Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ
Школа профессионального и академического образования

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

РОП И.Н. Обабков

(подпись)

«29» мая 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА DISCORD-БОТА ДЛЯ ВОСПРОИЗВЕДЕНИЯ МУЗЫКИ С ПОДДЕРЖКОЙ ВЕБ-ИНТЕРФЕЙСА. ПОЛЬЗОВАТЕЛЬСКАЯ ЧАСТЬ

Научный руководитель: Юманова Ирина Фарисовна подпись
к.ф.-м.н., доцент
Нормоконтролер: Егор Владимирович Огуренко подпись
Студент группы РИМ-210990 Гаев Михаил Алексеевич подпись

Екатеринбург
2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий-РТФ
Школа профессионального и академического образования
Направление подготовки 09.04.04 Программная инженерия
Образовательная программа «Разработка и управление в программных проектах»

УТВЕРЖДАЮ
РОП _____
«29» мая 2023 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студента *Гаева Михаила Алексеевича* **группы** *РИМ-210990*
(фамилия, имя, отчество)

1. Тема выпускной квалификационной работы *Разработка Discord-бота для воспроизведения музыки с поддержкой веб-интерфейса. Пользовательская часть*
Утверждена распоряжением по институту от «____» _____ 2023 г. № ___

2. Руководитель *Юманова Ирина Фарисовна к.ф.-м.н., доцент*
(Ф.И.О., должность, ученое звание, ученая степень)

3. Исходные данные к работе *материалы, полученные в ходе преддипломной практики, техническая литература, техническая документация*

4. Перечень демонстрационных материалов *Презентация*

5. Календарный план

№ п/п	Наименование этапов выполнения работы	Срок выполнения этапов работы	Отметка о выполнении
1.	*1 раздел (Анализ предметной области)*	до 15 апреля 2023 г.	Выполнено
2.	*2 раздел (Выбор и описание стека технологий пользовательской части)*	до 5 мая 2023 г.	Выполнено
3.	*3 раздел (Описание и принцип работы клиентской части приложения)*	до 20 мая 2023 г.	Выполнено
4.	*4 раздел (Перспективы, дальнейшая разработка и экономика проекта)*	до 28 мая 2023 г.	Выполнено

Руководитель _____ Юманова И.Ф.
(подпись) Ф.И.О.

Задание принял к исполнению 01.04.2023г. _____
дата (подпись)

6. Выпускная квалификационная работа закончена «__» мая 2023г. считаю возможным допустить Гаева Михаила Алексеевича к защите выпускной квалификационной работы закончена в Государственной экзаменационной комиссии.

Руководитель _____ *Юманова Ирина Фарисовна*
(подпись) Ф.И.О.

7. Допустить Гаева Михаила Алексеевича к защите магистерской диссертации в Государственной экзаменационной комиссии.

РОП _____ *Обабков И.Н*
(подпись) Ф.И.О.

РЕФЕРАТ

Тема выпускной квалификационной работы: Разработка Discord-бота для воспроизведения музыки с поддержкой веб-интерфейса. Пользовательская часть.

В состав ВКР входят: пояснительная записка 56 с., 32 рис., 2 табл., 59 источников.

Ключевые слова: Discord-бот, музыкальный сервис, музыка, ReactJS, Mobx, MUI.

Цель работы – создание Web-приложения для взаимодействия с серверной частью, на которой запущен Discord-бот для прослушивания музыки из разных источников, включая российские музыкальные сервисы.

Объектом выпускной квалификационной работы является VoIP-мессенджер Discord.

Предметом выпускной квалификационной работы является Frontend часть Discord-бота для прослушивания музыки с разных источников с поддержкой web-интерфейса.

В работе рассматривается предметная область, аналоги. Производится оценка и выбор технологий, архитектуры приложения. Описывается процесс разработки, трудности и подходы к их решению. Описывается тестирование и перспективы развития.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И ОПРЕДЕЛЕНИЯ

Web-приложение – это программное обеспечение, с помощью которого пользователь может взаимодействовать с данными через браузер

Discord – VoIP-мессенджер, на базе которого создан разработанный бот

Гильдия – сервер с набором голосовых и текстовых каналов

API – Application Programming Interface

WebSocket – протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между клиентом и сервером, используя постоянное полнодуплексное соединение

Discord-бот – программное обеспечение, выполняющее автоматизацию пользовательских сценариев в контексте Discord

Прототипирование – быстрая «черновая» реализация базовой функциональности будущего продукта

PWA – прогрессивные веб-приложения, которые создаются с помощью веб-технологий и которые можно устанавливать и запускать на всех устройствах из одной базы кода

MVP – минимально жизнеспособный продукт, имеющий минимальный набор функционала

Виртуальный DOM – концепция программирования, где идеальное или «виртуальное» представление UI хранится в памяти и синхронизируется с «реальным» DOM

Бойлерплейт – нетворческий программный код, который программисту приходится писать вследствие требований языка программирования

Beta тестирование – эксплуатационное тестирование потенциальными и/или существующими клиентами/заказчиками на внешней стороне, никак не связанными с разработчиками

ВВЕДЕНИЕ

Актуальность темы исследования.

Интернет стал неотъемлемой частью нашей жизни. Каждый день появляются новые технологии и сервисы. С начала 2000-х начались разработки сервисов коммуникации через глобальную сеть. Одним из лидеров мире по коммуникациям является платформа Discord, имеющая обширный функционал и открытый API для сторонних разработчиков. Благодаря этим возможностям появилось множество open-source проектов. Например, в списке популярных публичных ботов в русскоговорящем сообществе насчитывается около 152 активных ботов [49]. Спецификация ботов делится на несколько направлений:

– модерация серверов (по данным листинга, самый популярный бот в этом разделе – Juniper Bot [13]);

– создание игровых активностей – Jeggy Bot [52];

– боты для прослушивания музыки в голосовом канале, воспроизводящие аудио потоки со сторонних сервисов в сети – VK music bot [23].

Со временем сфера чат-ботов Discord развивалась и росла, что и повлияло на самих ботов – они стали более автоматизированы и функциональнее.

К сожалению, по требованию видеохостинга YouTube самые популярные музыкальные боты (Groovy [33], Rhythm [41]) прекратили свою работу, что вызвало большой спрос на использование менее известных ботов для проигрывания музыки. Изначально музыкальные чат-боты (в том числе и прекратившие работу Groovy и Rhythm) используют только текстовые команды в пользовательском чате, например, команда для запуска проигрывания `/play [URL]`, прописывая которую, бот присоединяется в канал и запускает аудиопоток с переданным URL в качестве второго аргумента. Кроме того, эти боты используют зарубежные сервисы, которые на данный момент официально не доступны в Российской Федерации.

Анализ актуальности обусловлен развитием API Discord и Web-технологий. В диссертационной работе предлагается изменить устаревший подход к написанию текстовых команд в чат на Web-интерфейс, облегчив тем самым использование и пользовательский опыт. На начало разработки не было аналогов подобному решению, но в процессе реализации некоторые боты перешли на подобный подход использования, например Freadboat [8].

Гипотеза исследования: Реализация Web-приложения для взаимодействия, контролируемого использования, реактивности с Discord-ботом для прослушивания музыки и поддержкой нескольких источников и музыкальных ресурсов, в том числе российских (Yandex music [5], VK music [1], YouTube [27], Spotify [37], iTunes [55]), упростит работу и повысит удобство совместного прослушивания музыки в Discord.

Целью исследования является создание Web-приложения для взаимодействия с серверной частью, на которой запущен Discord-бот для прослушивания музыки из разных источников, включая российские музыкальные сервисы. Для достижения поставленной цели необходимо решить следующие задачи:

– анализ существующих Discord-ботов для сравнительной характеристики используемых технологий;

– выбор оптимальных средств разработки Web-приложения с учетом существующих критериев, описанных в разделе 1.3 магистерской диссертации;

– выбор подходящей архитектуры приложения для упрощения масштабирования с учетом доступных ресурсов разработки;

– разработка Web-приложения для отображения состояния проигрывания, управления проигрыванием;

– получение, удаление, создание очередей воспроизведения через интерфейс;

– создание, удаление, обновление плейлистов пользователей;

– устранение ошибок, выявление уязвимостей, формирование беклога задач;

– выпуск бота и дальнейший сбор обратной связи и информации.

Объектом исследования является VoIP-мессенджер Discord.

Предметом исследования является Frontend часть Discord-бота для прослушивания музыки с разных источников с поддержкой Web-интерфейса.

Методы исследования включают в себя:

– анализ, сравнение, систематизация и обобщение данных о существующих и разработанных Discord-ботах с Web-интерфейсом;

– апробация Web, CI/CD технологий при построении Web-приложения;

– тестирование работы бота на ПК и мобильных устройствах с установленным на них приложением Discord.

– Теоретической основой исследования стали:

– популярные Discord-боты с функцией воспроизведения музыки;

– современные концепции и технологии организации Web-приложений;

– документация к различным используемым в современном программировании фреймворков и библиотек для создания web-приложения, инструменты тестирования интерфейса и взаимодействий с сервером.

Научная новизна и теоретическая значимость исследования.

Работа открывает направление исследований в области развития современных информационных и Web-технологий, применения информационных и Web-технологий для улучшения использования чат-ботов, упрощающих и автоматизирующих части взаимодействия пользователей в пределах какого-либо сервиса. В ходе работы над проектом, зафиксирована и описана потребность пользователей в наличии Discord-бота для воспроизведения музыки из разных источников, особенно из российских музыкальных платформ. Проведена работа по интеграции русскоязычных аудио ресурсов.

Практическая значимость исследования. Проведен анализ существующих Discord-ботов для проигрывания музыки, создан Discord-бот,

программный код, обеспечивающий использование функционала бота через Web-интерфейс. Реализовано Web-приложение для взаимодействия с Discord-ботом.

Структура работы. Во введении обоснована актуальность выбора темы, поставлены цель и задачи исследования, определен предмет и объект исследования, охарактеризованы методы исследования.

В первой главе описана и проанализирована предметная область, даны основные понятия и контекст выполнения работ.

Во второй главе приведено описание технологии проекта: анализ, выбор, разбор используемых инструментов разработки.

В третьей главе представлен процесс разработки, архитектура решения, тестирование.

В четвертой главе сформулированы направления развития, экономическая составляющая проекта.

В заключении сформулированы краткие выводы и рекомендации.

В работе имеется библиографический список, включающий 59 наименований.

1Анализ предметной области

Основным предметом диссертационной работы является VoIP мессенджер Discord. Discord – это бесплатный мессенджер, который позволяет обмениваться голосовым, видео и текстовым чатом с друзьями, игровыми сообществами и разработчиками. Функционал схож с аналогичными мессенджерами, например, Skype [50]. Отличительной чертой является наличие серверов, именуемых гильдиями. Пример интерфейса гильдии представлен на рисунке 1.

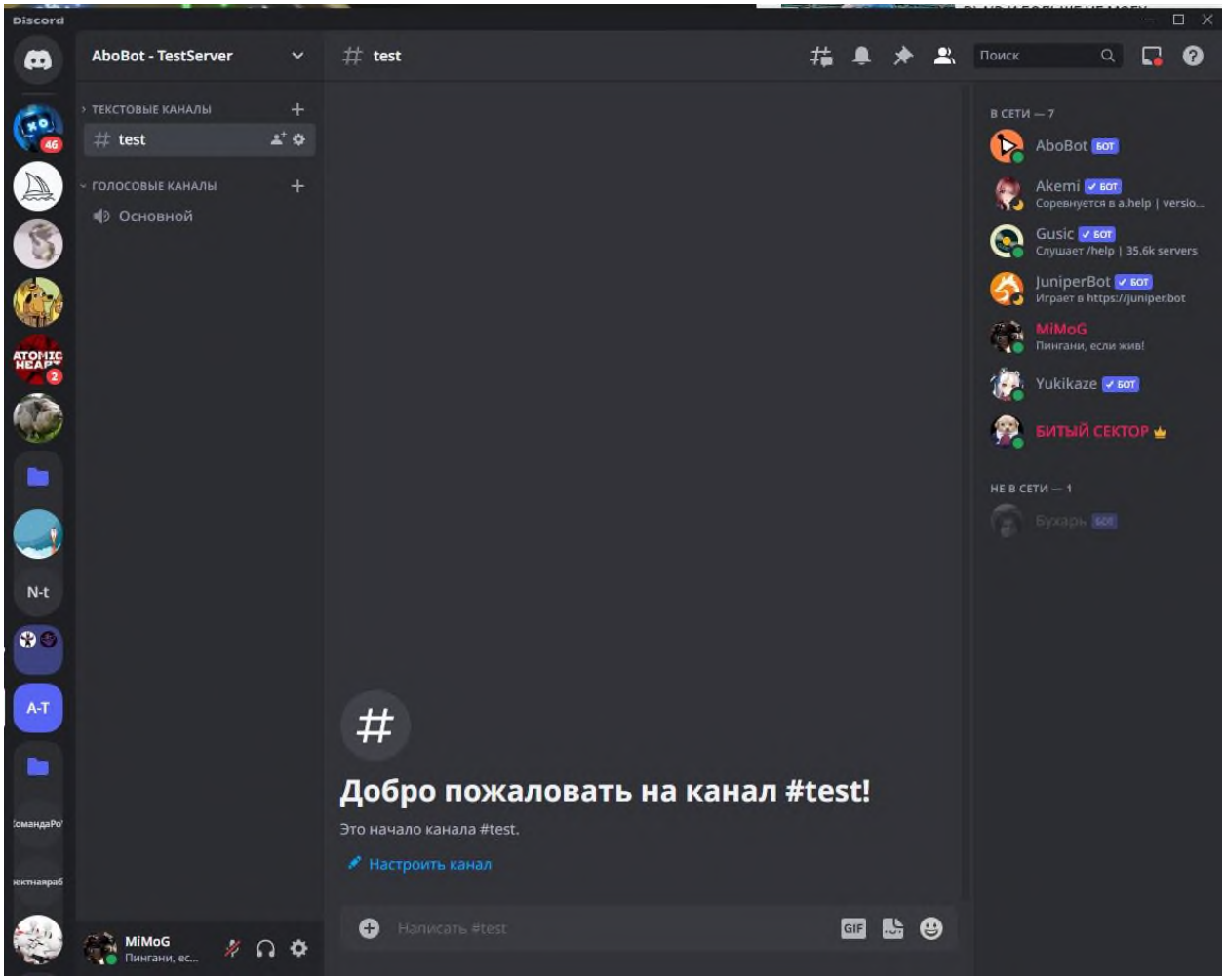

Рисунок 1 – Внешний вид гильдии в Discord

После релиза Discord, разработчики опубликовали открытый Discord API, используя который, появилась возможность создавать своих ботов и интеграции. Изначально управление ботами происходило посредством

текстовых команд (рисунок 2), но с развитием области программисты расширяли методы управления.

