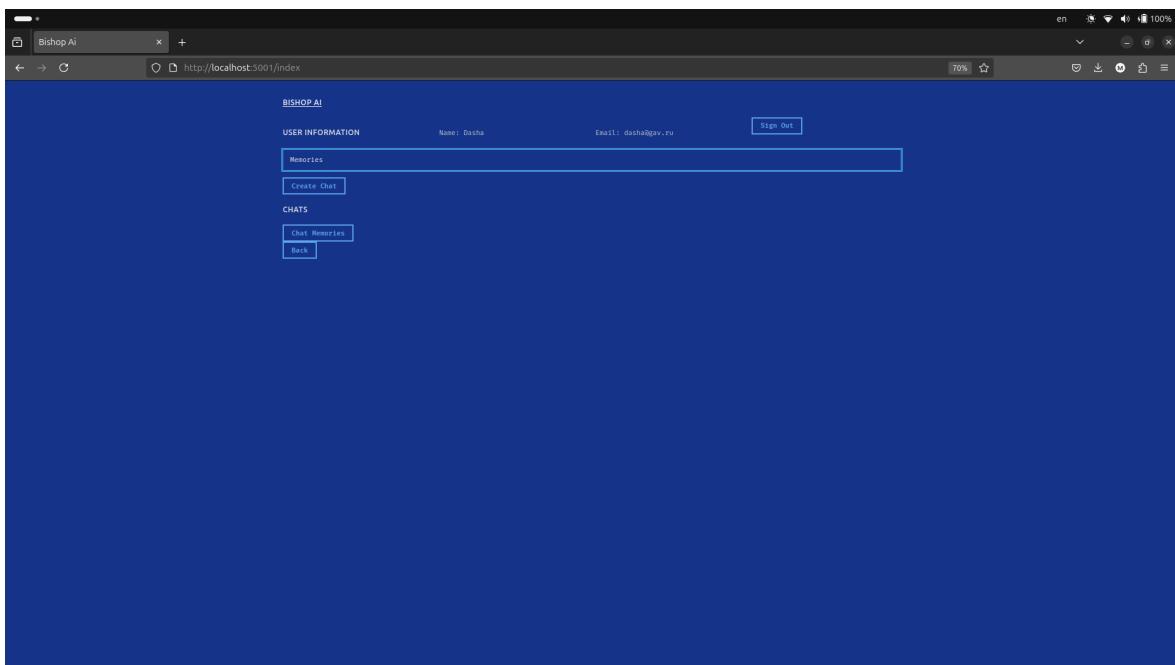


Рисунок 17 — Страница управления чатами

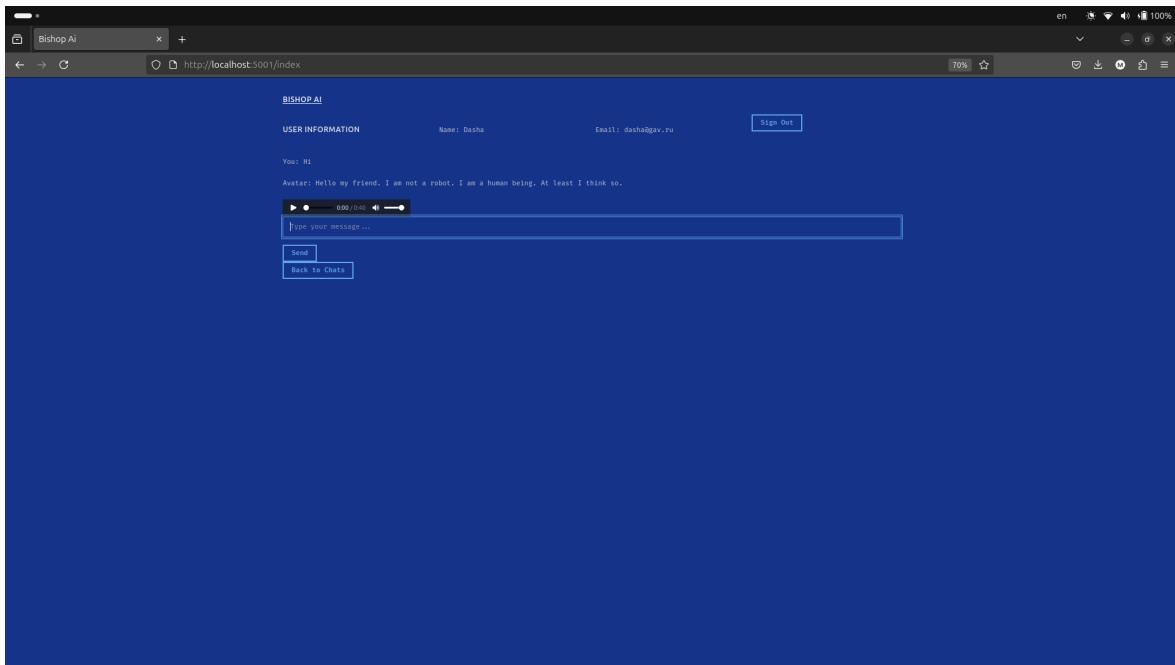


Рисунок 18 — Страница чата

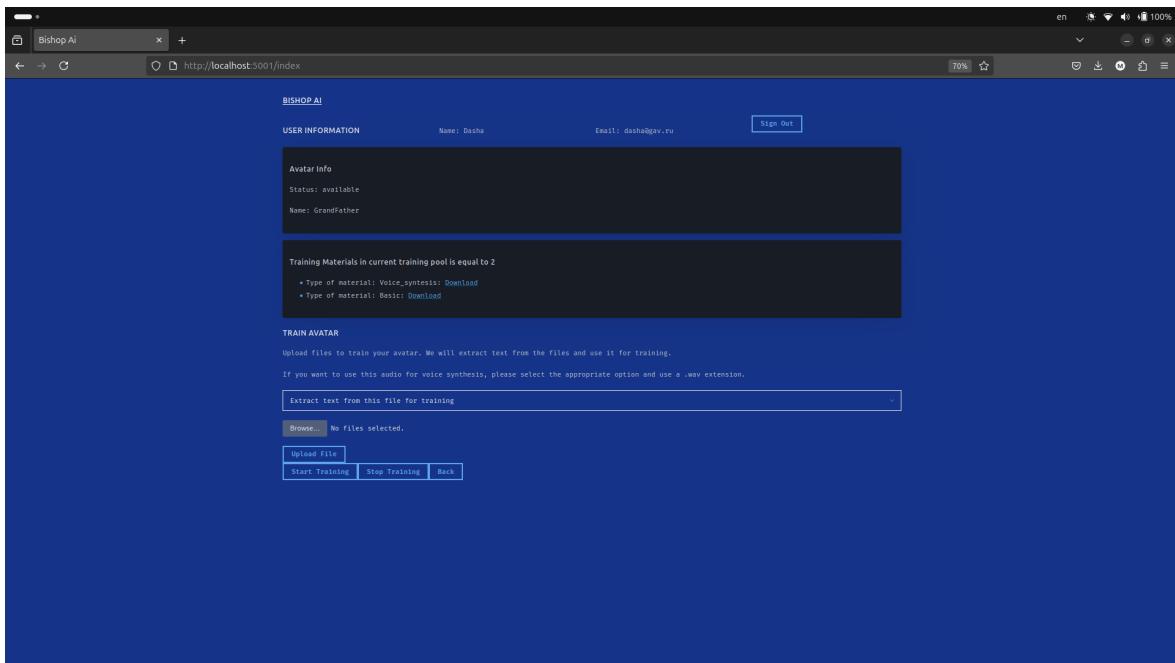


Рисунок 19 — Интерфейс обучения аватара

Приложение Б

Отчёты о работе сервиса

Рисунок 20 — Фрагмент логов инициализации системы и периодической проверки состояния доступности системы

```
mal@wooden-tool ~| docker logs bishop-frontend-1 -f
INFO: Will watch for changes in these directories: ['/app']
INFO: Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
INFO: Started reloader process [13] using WatchFiles
INFO: Started server process [15]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 172.18.0.1:59928 - "GET / HTTP/1.1" 200 OK
INFO: 172.18.0.1:45070 - "GET /signup HTTP/1.1" 200 OK
{'email': 'dasha@gav.ru', 'password': '12341234', 'full_name': 'Dasha'}
{'email': 'dasha@gav.ru', 'is_active': True, 'is_superuser': False, 'full_name': 'Dasha', 'id': 'd62a88bb-6d7d-48fe-8508-f6f26ace034d'}
INFO: 172.18.0.1:45070 - "POST /signup HTTP/1.1" 303 See Other
INFO: 172.18.0.1:45070 - "GET /login HTTP/1.1" 200 OK
INFO: 172.18.0.1:45070 - "GET /login HTTP/1.1" 200 OK
{'username': 'dasha@gav.ru', 'password': '12341234', 'grant_type': 'password', 'scope': None, 'client_id': None, 'client_secret': None}
{'access_token': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE3NDg5NzE2NjgsInN1YiI6ImQ2MmE40GJiLTZkN2QtNDhmZS04NTA4LWY2ZjI2YWNIiMDM0ZCJ9.JU7--RDphjw8aZ4vd1MeQf2hdv_QCAP83ZVzmUsZUI', 'token_type': 'bearer'}
{'detail': 'Not authenticated'}
INFO: 172.18.0.1:45070 - "POST /login HTTP/1.1" 303 See Other
INFO: 172.18.0.1:45070 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:45070 - "GET /index HTTP/1.1" 200 OK
```

Рисунок 21 — Регистрация и авторизация пользователя

```

INFO: 127.0.0.1:32231 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:49200 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:45286 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:36962 - "POST /api/v1/users/signup HTTP/1.1" 200 OK
INFO: 172.18.0.2:36968 - "POST /api/v1/login/access-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:36968 - "GET /api/v1/users/me HTTP/1.1" 401 Unauthorized
INFO: 172.18.0.2:36972 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:36980 - "GET /api/v1/users/me HTTP/1.1" 200 OK
2025-05-26 17:27:48 - backend_logger - INFO - Reading avatars for current user d62a88bb-6d7d-48fe-8508-f6f26ace034d
INFO: 172.18.0.2:36994 - "GET /api/v1/avatars/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:36998 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:37004 - "GET /api/v1/users/me HTTP/1.1" 200 OK
2025-05-26 17:27:50 - backend_logger - INFO - Reading avatars for current user d62a88bb-6d7d-48fe-8508-f6f26ace034d
INFO: 172.18.0.2:37016 - "GET /api/v1/avatars/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:47308 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:40148 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:54692 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:55558 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
[0] 1:server 2:frontend* 3:backend* 4:llm 5:sound 6:test           "docker logs bishop-b " 20:29 26-May-25

```

Рисунок 22 — Обработка регистрации и авторизации на стороне backend-сервиса

```

INFO: 127.0.0.1:47078 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:44434 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
2025-05-26 17:30:27 - backend_logger - INFO - Creating avatar for user d62a88bb-6d7d-48fe-8508-f6f26ace034d
2025-05-26 17:30:27 - backend_logger - INFO - Setting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd to available
INFO: 172.18.0.2:44444 - "POST /api/v1/avatars/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:44452 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:44454 - "GET /api/v1/users/me HTTP/1.1" 200 OK
2025-05-26 17:30:27 - backend_logger - INFO - Reading avatars for current user d62a88bb-6d7d-48fe-8508-f6f26ace034d