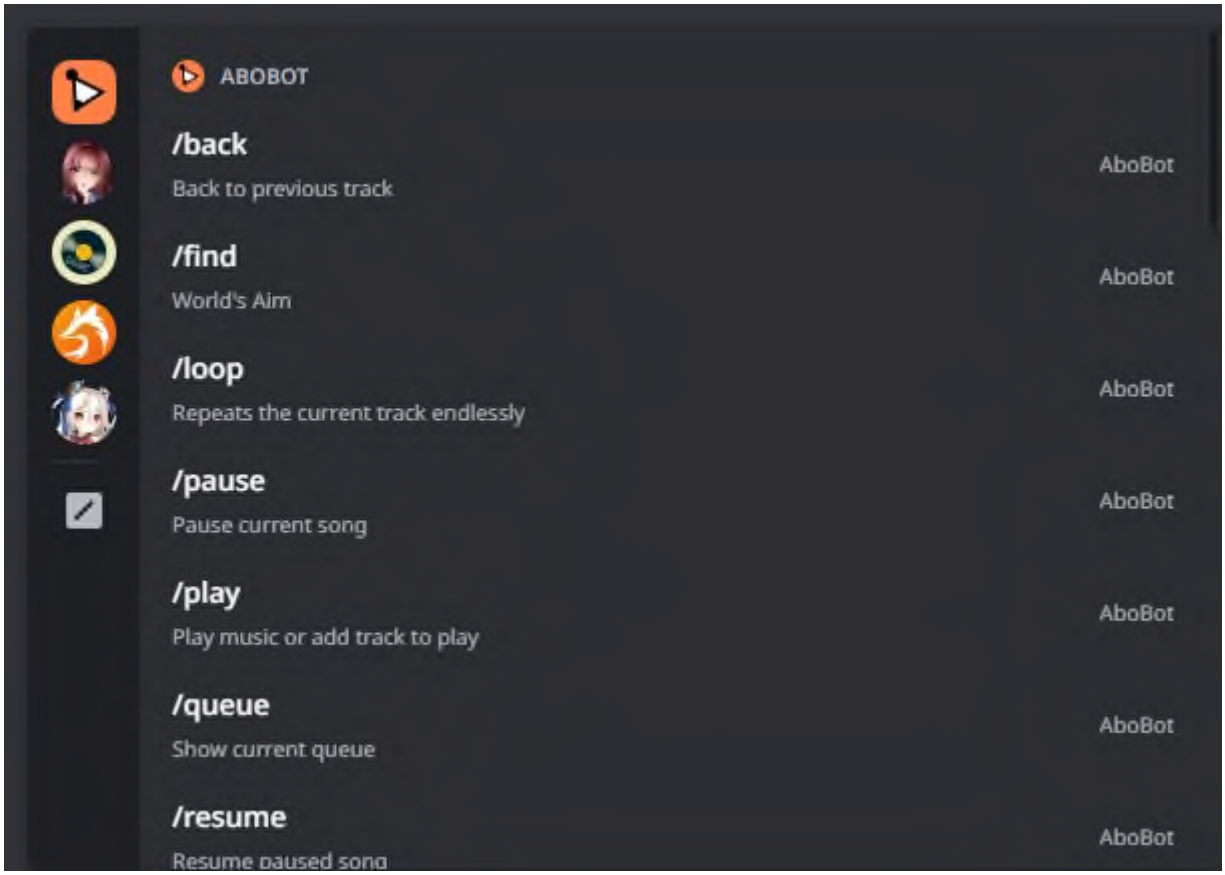

Рисунок 2 – Пример команд для управления ботами

Существует множество Discord-ботов для решения различных организаторских, развлекательных задач на сервере:

– модерирование сервера: удаление, отключение пользователей, использующих стоп-слова, спам, удаление сообщение, содержащих спам и запрещенные фразы, и даже приветствие новых участников сервера. Примеры: Mee6 [21], Dyno [20];

– запуск и прослушивание музыки, стримов с популярных сервисов, включая YouTube, Soundcloud и многих других. Примеры: Vk music bot [23], Akemi [51];

– развлекательные. Боты, которые установлены для игр или мероприятий в гильдии. Примеры: JeggyBot [52], MemeLand [14].

Многие популярные боты чаще всего совмещают в себе несколько функций. В работе будут рассмотрены боты со встроенным музыкальным плеером.

1.1 Обзор существующих музыкальных Discord-ботов

Музыкальные Discord-боты, преимущественно, представлены от зарубежных разработчиков, не каждый из них имеет поддержку русского языка. Также, в основном, все музыкальные боты используют только стандартный способ взаимодействия через текстовые команды для управления проигрыванием. Однако имеется представители и с поддержкой Web-интерфейса – Fredboat [2] и Yukikaze[7].

Разработанный бот – Abobot [6]. В числе первых ботов с собственным Web-приложением. Благодаря которому пользователю больше не требуется руками печатать текстовые команды в текстовом канале гильдии. Пользователю доступен полноценный Web-интерфейс о котором будет рассказано дальше.

На основании обзора существующих Discord-ботов и ботов с функцией проигрывания музыки была составлена сравнительная таблица, в которой в последней позицией представлен разрабатываемый бот Abobot.

Таблица 1 – сравнительная таблица музыкальных ботов

Название	Источники	Web-интерфейс	Premium версия	Требование подписки на музыкальные сервисы	Наличие плейлистов	Синхронизация прав доступа к Web-интерфейсу с Discord
FredBoat	Soundcloud, Deezer, Vimeo	+	-	-	Общие для одного сервера	+
Akemi	Яндекс Музыка, SoundCloud, Bandcamp	-	+	-	Общие для одного сервера. Доступны только с подпиской	-
Ear Tensifier	YouTube, SoundCloud, Spotify, Apple music	-	-	-	Несколько у каждогопользователя	-

Продолжение таблицы 1 – сравнительная таблица музыкальных ботов

Gusic	Apple Music, VK Music, SoundCloud, Spotify, Deezer, Яндекс Музыка	-	+	-	Общие для одного сервера. Доступны только с подпиской	-
Vk Music Bot	VK Музыка	-	+	+	Нет	-
Yukikaze	Vk Музыка, Яндекс Музыка, Spotify, SoundCloud	+	+	-	Несколько у каждого пользователя	-
Abobot	Youtube, Apple Music, Spotify, SoundCloud, Яндекс Музыка, VK Музыка	+	-	+	Несколько у каждого пользователя	+

На основании данных таблицы 1 можно сделать вывод, что на данный момент уже существуют Discord-боты и с Web-интерфейсом, и с поддержкой русскоязычных сервисов. Однако не каждый из них имеет достаточно простой и понятный Web-интерфейс (при его наличии), или малое количество функций помимо проигрывания и группового управления музыкой. Например, бот Yukikaze, в отличие от основных функций управления музыкой, не имеет разграничения прав управления между пользователями для Web-интерфейса. Иными словами, любой участник сервера, на котором стоит данный бот, может беспрепятственно управлять плеером на сайте, потенциально не имея на это прав.

Среди приведенных Discord-ботов с функцией проигрывания музыки в голосовой канал гильдии, самое большое количество источников музыки для воспроизведения (6 источников) у бота Gusic [10]. Однако это решение не имеет своего Web-интерфейса. Решение, описанное в данной магистерской диссертации, содержит такое же количество аудио ресурсов, но сделан акцент на использовании Web-приложения вместо команд.

1.2 Описание технологий аналогов

Все Discord-боты из сравнительной таблицы 1 созданы с помощью разных технологических и архитектурных решений. Для более корректного сравнения далее описаны боты, имеющие UI интерфейс:

- FredBoat – плеер создан на форке библиотеки lavaplayer [19] на языке Java [34]. Эта библиотека включает в себя несколько источников для поиска и воспроизведения музыки. Frontend сделан с помощью фреймворка Next [16] на языке JavaScript [48] (рисунок 3);
- Yukikaze – Frontend сделан с помощью фреймворка Vue [24]. Backend – с помощью Express (рисунок 4);

Для сообщений и обновлений состояний во всех случаях используется технология Web Socket [25]. Представленные боты с наличием управления из Web-интерфейса сделаны методом рендеринга на стороне сервера SSR [58].

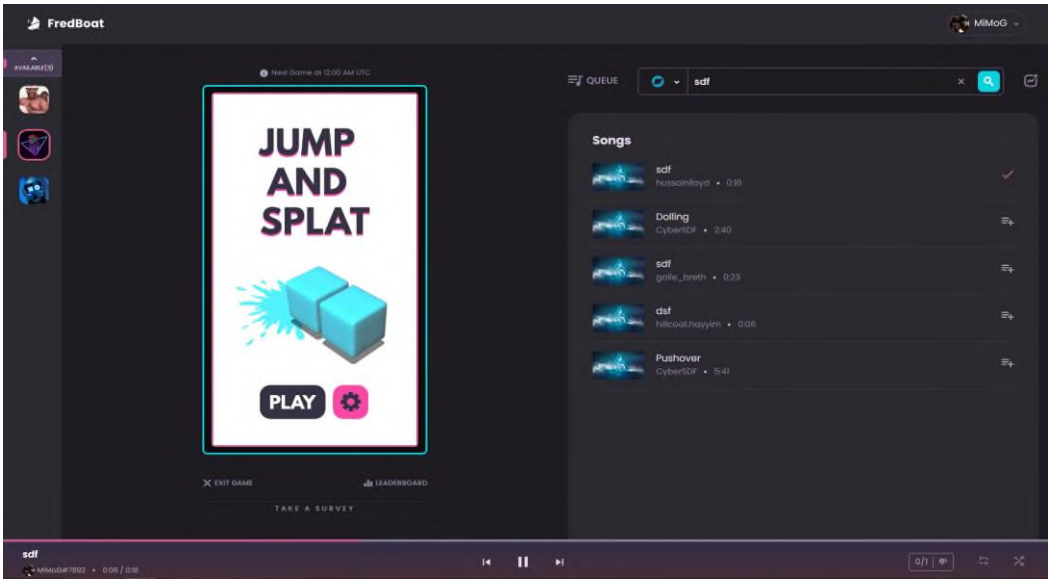

Рисунок 3 – Интерфейс плеера FreadBoat

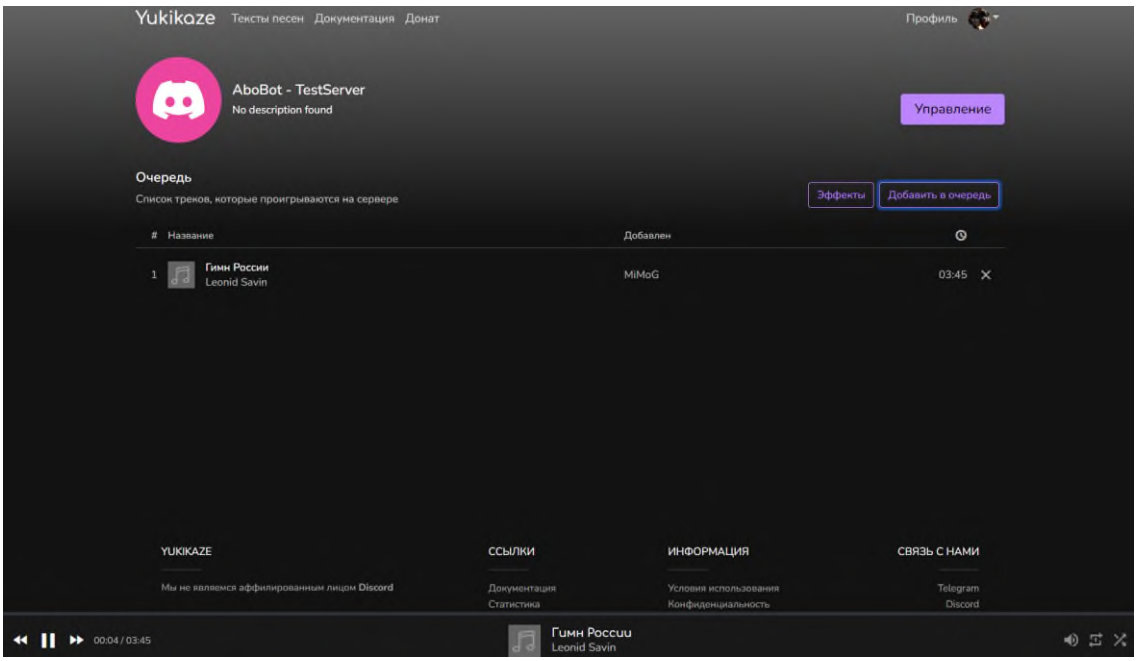

Рисунок 4 – Интерфейс плеера Yukikadze

1.3 Формулировка технических требований к приложению

Исходя из обзора аналогичных решений был сформирован список технических требований к web-приложению:

– круглосуточная доступность сервиса в сети;

– высокая отказоустойчивость;

– современный интерфейс;

– оптимизация под современные браузеры;

– адаптивная верстка для доступа с мобильных устройств;

– автоматическое сохранение данных при выходе пользователя.