INFO: 172.18.0.2:44464 - "GET /api/v1/avatars/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:44476 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
2025-05-26 17:30:28 - backend_logger - INFO - Reading avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
INFO: 172.18.0.2:44478 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd HTTP/1.1" 200 OK
INFO: 172.18.0.2:44484 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:44498 - "GET /api/v1/users/me HTTP/1.1" 200 OK
2025-05-26 17:30:29 - backend_logger - INFO - Reading avatars for current user d62a88bb-6d7d-48fe-8508-f6f26ace034d
INFO: 172.18.0.2:44508 - "GET /api/v1/avatars/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:41122 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:36578 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
[0] 1:server 2:frontend* 3:backend* 4:llm 5:sound 6:test           "docker logs bishop-b " 20:31 26-May-25

```

Рисунок 23 — Создание нового цифрового аватара пользователем

```

INFO: 172.18.0.1:56638 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:56638 - "GET /avatar/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd HTTP/1.1" 200 OK
INFO: 172.18.0.1:56638 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:55260 - "GET /avatar/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train_widget HTTP/1.1" 200 OK
INFO: 172.18.0.1:42088 - "POST /avatar/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train HTTP/1.1" 200 OK
INFO: 172.18.0.1:42088 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:42102 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:41244 - "POST /avatar/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train HTTP/1.1" 200 OK
INFO: 172.18.0.1:45184 - "GET /index HTTP/1.1" 200 OK
INFO: 172.18.0.1:41244 - "GET /index HTTP/1.1" 200 OK
[0] 1:server 2:frontend* 3:backend* 4:llm 5:sound 6:test           "docker logs bishop-b " 20:33 26-May-25

```

Рисунок 24 — Загрузка текстовых и аудиофайлов через интерфейс

Рисунок 25 — Фиксация и сохранение обучающих материалов на стороне backend

```
[INFO]: 127.0.0.1:33318 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:57620 - "POST /api/v1/login/test token HTTP/1.1" 200 OK
2025-05-26 17:37:01 - backend_logger - INFO - Getting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd is available
2025-05-26 17:37:01 - backend_logger - INFO - Setting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd to training
2025-05-26 17:37:01 - backend_logger - INFO - Getting available training materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Found 2 untrained materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Sending START training message for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
Topic llm-train is not available during auto-create initialization
2025-05-26 17:37:01 - backend_logger - INFO - Sent message to topic 'llm-train': {'event': 'train_start', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd', 'train_materials': [{"id": "ca22187c-5d71-4bc8-bb06-90ae36322a41", "type": "basic", "url": "http://:s3:9000/app/users/d62a8bb8-b6d7d-4f8-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60ba865-3692-4fb-83c1-64cb0e2c084c.txt"}, {"id": "3bb7dd2d-b6dc-4b5d-9c4-424e10f953df", "type": "voice_synthesis", "url": "http://:s3:9000/app/users/d62a8bb8-b6d7d-4f8-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav"}]}
2025-05-26 17:37:01 - backend_logger - INFO - Message sent: b'{"event": "train_start", "avatar_id": "8c9cd04b-bb63-4680-b9e1-a1d7c03847fd", "train_materials": [{"id": "ca22187c-5d71-4bc8-bb06-90ae36322a41", "type": "basic", "url": "http://:s3:9000/app/users/d62a8bb8-b6d7d-4f8-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60ba865-3692-4fb-83c1-64cb0e2c084c.txt"}, {"id": "3bb7dd2d-b6dc-4b5d-9c4-424e10f953df", "type": "voice_synthesis", "url": "http://:s3:9000/app/users/d62a8bb8-b6d7d-4f8-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav"]}]'
2025-05-26 17:37:01 - backend_logger - INFO - Getting available training materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Invalidated training materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Invalidated 2 materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Getting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:37:01 - backend_logger - INFO - Status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd is training
2025-05-26 17:37:01 - backend_logger - INFO - END of training status check for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd: training
INFO: 172.18.0.2:57636 - "POST /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train/start HTTP/1.1" 200 OK
2025-05-26 17:37:01 - backend_logger - INFO - Reading avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
INFO: 172.18.0.2:57638 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd HTTP/1.1" 200 OK
2025-05-26 17:37:01 - backend_logger - INFO - Getting available training materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
train_materials: data=[] count=0
INFO: 172.18.0.2:57638 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train/materials HTTP/1.1" 200 OK
INFO: 127.0.0.1:42594 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
[0] 1:server 2:frontend 3:backend 4:llm 5:sound 6:gitest
[docker logs bishop:b "20:37:26-May-25
```