Плеер Abobot выполнен с помощью discord-player.js, что включает в себя расширенную поддержку Яндекс Музыки и VK Музыки для библиотеки discord-player (дополнительные методы взаимодействия с аудио-ресурсами, изначально не заложенными в библиотеку). Frontend бота сделан с помощью фреймворка React и инструментов Mobx, MUI. Backend бота сделан с помощью фреймворка NestJS [15] (интерфейс и его разработка описаны в 3 главе).

Web-приложение сделано методом рендеринга на стороне клиента [58], по причине скорости разработки и дешевизне хостинга итогового приложения, а также снижения нагрузки на сервер, перемещая вычисления на сторону пользователя.

1.4 Выводы

С точки зрения удобства использования, разница всех представленных аналогах заключается в наличии Web-интерфейса. Среди же аналогов с функцией проигрывания музыки (воспроизведение, пауза, очередь, перемотка, переключение между треками), различия заключаются в количестве доступных источников для загрузки песен и дополнительных функций, сопутствующих проигрыванию, например разграничение прав доступа.

Техническая реализация аналогов различается в выборе стека технологий, библиотек и фреймворков использованных в разработке. Данные различия накладывают как технические ограничения, так и дополнительные возможности. При правильной приоритезации, оценке имеющихся средств и выборе технологий возможно создать оптимальное решение для поставленной задачи разработки. Подробное описание различие и преимущества технологий описано в разделе 2 магистерской диссертации.

2Выбор и описание стека технологий пользовательской части приложения

2.1 Сравнительный анализ производительности фреймворков

При подготовке к разработке Discord-бота Abobot был проведен анализ и поиск оптимального набора библиотек и фреймворков для реализации клиентского интерфейса приложения.

Фреймворк – программная платформа, в данном случае написанная на языке программирования (далее ЯП) JavaScript [48]. В переводе с английского языка фреймворк означает «каркас», что и является описанием функций, которые выполняет фреймворк. Он облегчает процесс разработки предоставляя более высокий уровень абстракций и конструкций ЯП.

Библиотека – сборник программ, написанных на определенном ЯП, содержит набор утилит для облегчения и структурирования процесса разработки.

Количество фреймворков увеличивается каждый день, но основной список самых популярных не меняется уже несколько лет. В таблице 2 представлены три наиболее популярных фреймворка.

Таблица 2 – сравнительная таблица популярных фреймворков

Наименование	Плюсы	Минусы
Angular	«Все в одном». Хранит в себе инструменты тестирования, разделение кода, шаблонизаторы, удобную работу с анимациями. имеет свою CLI и обширный API. Чаще используется для создания PWA приложений.	Возможны проблемы с производительностью, так как предоставляет для программиста высокий уровень абстракций. Для масштабных сценариев усложняется. Кривая обучения не линейна и «недружелюбна» к новичкам. Большое количество бойлерплейта.

Продолжение таблицы 2 – сравнительная таблица популярных фреймворков

React	Прост в освоении. Декларативный подход к программированию, что позитивно сказывается на читаемости кода. Обработка событий W3C. Свой синтаксис разметки JSX. Виртуальный DOM – ускоряющий взаимодействия с DOM деревом. Повторно используемые компоненты. Оптимизация – flux control. Активное сообщество разработчиков.	JSX – не так гибок, как разделение JS и HTM, может усложнить логику компонентов. Разработка React предполагает использование широкого спектра инструментов сборки для обеспечения хорошей работы и совместимости с другими инструментами веб-разработки. Скупая официальная документация.
Vue	Легковесный и быстрый. Обширная документация. Реактивная двусторонняя привязка. Виртуальный DOM. Простая работа с анимацией. Прост в освоении.	Сравнительно меньше доступных компонентов для использования ввиду относительной новизны. Маленькая экосистема плагинов и расширений. Отсутствие поддержки для масштабных проектов. Чрезмерно гибок.

На основании данных из таблицы 2 можно сделать вывод, что для текущих технических требований к разрабатываемому приложению лучше всего подходит фреймворк React. Помимо информации из таблицы, стоит обратить внимание на данные из сводных результатов быстродействия популярных фреймворков и связок с библиотеками в браузере Chrome [18] (рисунок 5).

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	vue-v3.2.47	angular-v15.0.1	react-v18.2.0
Implementation notes			
Implementation link	code	code	code
create rows creating 1,000 rows (5 warmup runs).	45.4 ± 0.7 (1.18)	48.3 ± 0.4 (1.26)	52.2 ± 0.9 (1.36)
replace all rows updating all 1,000 rows (5 warmup runs).	45.6 ± 0.6 (1.13)	51.9 ± 0.5 (1.29)	54.6 ± 1.0 (1.35)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	120.3 ± 3.8 (1.23)	110.8 ± 1.9 (1.14)	145.6 ± 3.8 (1.49)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	19.9 ± 1.0 (1.82)	16.6 ± 1.4 (1.51)	44.4 ± 1.5 (4.05)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	29.3 ± 0.3 (1.08)	171.0 ± 0.9 (6.30)	168.6 ± 1.4 (6.21)
remove row removing one row. (5 warmup runs). 4x CPU slowdown.	51.6 ± 0.8 (1.13)	47.9 ± 1.3 (1.05)	53.5 ± 1.1 (1.17)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	494.5 ± 4.3 (1.19)	497.2 ± 2.5 (1.19)	682.9 ± 3.5 (1.64)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	98.9 ± 0.8 (1.14)	106.9 ± 0.4 (1.23)	117.7 ± 0.6 (1.36)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	37.1 ± 1.4 (1.28)	69.4 ± 1.6 (2.39)	40.3 ± 1.0 (1.39)
geometric mean of all factors in the table	1.23	1.59	1.85
compare: Green means significantly faster, red significantly slower	compare	compare	compare

Рисунок 5 – Сводная характеристика выбранного фреймворка [18]

Выбранное решение довольно тяжело справляется с индексированием данных в больших таблицах в сравнении с Vue. Данный недостаток не повлияет на быстродействие и производительность работы приложения ввиду отсутствия больших списков данных. Порядок быстродействия, 168 миллисекунд не уловим человеческим глазом. Так же проанализирована работа фреймворка с метриками, полученными через инструмент Lighthouse [36] (рисунок 6).

Startup metrics (lighthouse with mobile simulation)

Name	vue-v3.2.47	angular-v15.0.1	react-v18.2.0
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,027.1 ± 49.2 (1.12)	2,780.3 ± 0.7 (1.54)	2,552.8 ± 50.0 (1.42)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	197.0 ± 0.0 (1.38)	282.8 ± 0.0 (1.99)	280.9 ± 0.0 (1.97)
geometric mean of all factors in the table	1.25	1.75	1.67

Рисунок 6 – Характеристики на основе метрик Lighthouse [18]

На рисунке представлены «чистые» результаты, без оптимизаций приложения. Явным лидером является фреймворк Vue, но ввиду новизны и малого количества адаптированных плагинов данный Фреймворк не подходит под технические требования, сформулированные в разделе 1.3 магистерской диссертации.

2.2 Технологический стек frontend-приложения

Для разработки приложения были выбран фреймворк React [40]. Для взаимодействия с состоянием приложения – Mobx [53]. Компоненты интерфейса – Material UI [38]. Разберем выбранный стек более подробно.

Основным фреймворком разрабатываемого приложения, является библиотека React. Начиная с 2015 года позиционируется как фреймворк для Web-разработки современных высокопроизводительных приложений любой сложности. Данный фреймворк в большей степени упрощает визуальную составляющую приложения. Фреймворк лишен встроенных функций, таких как общее хранилище состояния с инструментами управления, маршрутизации. Плюсом фреймворка является его простота. Поэтому разработка качественного и большого приложения требует от программиста знания и опыт для реализации.

В разрабатываемом Discord-боте библиотека Mobx используется для восполнения функционала, в качестве менеджера состояния приложения, изначально в приложении использовалась дочерняя библиотека Mobx-state-tree, от которой пришлось отказаться ввиду разницы их производительности. Mobx является более легковесной производительной библиотекой или набором утилит для управления состоянием приложения, включающая в себя принцип реактивности [28], написанная в декларативном стиле. Mobx-state-tree в свою очередь – полноценное решение, основанное на Mobx для простого и удобного изменения, отслеживания, удаления данных из хранилища. Недостаток заключается в размере и излишнем функционале, которые сказываются на скорости сборки и размере собранного приложения.

Для визуальной и функциональной составляющей Web-интерфейса используется библиотека готовых визуальных компонентов Material UI, включающая в себя: кнопки, поисковые строки, поля ввода и т. д. Это популярное поддерживаемое решение для быстрого и простого создания интерфейса любой сложности, с инструментами кастомизации. Внутри библиотеки уже реализована поддержка разных браузеров, чтобы во всех них был идентичный визуальный результат.

Для сборки и оптимизации приложения для пользователей используется сборщик приложения Webpack [43], оптимизирующий код под браузер. Разделяет код на модули, позволяя не загружать все сразу и целиком, что оптимизирует и ускоряет отзывчивость и производительность собранного приложения. Вместе с webpack используется дополнительный набор утилит и компилятор Babel [29] для совместимости с разными браузерами.

Для полнодуплексного соединения и обмена сообщений в реальном времени используется надстройка SocketIO [42] над технологией web-socket [25]. Реализована по средствам протокола web-socket и включает в себя утилиты для обмена информацией с сервером.

В качестве пользовательской маршрутизации приложения для перехода, изменения данных, между конечными точками используется Mobx-state-router

[32] является восполнением функционала маршрутизации приложения, соединенным с общим состоянием хранилища Mobx.

Для тестирования разработанного функционала и приложения целиком используется библиотека тестирования компонентов Jest [35] с упором на простоту, скорость и надежность. Для тестирования функционала отдельных компонентов созданы их «пародии», отвязанные от реального кода, с сохранением абстракции и внутренних функций.

2.3 Фреймворк управления и аппаратно-программный комплекс средств разработки

Процесс создания Web-приложения и сервера для Discord-бота Abobot разделен на 2 этапа. В первом этапе разработки от идеи до MVP был выбран подход Waterfall [1]. Подразумевается, что каскадная модель в начале жизненного цикла приложения Abobot подходит по существующим критериям:

– понятный план продукта;

– понятный перечень начального функционала;

– малый ресурс разработки (о котором подробнее в 4 главе).

Следующим этапом разработки приложения стала публикация и развертывание в общий доступ. На этом этапе было принято решение перейти к итеративному фреймворку – Scrum [2; 26; 45]. Пример спринтов приведен на рисунке 7.

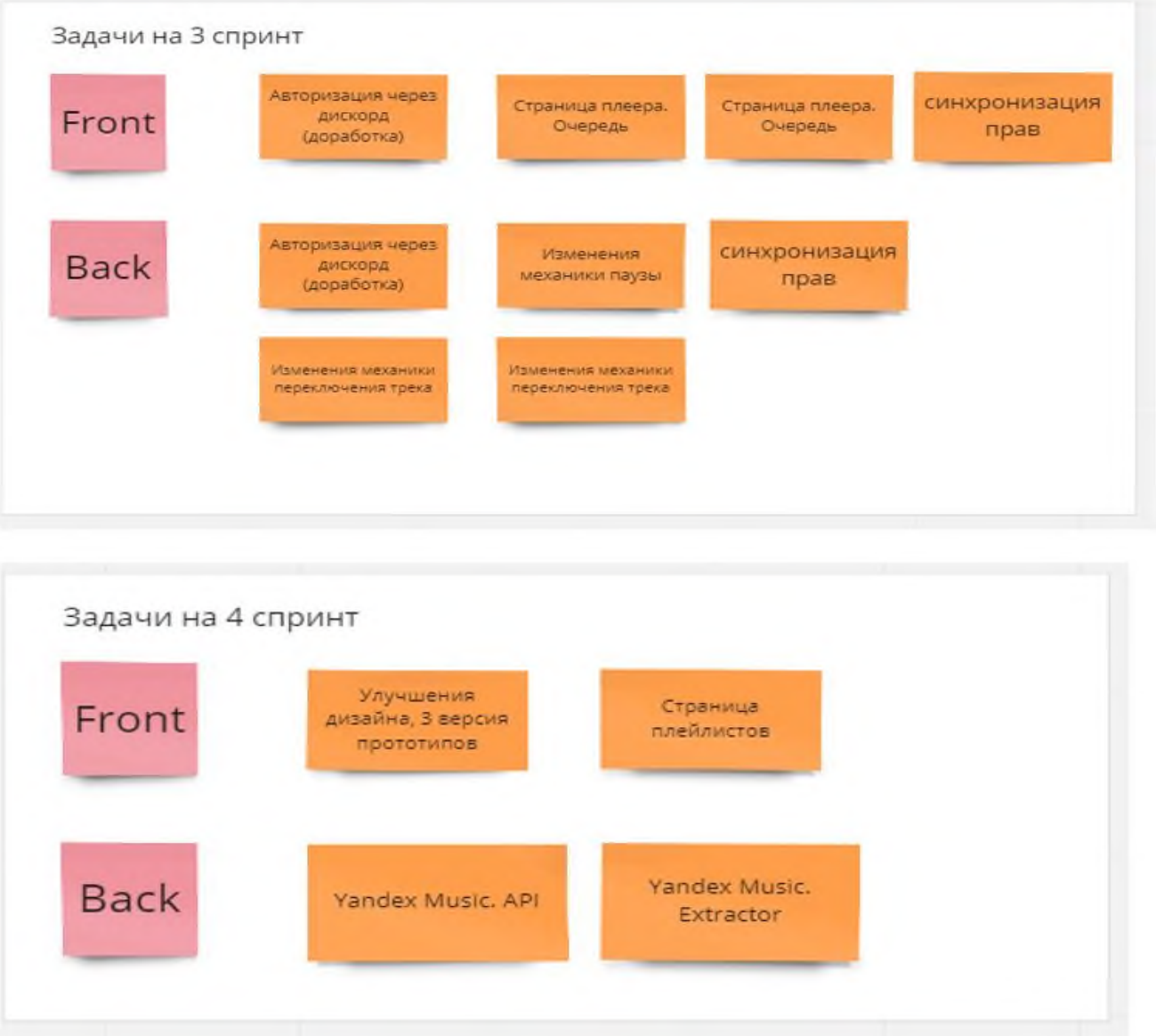

Рисунок 7 – Пример 3 и 4 спринтов

Временные рамки 2 этапа разработки с 01.03.2022 по 29.05.2023. Временной промежуток разбит на спринты, длящиеся 2 недели. Для каждого спринта выделялся пул неделимых задач (на рисунке 7 описаны краткие заголовки). По итогам спринта происходит тестирование и релиз нововведений. Ввиду малого количества ресурсов и небольшой аудитории (рисунок 8), проверка также происходит в формате Beta тестирования, на малом количестве пользователей.