Рисунок 26 — Инициация процесса обучения через backend

```

[24-] warnings.warn(message, stacklevel=1)
2025-05-26 17:02:31 - ml_logger - INFO - Subscribed to topics: ['health-check-l1m', 'l1m-inference', 'l1m-train']
[3|1748278951.243|FAIL|rdkafka#producer-2| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 3ms in state CONNECT)
[3|1748278951.246|FAIL|rdkafka#consumer-3| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 2ms in state CONNECT)
[3|1748278951.246|FAIL|rdkafka#producer-1| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 4ms in state CONNECT)
[3|1748278952.239|FAIL|rdkafka#producer-1| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 0ms in state CONNECT, 1 identical error(s) suppressed)
[3|1748278952.240|FAIL|rdkafka#producer-2| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 0ms in state CONNECT, 1 identical error(s) suppressed)
[3|1748278952.242|FAIL|rdkafka#consumer-3| [thrd:kafka:9092/bootstrap]: kafka:9092/bootstrap: Connect to ipv4#172.18.0.11:9092 failed: Connection refused (after 0ms in state CONNECT, 1 identical error(s) suppressed)
2025-05-26 17:02:52 - ml_logger - ERROR - Kafka error: KafkaError{code=UNKNOWN_TOPIC_OR_PART,val=3,str="Subscribed topic not available: health-check-l1m: Broker: Unknown topic or partition"}
2025-05-26 17:02:52 - ml_logger - Kafka error: KafkaError{code=UNKNOWN_TOPIC_OR_PART,val=3,str="Subscribed topic not available: l1m-inference: Broker: Unknown topic or partition"}
2025-05-26 17:02:52 - ml_logger - Kafka error: KafkaError{code=UNKNOWN_TOPIC_OR_PART,val=3,str="Subscribed topic not available: l1m-train: Broker: Unknown topic or partition"}
2025-05-26 17:37:04 - ml_logger - INFO - Received task from l1m-train: {'event': 'train_start', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd', 'train_materials': [{ 'id': 'ca22187c-5d71-4bc8-bb06-90a3e6322a41', 'type': 'basic', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60baa865-3692-4fbf-83c1-64cb0e2c08c4.txt'}, { 'id': '3bb7dd2d-b6dc-4b5d-9c94-424e10f953df', 'type': 'voice_synthesis', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'}]}
2025-05-26 17:37:04 - ml_logger - INFO - Execute task with task_data: {'event': 'train_start', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd', 'train_materials': [{ 'id': 'ca22187c-5d71-4bc8-bb06-90a3e6322a41', 'type': 'basic', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60baa865-3692-4fbf-83c1-64cb0e2c08c4.txt'}, { 'id': '3bb7dd2d-b6dc-4b5d-9c94-424e10f953df', 'type': 'voice_synthesis', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'}]}
2025-05-26 17:37:04 - ml_logger - INFO - [Train task] Executing train start pipelines
[24-]
[0] 1:server 2:frontend 3:backend 4:l1m* 5:sound 6:test
"docker logs bishop-l " 20:38 26-May-25

```

Рисунок 27 — Запуск дообучения в модуле генерации текста

```

INFO: 127.0.0.1:39836 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:59556 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
2025-05-26 17:38:32 - backend_logger - INFO - Getting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:38:32 - backend_logger - INFO - Status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd is training
2025-05-26 17:38:32 - backend_logger - INFO - Stopping training for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:38:32 - backend_logger - INFO - Sending STOP training message for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
2025-05-26 17:38:32 - backend_logger - INFO - Sent message to topic 'l1m-train': {'event': 'train_stop', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd'}
2025-05-26 17:38:32 - backend_logger - INFO - Setting training status for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd to available
2025-05-26 17:38:32 - backend_logger - INFO - Training stopped for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
INFO: 172.18.0.2:59572 - "POST /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train/stop HTTP/1.1" 200 OK
2025-05-26 17:38:32 - backend_logger - INFO - Reading avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
INFO: 172.18.0.2:59584 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd HTTP/1.1" 200 OK
2025-05-26 17:38:32 - backend_logger - INFO - Getting available training materials for avatar 8c9cd04b-bb63-4680-b9e1-a1d7c03847fd
train_materials: data=[] count=0
INFO: 172.18.0.2:59584 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/train/materials HTTP/1.1" 200 OK
INFO: 127.0.0.1:36262 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
[24-]
[0] 1:server 2:frontend 3:backend 4:l1m* 5:sound 6:test
"docker logs bishop-b " 20:39 26-May-25

```

Рисунок 28 — Завершение обучения на стороне backend и обновление статуса

```

2025-05-26 17:37:04 - ml_logger - INFO - Received task from l1m-train: {'event': 'train_start', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd', 'train_materials': [{ 'id': 'ca22187c-5d71-4bc8-bb06-90a3e6322a41', 'type': 'basic', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60baa865-3692-4fbf-83c1-64cb0e2c08c4.txt'}, { 'id': '3bb7dd2d-b6dc-4b5d-9c94-424e10f953df', 'type': 'voice_syntesis', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'}]}
2025-05-26 17:37:04 - ml_logger - INFO - Execute task with task_data: {'event': 'train_start', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd', 'train_materials': [{ 'id': 'ca22187c-5d71-4bc8-bb06-90a3e6322a41', 'type': 'basic', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/text/60baa865-3692-4fbf-83c1-64cb0e2c08c4.txt'}, { 'id': '3bb7dd2d-b6dc-4b5d-9c94-424e10f953df', 'type': 'voice_syntesis', 'url': 'http://s3:9000/app/users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'}]}
2025-05-26 17:37:04 - ml_logger - INFO - [Train task] Executing train start pipelines
[24-2025-05-26 17:38:32 - ml_logger - INFO - Received task from l1m-train: {'event': 'train_stop', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd'}
2025-05-26 17:38:32 - ml_logger - INFO - Execute task with task_data: {'event': 'train_stop', 'avatar_id': '8c9cd04b-bb63-4680-b9e1-a1d7c03847fd'}
[0] 1:server 2:frontend 3:backend 4:l1m* 5:sound 6:test
"docker logs bishop-l " 20:38 26-May-25

```

Рисунок 29 — Окончание дообучения модели генерации текста

```

INFO: 172.18.0.2:41978 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:41984 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:41988 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
2025-05-26 17:42:31 - backend_logger - INFO - Sending generate_response message: {'event': 'inference_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hi'}
Topic llm-inference is not available during auto-create initialization
2025-05-26 17:42:31 - backend_logger - INFO - Sent message to topic 'llm-inference': {'event': 'inference_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hi'}
INFO: 172.18.0.2:41990 - "POST /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/ HTTP/1.1" 200 OK
INFO: 172.18.0.2:41992 - "POST /api/v1/Login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:49626 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
\[[24-INFO: 172.18.0.2:49626 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
INFO: 172.18.0.2:49626 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
INFO: 172.18.0.2:49628 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:57422 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
[0] 1:server 2:frontend 3:backend* 4:llm 5:sound 6:test docker logs bishop-b " 20:42 26-May-25

```

Рисунок 30 — Отправка пользовательского сообщения и генерация текста

```

2025-05-26 17:47:32 - ml_logger - INFO - Received task from llm-inference: {'event': 'inference_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hi'}
2025-05-26 17:47:32 - ml_logger - INFO - Execute task with task_data: {'event': 'inference_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hi'}
2025-05-26 17:47:32 - ml_logger - INFO - Executing inference pipelines
2025-05-26 17:47:32 - ml_logger - INFO - Processing inference task with task_data: {'event': 'inference_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hi'}
2025-05-26 17:47:32 - ml_logger - INFO - Sending generate_response message: {'event': 'save_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'generated_text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
[0] 1:server 2:frontend 3:backend* 4:llm 5:sound 6:test docker logs bishop-b " 20:55 26-May-25

```

Рисунок 31 — Генерация ответа llm-моделью по заданному сообщению

```

INFO: 172.18.0.2:56914 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:56920 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
2025-05-26 17:47:32 - backend_logger - INFO - Received task from save-response: {'event': 'save_response', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'generated_text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO: 172.18.0.2:56920 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 200 OK
2025-05-26 17:47:34 - backend_logger - INFO - [get_avatar_response_dub_stream] Called with avatar_id=8c9cd04b-bb63-4680-b9e1-a1d7c03847fd, chat_id=859ac5a0-7719-4526-9879-b3e18bc0e717, rsp_msg_id=537aecb6-50c5-43f8-a40d-f85b7caf1792, user_id=d62a88bb-6d7d-48fe-8508-f6f26ace034d
2025-05-26 17:47:34 - backend_logger - INFO - [get_avatar_response_dub_stream] Fetched response message: dub_url=None dub_status='pending' chat_id=UUID('859ac5a0-7719-4526-9879-b3e18bc0e717') user_id=UUID('d62a88bb-6d7d-48fe-8508-f6f26ace034d') is_generated=True text_status='ready' text='Hello my friend. I am not a robot. I am a human being. At least I think so.' id=UUID('537aecb6-50c5-43f8-a40d-f85b7caf1792') avatar_id=UUID('8c9cd04b-bb63-4680-b9e1-a1d7c03847fd')
2025-05-26 17:47:34 - backend_logger - INFO - [get_avatar_response_dub_stream] Dub is pending. Sending message to broker for generation. Message ID: 537aecb6-50c5-43f8-a40d-f85b7caf1792
2025-05-26 17:47:34 - backend_logger - INFO - Sending generate_dub message: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'base_voice_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
Topic sound-inference is not available during auto-create initialization
2025-05-26 17:47:34 - backend_logger - INFO - Sent message to topic 'sound-inference': {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'base_voice_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
2025-05-26 17:47:34 - backend_logger - INFO - [get_avatar_response_dub_stream] Sent dub generation request to Kafka. Updating message status to 'processing'.
INFO: 172.18.0.2:56920 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 404 Not Found
INFO: 172.18.0.2:56920 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e717 msgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/ HTTP/1.1" 200 OK
[0] 1:server 2:frontend 3:backend* 4:llm 5:sound 6:test docker logs bishop-b " 20:49 26-May-25