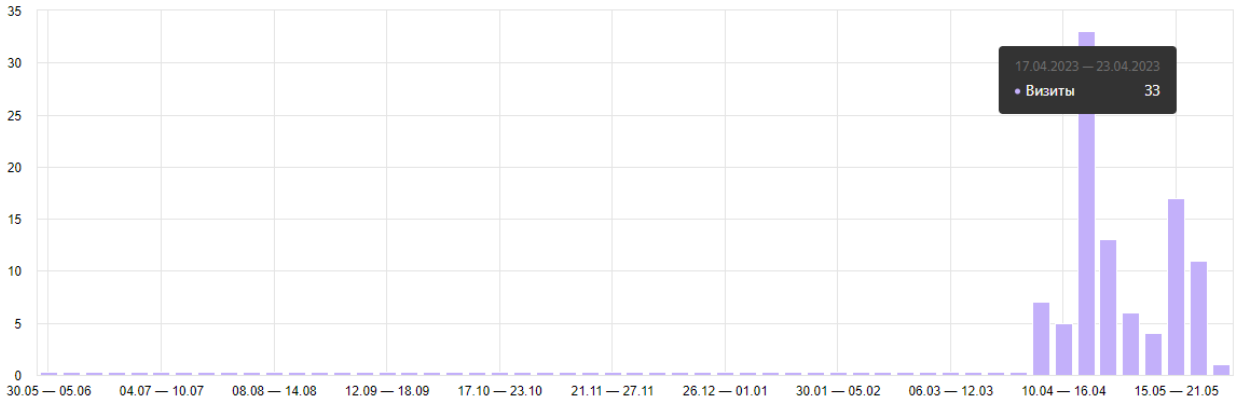

Рисунок 8 – Посещаемость abobot.ru

Программный комплекс для разработки приложения включает в себя несколько приложений. Первое приложение – IDE (integrated development environment) WebStorm [44]. Программное обеспечение WebStorm, обеспечивает автодополнение, анализ кода на лету, навигацию по коду, рефакторинг, отладку, и интеграцию с системами управления версиями.

Для тестирования взаимодействия с серверной частью приложения спользуется программа Postman [39], благодаря которой происходит отладка и тестирование REST запросов (рисунок 9).

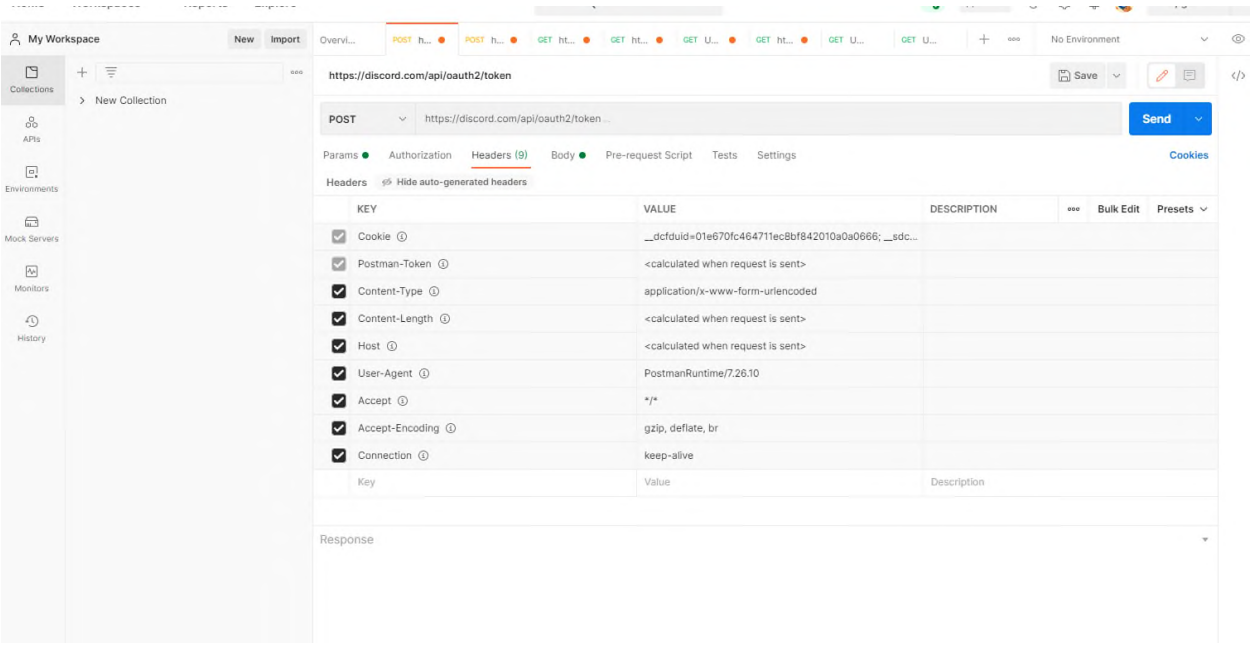

Рисунок 9 – Интерфейс приложения Postman

Приложение Postman очень простое в использовании и тестировании. Для проверки любого API необходимо создать запрос через конструктор запросов, Postman автоматически проставит необходимые заголовки для доступа к ресурсу. После создания запроса необходимо отправить его и получить ответ, проанализировав который, можно составить представление о возвращаемых данных и их структуре.

Для контроля версий и изменений приложения используется система контроля версий Git [9], позволяющая гибко и понятно сохранять изменения в кодовую базу проекта. Это необходимо для разработки проектов любого масштаба, так же для создания приложений совместно с несколькими программистами. Система контроля версий позволяет не потерять созданный код, структурировать его и соединять части проекта с общей версией master ветки приложения. Для работы с деревом Git используется программа Fork [7] (рисунок 10).

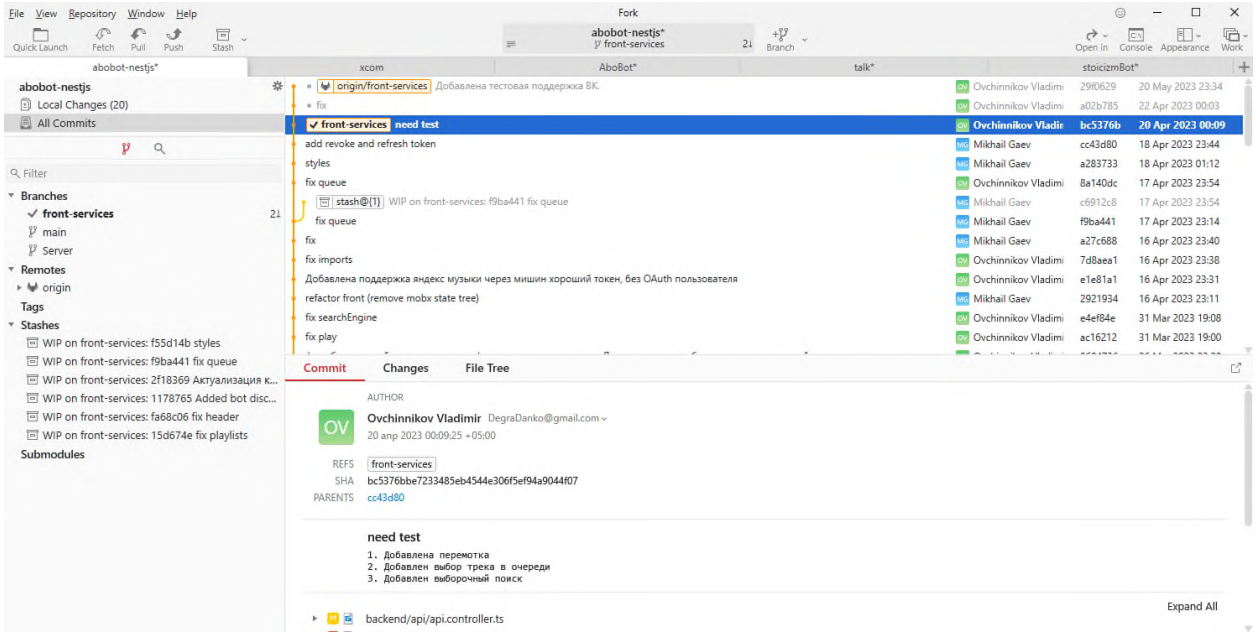

Рисунок 10 – Интерфейс программы Fork

Данная программа облегчает утилиты Git, превращая текстовые команды Git в удобный UI-интерфейс, который позволяет быстро и четко с визуальным представлением понять, что происходит с текущим деревом версий. В ней отображается автор коммита, время и изменения кода.

Система управления репозиториями – GitLab, являющаяся хранилищем проекта, включает набор побочных утилит для разработки приложений (рисунок 11).

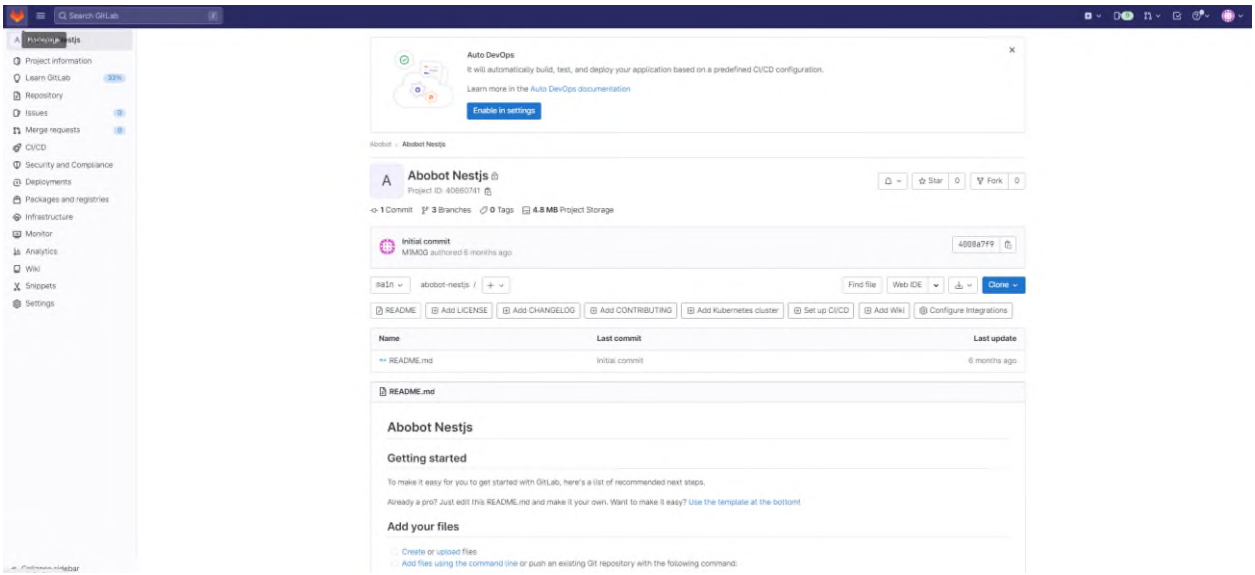

Рисунок 11 – Web-интерфейс GitLab

GitLab позволяет использовать простой и понятный CI/CD [4]. Проект не доступен в открытом доступе, имеет свой pipeline публикации, конфигурация развертывания представлена в приложении А, написанное в формате YAML [22]. Для выполнения написанной конфигурации на виртуальной машине развернута программа, работающая в фоновом режиме – gitlab-runner [47]. Интерфейс GitLab CI/CD представлен на рисунке 12.

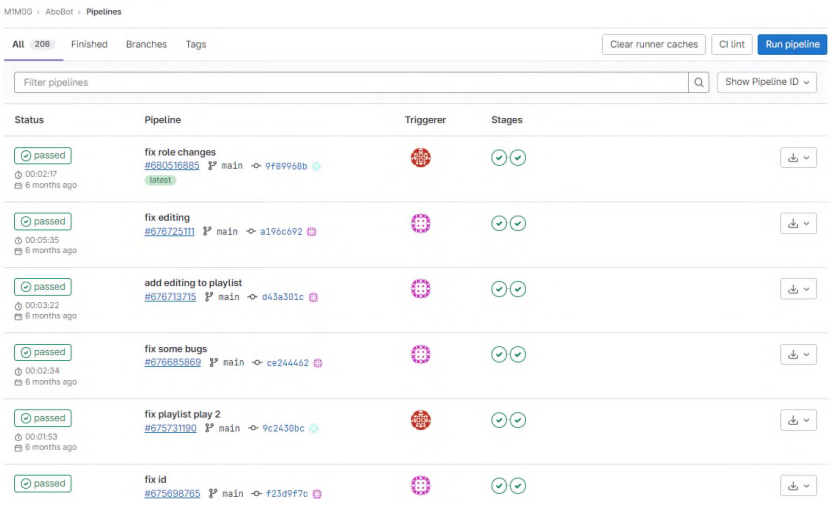

Рисунок 12 – Интерфейс GitLab CI/CD

2.4 Выводы

Используя все приведенные выше средства разработки, стек технологий, основой которого явдяется React и Mobx, позволяет создать простое масштабируемое, управляемое приложение любой сложности. Активное сообщество разработчиков всегда может помочь с возникшими проблемами и решить их в кратчайшие сроки или найти решение у других программистов, столкнувшихся с такими же проблемами.

Современная IDE Webshtorm предоставляет огромный набор плагинов и утилит, облегчающих отладку и разработку функций для приложений на ЯП JavaScript.

3Описание и принцип работы клиентской части приложения

3.1 Прототипирование

Начальная стадия разработки – разработка макетов будущего приложения. Прототипирование – сложный и долгий процесс, идущий либо до начала разработки фичи приложения, либо параллельно с ходом разработки. Прототипы непостоянны и меняются, даже самые большие компании и корпорации проводят ребрендинг. Web-интерфейс Abobot сменил 4 представления и 4 версии прототипов. Конечный результат был утвержден и радикально меняться не будет. Все представленные прототипы и логотип были разработаны в приложении Figma [31]. Которое является удобным и бесплатным для текущих целей инструментом прототипирования интерфейсов для ПО. Первая версия прототипов представлена на рисунке 13.

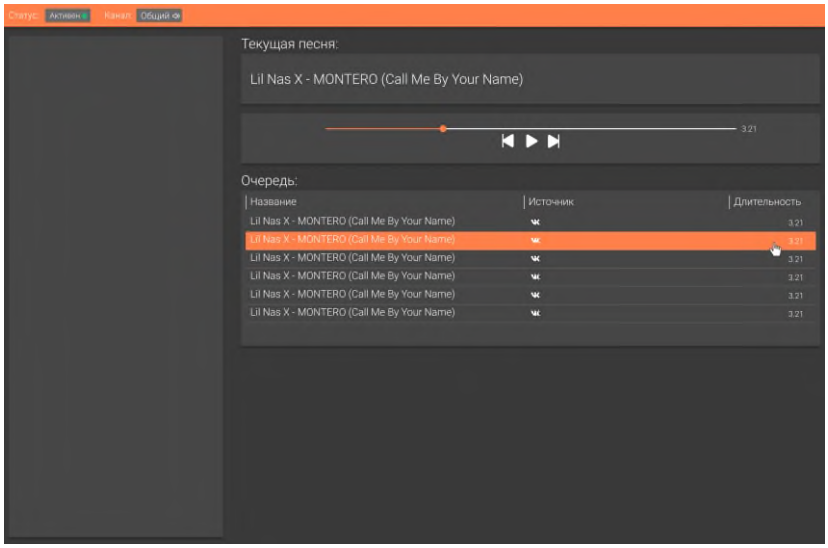

Рисунок 13 – Первая версия abobot.ru

Первое видение интерфейса для пользователя. Простота обеспечила быстрый анализ и «прикидку» функционала, не требуя дизайнерского мастерства.

Далее представлена вторая версия интерфейса, уже учитывая первые неудобства (рисунок 14).

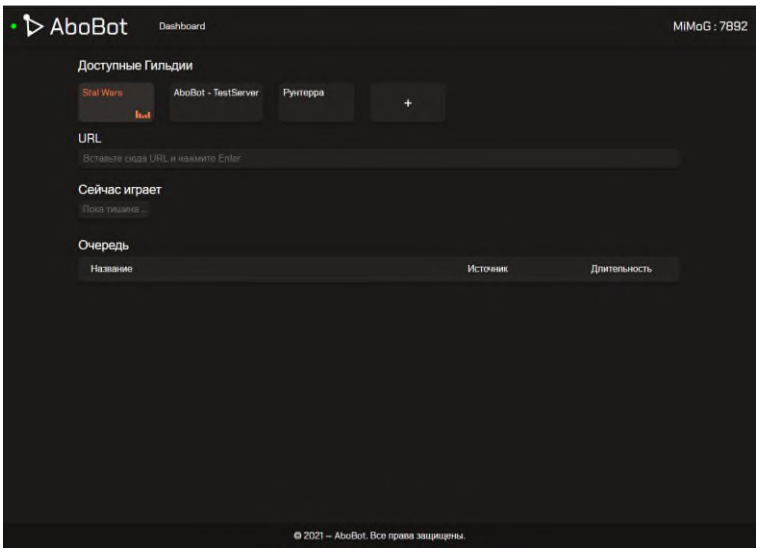

Рисунок 14 – Вторая версия abobot.ru

Во второй версии уже учтены недочеты и убраны лишние элементы, «утяжеляющие» нешний вид интерфейса. Так как интерфейс довольно темный (темная тема). В дополнение к текущему виду был добавлен светлый вариант (рисунок 15).

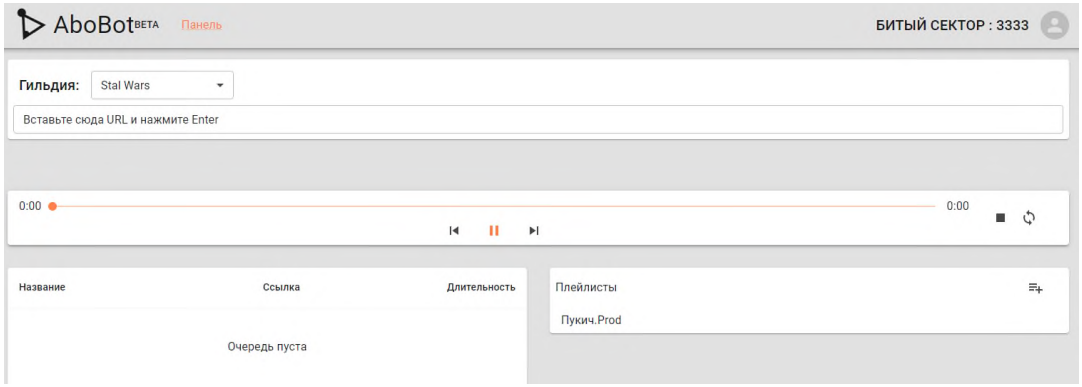

Рисунок 15 – Светлая тема abobot.ru

Третья версия также отличается от предыдущей, все элементы управления были сгруппированы ближе к друг другу, тем самым лаконично дополняя и упрощая интерфейс для пользователя. Но все же были еще лишние элементы, которые “загрязняли” вид и использовались редко в ключевых сценариях. Поэтому была создана четвертая и текущая версия интерфейса (рисунок 16, 17).

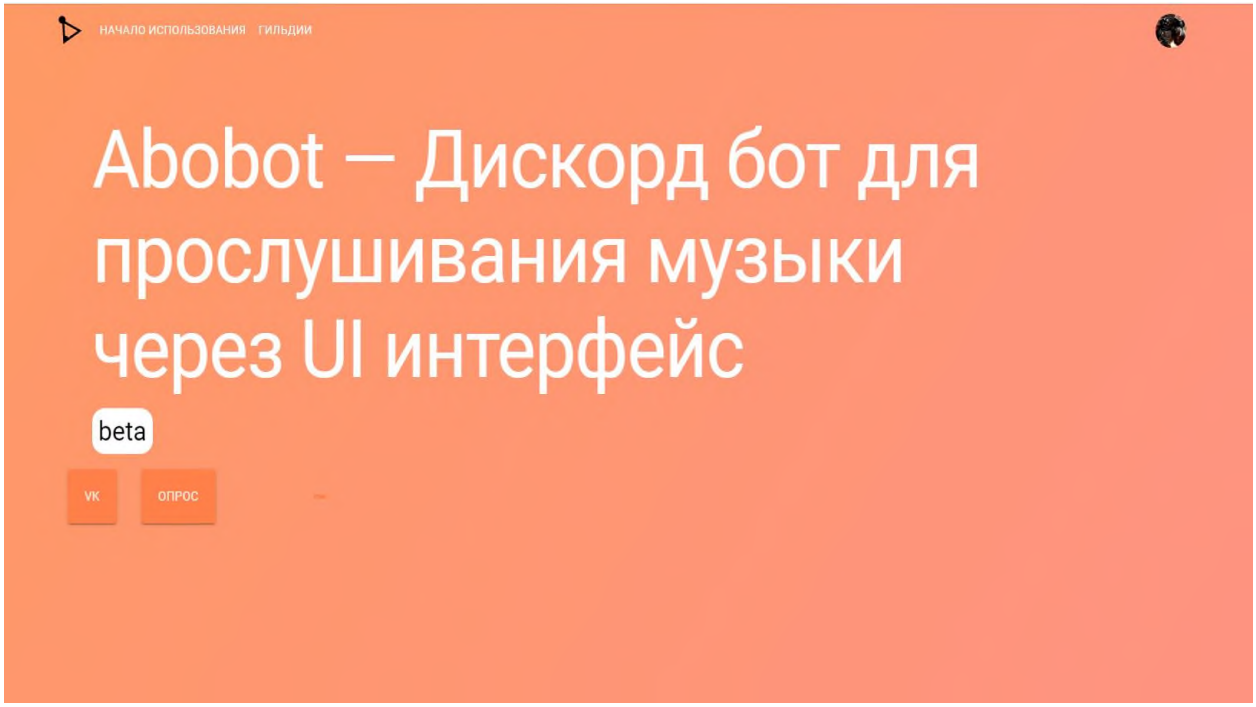

Рисунок 16 – Четвертая версия abobot.ru

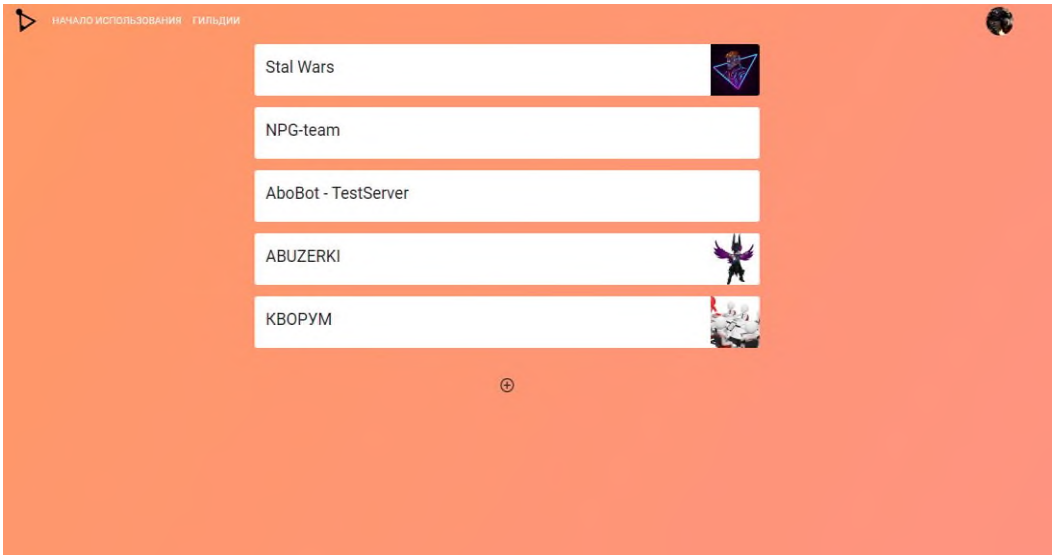

Рисунок 17 – Страница выбора гильдии

Интерфейс стал более красочным и ярким. Минимализм ориентирует на современность. Добавились ссылки на социальные сети и обратную связь.

Элементы разделились на свои конечные точки, выбор гильдии с помощью которого создавался плеер, в ключевом сценарии это происходит один раз, поэтому выполнено решение вынести этот функционал (рисунок 16). Добавилась инструкция использования и добавления бота в Discord.

Новый вид плеера был одобрен большинством пользователей приложения, результаты взяты из формы обратной связи, представленной на рисунке 18. Оценка дизайна по пятибалльной шкале, где 1 – ужасно, 5 – превосходно.