```

Рисунок 32 — Обработка сгенерированного текста и подготовка к озвучке

```

2025-05-26 17:47:38 - ml_logger - INFO - Received task from sound-inference: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'base_voice_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO:ml_logger:Received task from sound-inference: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO:ml_logger:Execute task with task_data: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'base_voice_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO:ml_logger:Execute task with task_data: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
2025-05-26 17:47:38 - ml_logger - INFO - Execute task with task_data: {'event': 'sound_inference', 'storage_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'base_voice_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO:ml_logger:[init] Loading Coqui TTS model...
INFO:ml_logger:[init] Loading Coqui TTS model...
100% [██████████] 1.87G/1.87G [03:28:00:00, 8.95MiB/s]
100% 4.37K/4.37K [00:00:00:00, 40.0KiB/s]
100% 361K/361K [00:00:00:00, 780KiB/s]
100% 32.0/32.0 [00:00:00:00, 47.6iB/s]
100% 7.75M/7.75M [00:20:00:00, 10.9MiB/s]2025-05-26 17:51:29 - ml_logger - INFO - [init] Coqui TTS model loaded successfully.
INFO:ml_logger:[init] Coqui TTS model loaded successfully.
2025-05-26 17:51:29 - ml_logger - INFO - [process_inference] Starting task 537aecb6-50c5-43f8-a40d-f85b7caf1792 with text length 75
INFO:ml_logger:[process_inference] Starting task 537aecb6-50c5-43f8-a40d-f85b7caf1792 with text length 75
DEBUG:ml_logger:[process_inference] Storage URL: users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792
Base voice URL: users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav
DEBUG:ml_logger:[wrapper] Initialized BytesIOwrapper
DEBUG:ml_logger:[download_minio] Starting download for 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'
DEBUG:ml_logger:[download_minio] Read 144460 bytes from MinIO for 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav'
2025-05-26 17:51:29 - ml_logger - INFO - [download_minio] Downloaded 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/audio/4423036e-65d8-4989-b820-772c77578344.wav' to '/tmp/tmp64lg234k.wav'
DEBUG:ml_logger:[process_inference] Using custom voice from '/tmp/tmp64lg234k.wav'
K4|1748281958.664|MAX_POLL_ID:kafka#consumer-2| [thrd:main]: Application maximum poll interval (300000ms) exceeded by 92ms (adjust max.poll.interval.ms for long-running message processing); leaving group
[4] 1:server 2:frontend 3:backend 4:lib 5:sound 6:test docker logs bishop-s 20:53 26-May-25