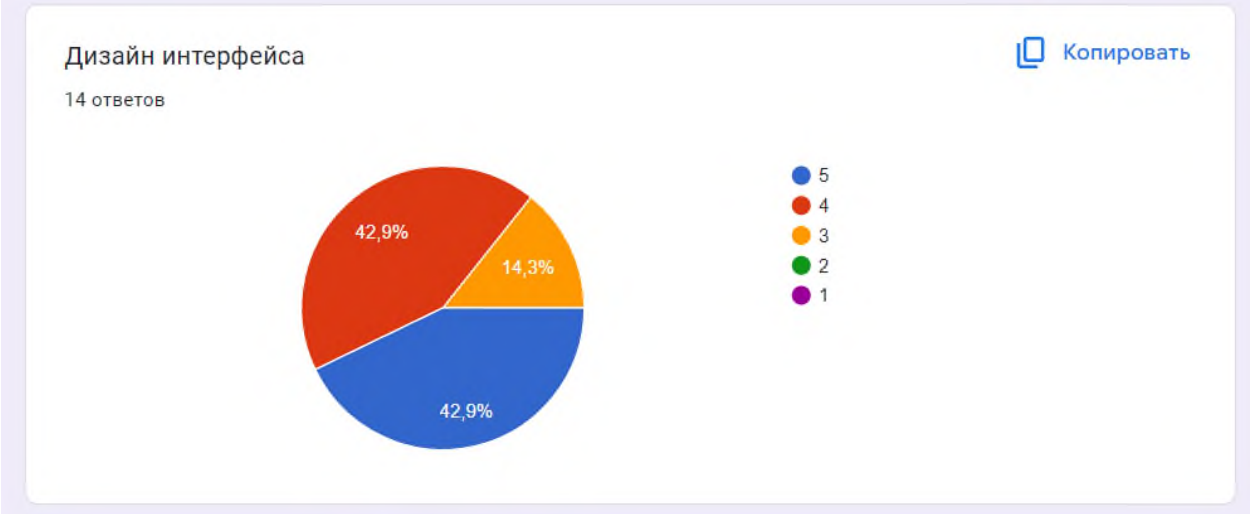

Рисунок 18 – Обратная связь от пользователей abobot.ru

Основываясь на ключевых сценариях для управления ботом из Web интерфейса, элементы были изменены и скомпонованы в текущее представление (рисунки 24, 25, 26). Визуально интерфейс напоминает популярные современные плееры. (Yandex music, VK music, SoundCloud , Spotify, iTunes) В последней итерации прототипов были выбраны референсы (рисунки 19, 20, 21, 22), из которых были переняты решения по расположению и функционалу, что дает похожий UX для пользователей и позволяет не нагружать сервис лишней информацией.

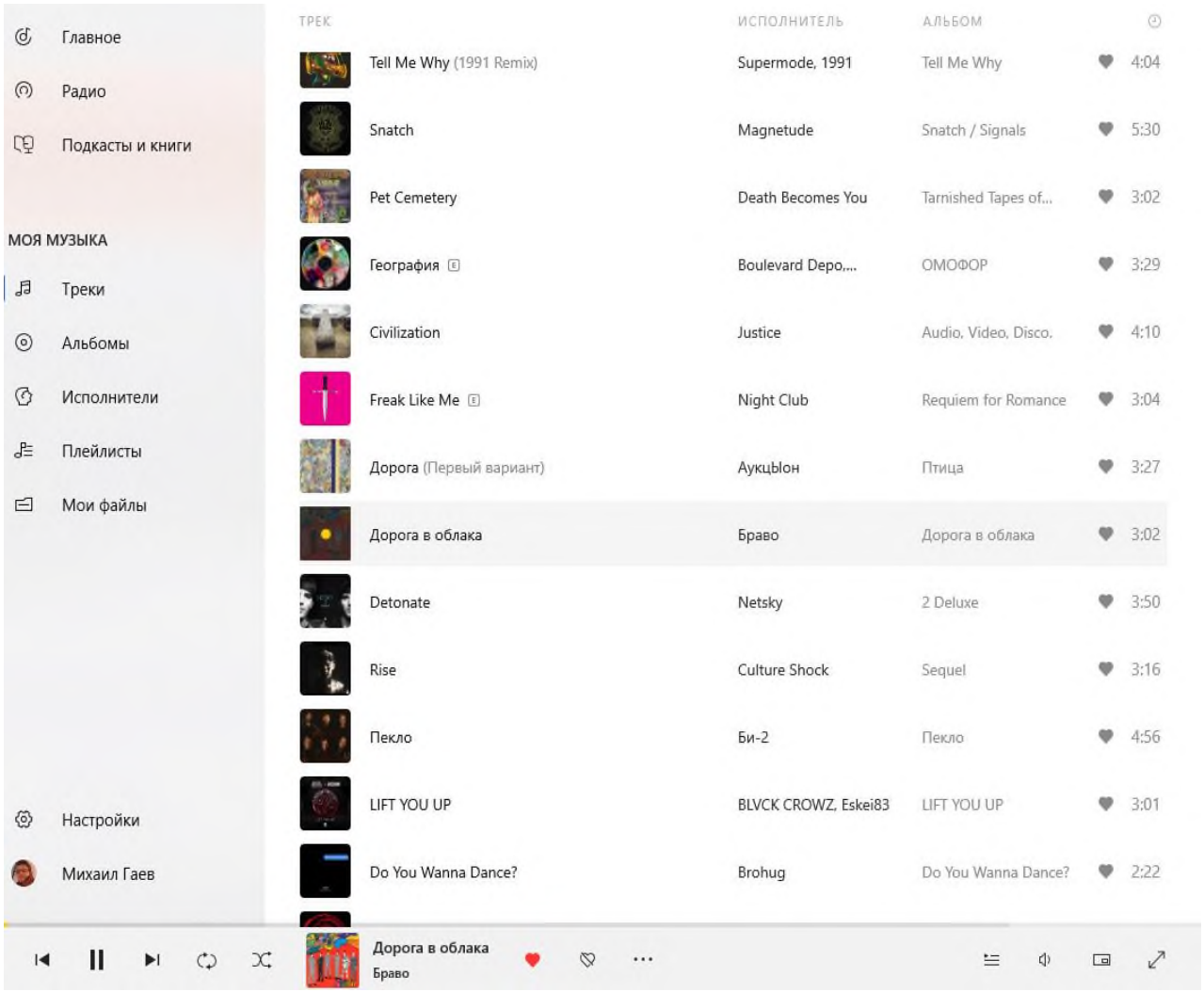

Рисунок 19 – Референсы Yandex Music

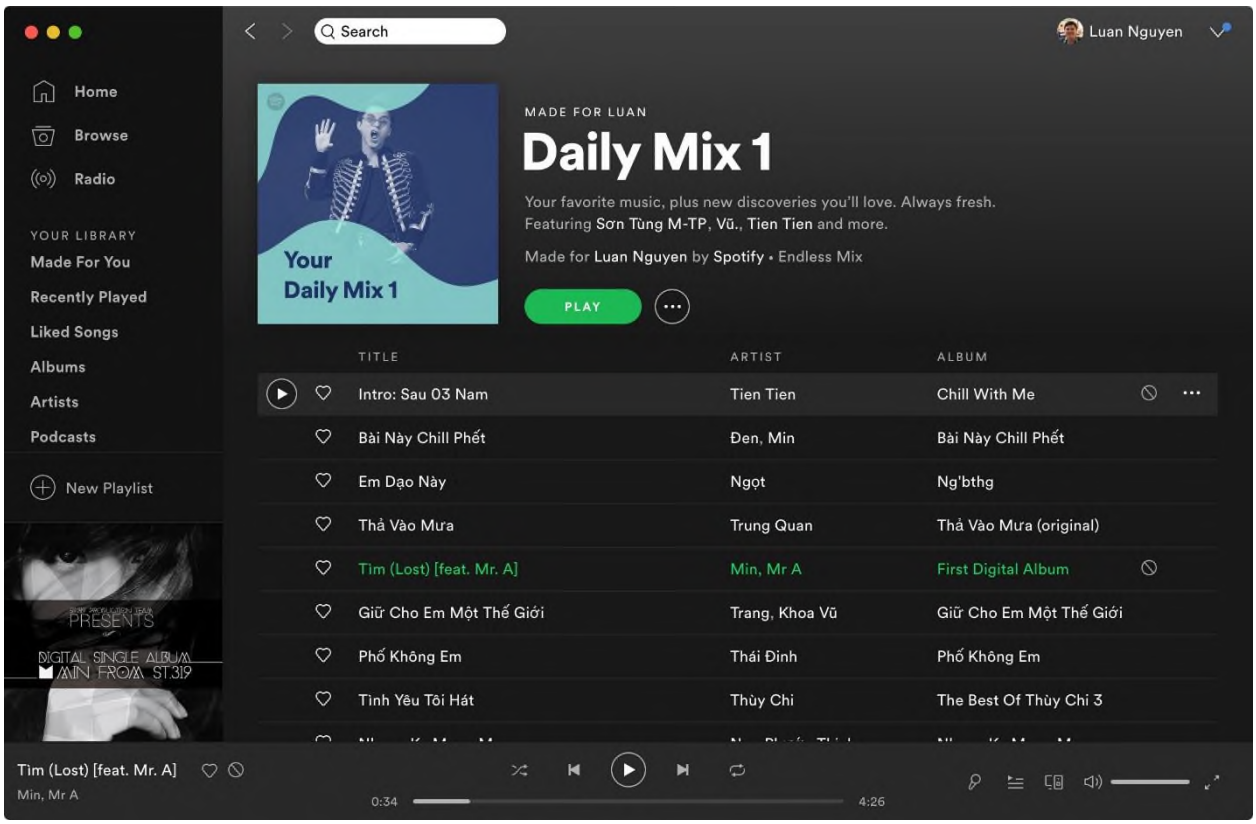

Рисунок 20 – Референсы Spotify

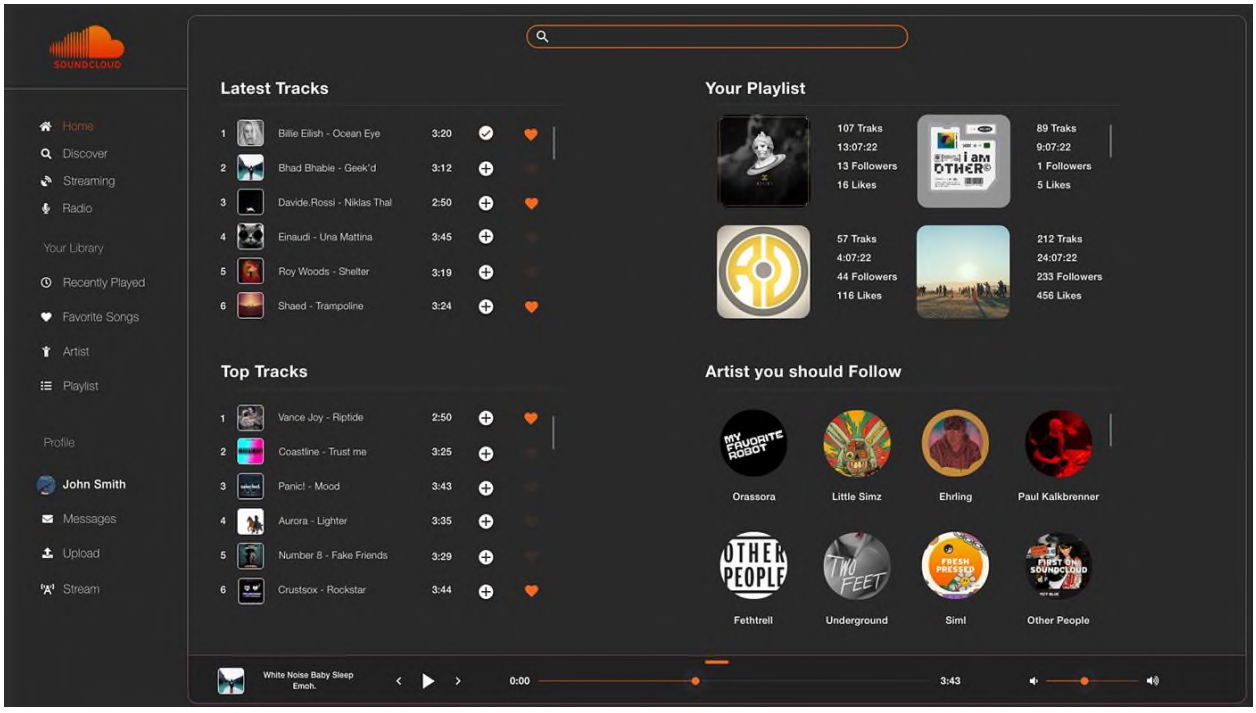

Рисунок 21 – Референсы SoundCloud

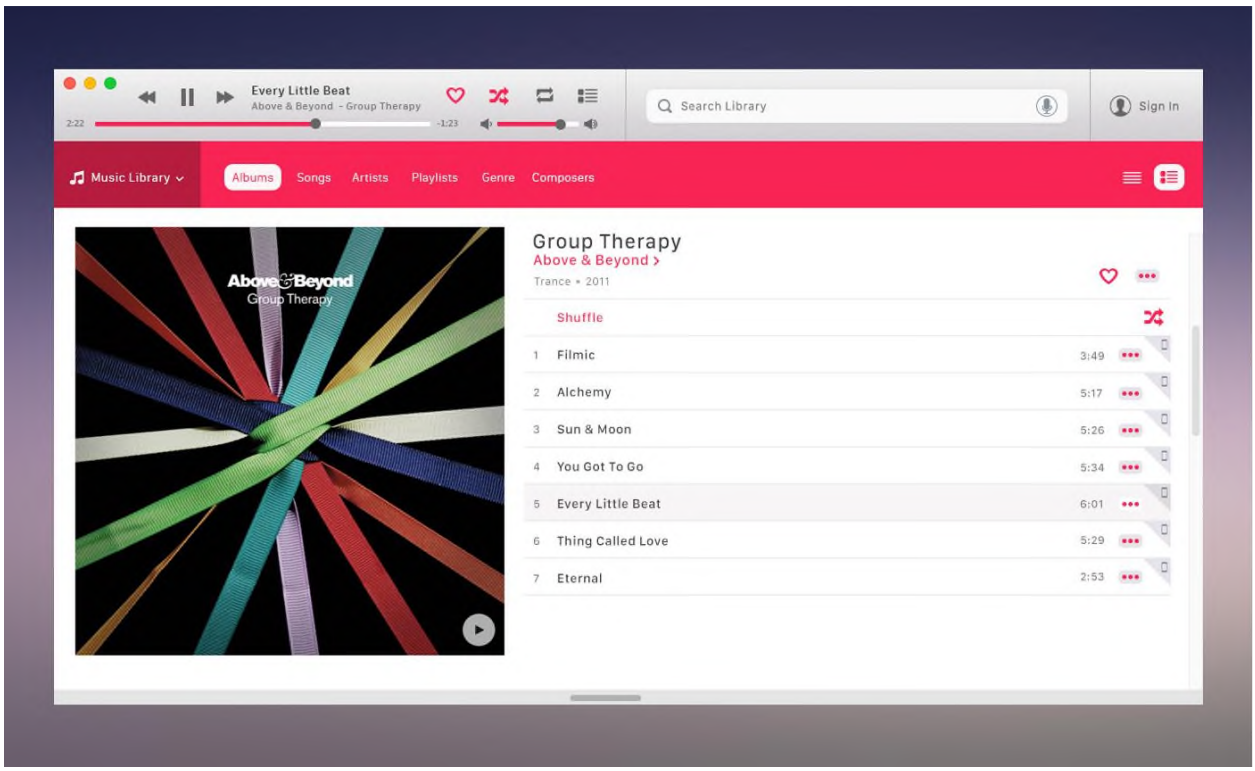

Рисунок 22 – Референсы iTunes

3.2 Архитектура

В качестве основного архитектурного решения для frontend приложения была выбрана чистая архитектура [3]. Это набор рекомендаций для создания масштабируемого, поддерживаемого, адаптируемого приложения. Использование этих рекомендаций в течении длительное времени, вплоть до настоящего времени, показывает их актуальность. Данный подход делает акцент на независимости архитектуры от фреймворков и минимизация зависимости всех частей приложения. Ориентируясь на них, была разработано схематичное представление внутренних частей приложения (рисунок 23).

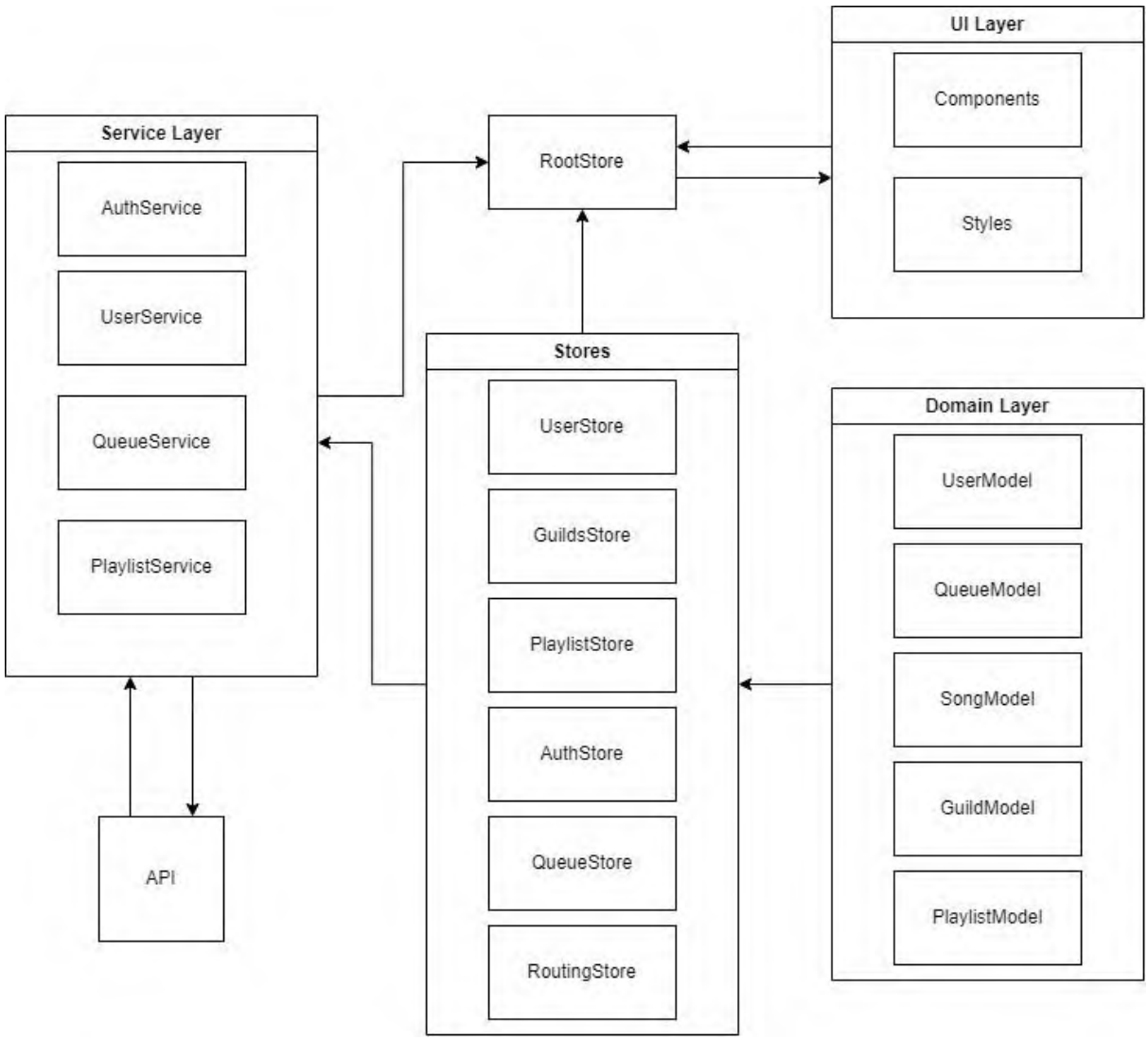

Рисунок 23 – Схематичный вид структуры приложения

Чистая архитектура подразумевает наличие абстрагированных частей приложения, не зависящих от внешних факторов, таких как фреймворк, язык программирования, и т. п. Основным принципом в этом подходе, является создание «чистых зависимостей», что означает что высокоуровневые модули не должны завесить от более низкоуровневых деталей, а детали не должны зависеть от абстракций. Данный принцип создает более прозрачное видение приложения, что упрощает использование кодовой базы и ее поддержки.

В frontend приложении Abobot также используется дизайн паттерн Service locator [12]. Этот паттерн часто критикуют [59], но для разработки Web-приложения масштабов разрабатываемого приложения, является очень простым и понятным решением для связи визуальной и функциональной

части, позволяя внедрять функциональные компоненты в компоненты интерфейса.

3.3 Реализация

Все приложение разделено на 5 уровней (Services, UI, Store, Domain, API) и односторонними зависимостями между ними. В дополнение к разделению проведена работа по абстрагированию реализации, для каждого элемента доменных моделей, хранилищ и сервисов созданы соответствующие интерфейсы и абстрактные классы.

Помимо обычного REST API [30; 54] используется протокол web-socket, посредствам которого отображается информация об очереди, текущем треке и текущей длительности трека.

Для взаимодействия с API Discord на стороне клиента и сервера Abobot реализована авторизация по средствам протокола Oauth2 [17]. Использование этого протокола позволяет использовать данные Discord для приложения без надобности сохранять и обновлять информацию в базе данных. Для повторной авторизации и выхода из сервиса создан сервис AuthService и StorageService, которые взаимодействуют с API браузера (Local Storage), куда записывается авторизационный токен Discord, его время жизни и токен обновления.

Функционал приложения разделен по конечным точкам, доступ к ним осуществляется благодаря навигационной панели в верхней части сервиса (приложение Б). Доступ на некоторые страницы возможен только авторизованным пользователям через Discord.

MainPage – главная страница приложения, на ней расположено приветственное сообщение, ссылки на сервис сбора средств и опрос обратной связи (рисунок 24).

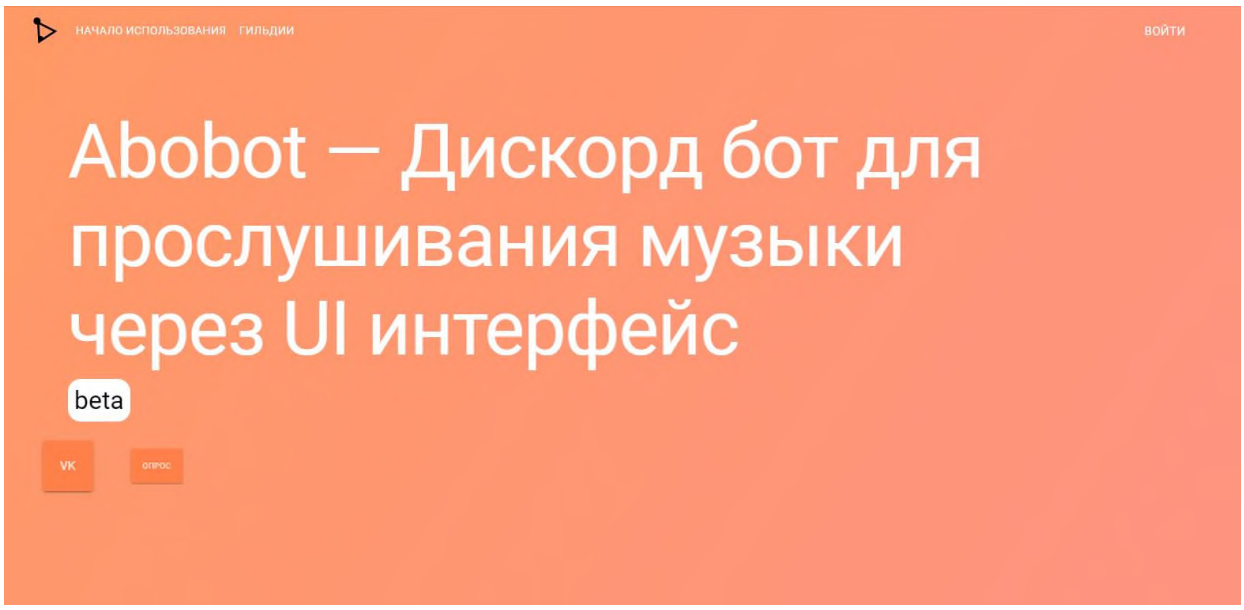

Рисунок 24 – MainPage приложения

HowToStartPage – страница с инструкцией по началу работы с ботом, добавление на сервер, доступ к управлению (рисунок 25).

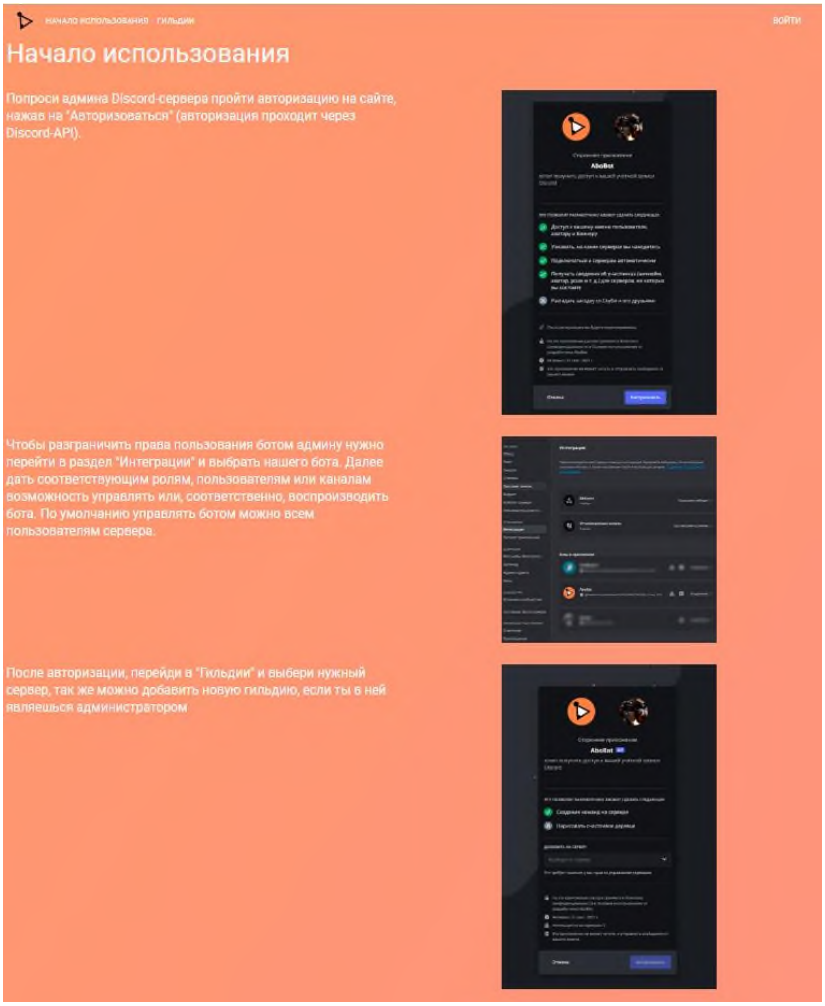

Рисунок 25 – HowToStartPage приложения

GuildsPage – страница выбора гильдии (рисунок 8). После авторизации, пользователю доступен список гильдий, гильдия – сущность в контексте сервиса Discord, описывающая виртуальный сервер пользователей, включающий в себя разноплановый функционал. В контексте приложения Abobot, авторизованный пользователь делится набором своих гильдий, в которых есть голосовые каналы, через которые возможна потоковая передача аудиодорожек. Набор гильдий формируется исходя из правила, что бот присутствует (добавлен) в гильдии пользователя, пользователь может самостоятельно добавить новую гильдию и пригласить в нее бота Abobot в качестве участника, для отображения в общем списке доступных гильдий (рисунок 26).