```

Рисунок 33 — Генерация аудиофайла на основе текста в сервисе озвучивания

```

INFO: 172.18.0.2:54340 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 404 Not Found
INFO: 172.18.0.2:43160 - "POST /api/v1/login/test-token HTTP/1.1" 200 OK
INFO: 172.18.0.2:43160 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 200 OK
2025-05-26 17:55:48 - backend_logger - INFO - [get_avatar_response_dub_stream] Called with avatar_id=8c9cd04b-bb63-4680-b9e1-a1d7c03847fd, chat_id=859ac5a0-7719-4526-9879-b3e18bc0e777/rsp_msg_id=537aecb6-50c5-43f8-a40d-f85b7caf1792, user_id=d62a88bb-6d7d-48fe-8508-f6f26ace034d
2025-05-26 17:55:48 - backend_logger - INFO - [get_avatar_response_dub_stream] Fetched response message: dub_url=None dub_status='processing' chat_id=UUID('859ac5a0-7719-4526-9879-b3e18bc0e777') user_id=UUID('d62a88bb-6d7d-48fe-8508-f6f26ace034d') is_generated=True text_status='ready' (text='Hello my friend. I am not a robot. I am a human being. At least I think so.') id=UUID('537aecb6-50c5-43f8-a40d-f85b7caf1792')
2025-05-26 17:55:48 - backend_logger - INFO - [get_avatar_response_dub_stream] Dub is currently processing. Message ID: 537aecb6-50c5-43f8-a40d-f85b7caf1792
INFO: 172.18.0.2:43160 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 404 Not Found
INFO: 172.18.0.2:43160 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 200 OK
2025-05-26 17:55:50 - backend_logger - INFO - [get_avatar_response_dub_stream] Called with avatar_id=8c9cd04b-bb63-4680-b9e1-a1d7c03847fd, chat_id=859ac5a0-7719-4526-9879-b3e18bc0e777/rsp_msg_id=537aecb6-50c5-43f8-a40d-f85b7caf1792, user_id=d62a88bb-6d7d-48fe-8508-f6f26ace034d
2025-05-26 17:55:50 - backend_logger - INFO - [get_avatar_response_dub_stream] Fetched response message: dub_url=None dub_status='processing' chat_id=UUID('859ac5a0-7719-4526-9879-b3e18bc0e777') user_id=UUID('d62a88bb-6d7d-48fe-8508-f6f26ace034d') is_generated=True text_status='ready' (text='Hello my friend. I am not a robot. I am a human being. At least I think so.') id=UUID('537aecb6-50c5-43f8-a40d-f85b7caf1792')
2025-05-26 17:55:50 - backend_logger - INFO - [get_avatar_response_dub_stream] Dub is currently processing. Message ID: 537aecb6-50c5-43f8-a40d-f85b7caf1792
INFO: 172.18.0.2:43160 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 404 Not Found
2025-05-26 17:55:51 - backend_logger - INFO - Received task from save-response-dub: {'event': 'save_response_dub', 'message_id': '537aecb6-50c5-43f8-a40d-f85b7caf1792', 'dub_url': 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792', 'generated_text': 'Hello my friend. I am not a robot. I am a human being. At least I think so.'}
INFO: 172.18.0.2:43160 - "GET /api/v1/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/chat/859ac5a0-7719-4526-9879-b3e18bc0e777/mgs/537aecb6-50c5-43f8-a40d-f85b7caf1792/response/dub/ HTTP/1.1" 200 OK
2025-05-26 17:55:52 - backend_logger - INFO - [get_avatar_response_dub_stream] Called with avatar_id=8c9cd04b-bb63-4680-b9e1-a1d7c03847fd, chat_id=859ac5a0-7719-4526-9879-b3e18bc0e777/rsp_msg_id=537aecb6-50c5-43f8-a40d-f85b7caf1792, user_id=d62a88bb-6d7d-48fe-8508-f6f26ace034d
2025-05-26 17:55:52 - backend_logger - INFO - [get_avatar_response_dub_stream] Fetched response message: dub_url='users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792' user_id=UUID('859ac5a0-7719-4526-9879-b3e18bc0e777') id=UUID('537aecb6-50c5-43f8-a40d-f85b7caf1792')
2025-05-26 17:55:52 - backend_logger - INFO - [get_avatar_response_dub_stream] MinIO object key: users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792
2025-05-26 17:55:52 - backend_logger - INFO - [get_avatar_response_dub_stream] Attempting to fetch object 'users/d62a88bb-6d7d-48fe-8508-f6f26ace034d/avatars/8c9cd04b-bb63-4680-b9e1-a1d7c03847fd/messages/537aecb6-50c5-43f8-a40d-f85b7caf1792' from MinIO bucket 'app'
2025-05-26 17:55:52 - backend_logger - INFO - [get_avatar_response_dub_stream] Audio object retrieved successfully for message ID: 537aecb6-50c5-43f8-a40d-f85b7caf1792
INFO: 127.0.0.1:39878 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:56474 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
INFO: 127.0.0.1:45236 - "GET /api/v1/utils/health-check/ HTTP/1.1" 200 OK
[4] 1:server 2:frontend 3:backend 4:lib 5:sound 6:test docker logs bishop-s 20:57 26-May-25

```

Рисунок 34 — Завершение ответа: возврат текста и аудио пользователю