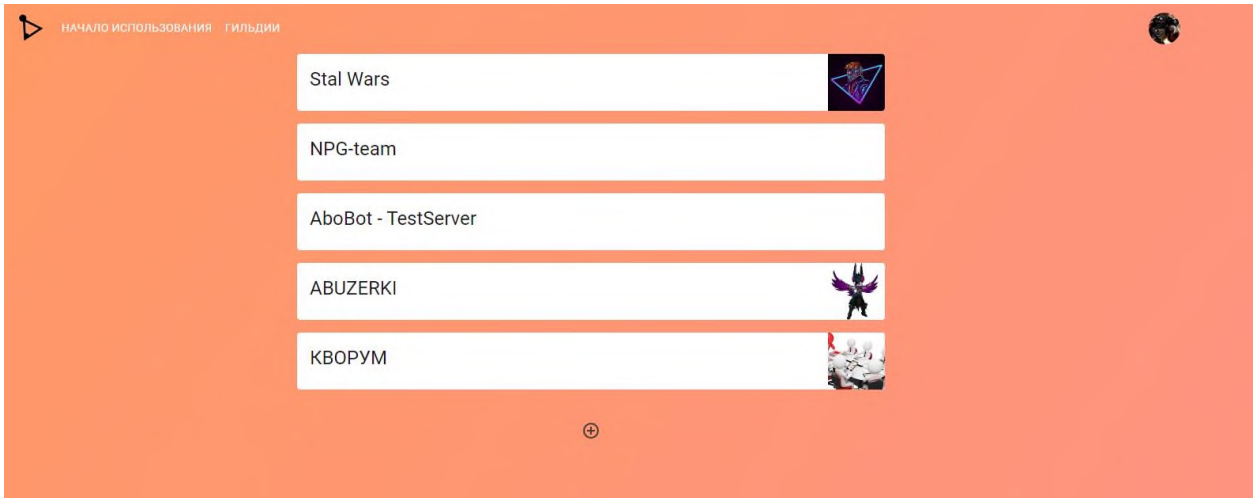

Рисунок 26 – GuildsPage приложения

PlayerPage – основная страница плеера Abobot (рисунок 9). После выбора сервера из списка гильдий, пользователь перемещается в плеер, с помощью которого можно управлять аудиодорожками из очереди воспроизведения. Чтобы управлять состоянием плеера, пользователю необходимо перейти в голосовой канал гильдии, после добавления первого трека в очередь, бот добавится в голосовой канал. После загрузки плеера создается web-socket соединение с сервером Abobot, которое позволяет всем подключенным пользователям к плееру получать актуальную информацию о текущем состоянии (приложение В) (рисунок 27).

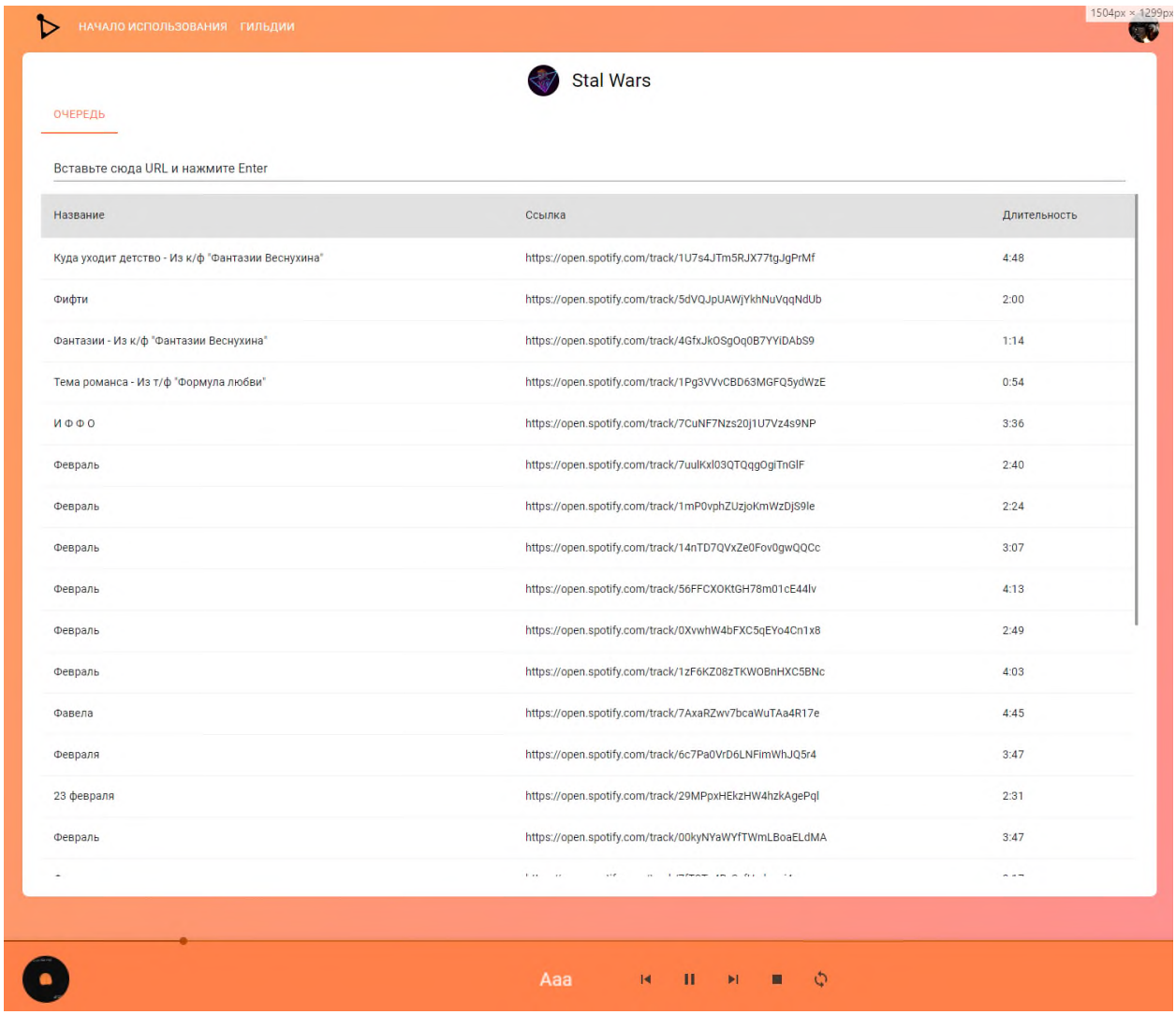

Рисунок 27 – PlayerPage приложения

3.4 Тестирование

Тестирование разрабатываемого приложения реализовано в формате unit тестов с помощью библиотеки Jest [35], также создан механизм логирование ошибок через LoggerService в файл на сервере (приложение Г).

Тестирование приложения с использованием Jest – это мощный инструмент для проверки качества кода и обнаружения ошибок в приложениях на JavaScript. Библиотека предоставляет множество функций для написания и запуска тестов, а также для управления ими.

Одним из главных преимуществ Jest является возможность использования кода, написанного на любом языке программирования, в сочетании с JavaScript. Это позволяет разработчикам использовать утилиты

для тестирования приложений, написанных на различных языках программирования.

Jest также предоставляет удобный интерфейс для написания и управления тестами, что упрощает процесс тестирования и делает его более эффективным. Кроме того, Jest имеет встроенную систему отладки, которая позволяет быстро находить ошибки в тестах и коде приложения.

В целом, использование Jest при тестировании приложений на JavaScript является мощным инструментом для обеспечения качества кода и уменьшения количества ошибок в приложении. Пример Unit теста, написанного с использованием Jest (рисунок 28).

Рисунок 28 – Пример unit теста

В этом примере импортируется компонент, который необходимо протестировать. Используется модуль «shallow» из enzyme [11], чтобы создать оболочку для нашего компонента. Далее проверяется, что оболочка соответствует ожидаемому результату, сравнивая ее с «snapshot», который создается во время выполнения теста.

3.5 Выводы

Процесс разработки приложения неоднородный, изменчивый, примером является количество прототипов и смена некоторых элементов архитектуры. Многие изменения появлялись по просьбам пользователей, пример обратной связи с предложениями представлен на рисунке 29.

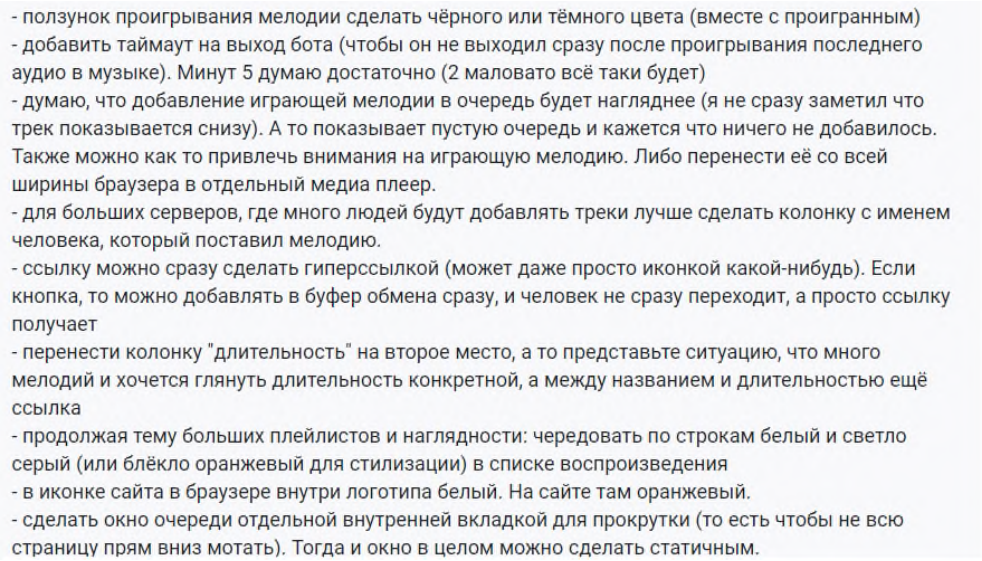

Рисунок 29 – Обратная связь и пожелание пользователей

Итоговое решение подобрано максимально оптимально, приближено к архитектурным подходам и использованию приложения. Текущая реализация еще не конечный продукт, требующий развития и отладки, находящийся на стадии Beta тестирования.

4Discord-бот Abobot: развитие и экономическая составляющая проекта

4.1 Беклог продукта

Для дальнейшей разработки был разработан набор задач, который сделает продукт более удобным для пользователей. Так же в беклог добавлены задачи по развитию инфраструктуры и стабильности работы приложения:

– создать новый поиск среди нескольких источников, официально разрешить поиск по названию;

– создать механику добавления треков в очередь из созданных плейлистов;

– создать механику перемотки трека в момент проигрывания;

– создать механику перемещения треков внутри очереди;

– добавить никнейм пользователя, который добавил трек;

– создать механику перемешивания очереди, продумать архитектуру решения;

– изменить методы доступа пользователей до управления очередью;

– исследовать масштабирование серверов хостинга;

– создать тестовый стенд на VDS.

4.2 Афиширование в Discord сообществе

Для дальнейшего распространения среди пользователей Discord необходимо добавить Abobot в листинги ботов. Верификация бота Discord, для этого необходимо набрать постоянное использование минимум на 75 серверах и удовлетворять требованиям Discord [46].

4.3 Экономическая составляющая проекта

Ресурсы, потраченные для разработки продукта, ресурсы делятся на временные и материальные. Временные ресурсы, потраченные на разработку (рисунок 31):

– на разработку Web-приложения понадобилось 200 человеко-часов;

– на разработку сервера потрачено 180 человеко-часов.

– Материальные ресурсы, потраченные в процессе разработки:

– стоимость домена abobot.ru для приложения 1000 рублей в год;

– стоимость хостинга NetAngels в среднем 4000 рублей в год (рисунок 30);

– gitlab CI/CD – стоимость за хранение gitlab-runner, входит в стоимость хостинга;

– подписка на IDE Webstorm – бесплатно, учебная лицензия.

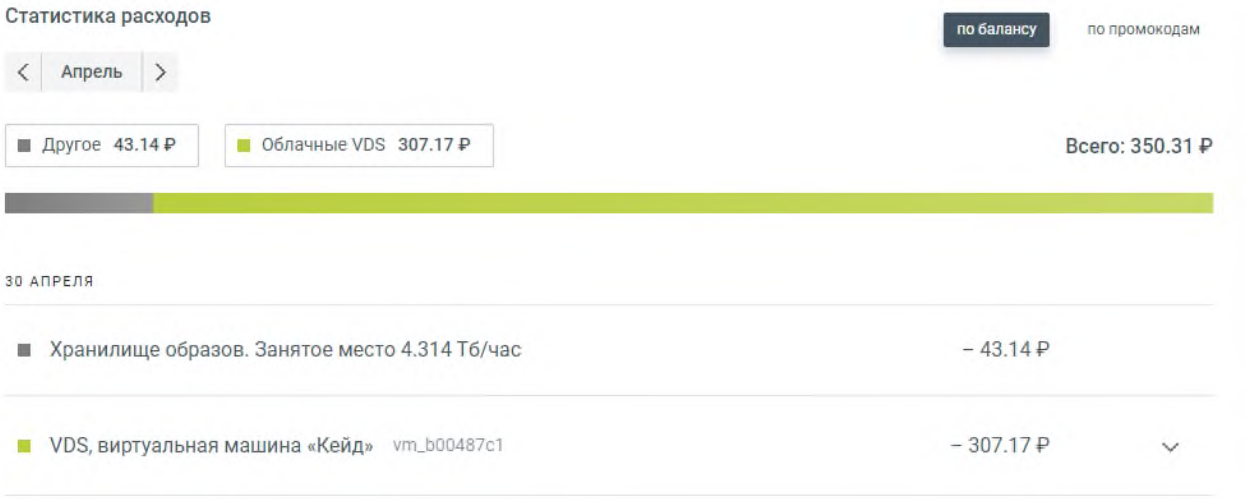

Рисунок 30 – Детализация расходов на хостинг за апрель 2023 года

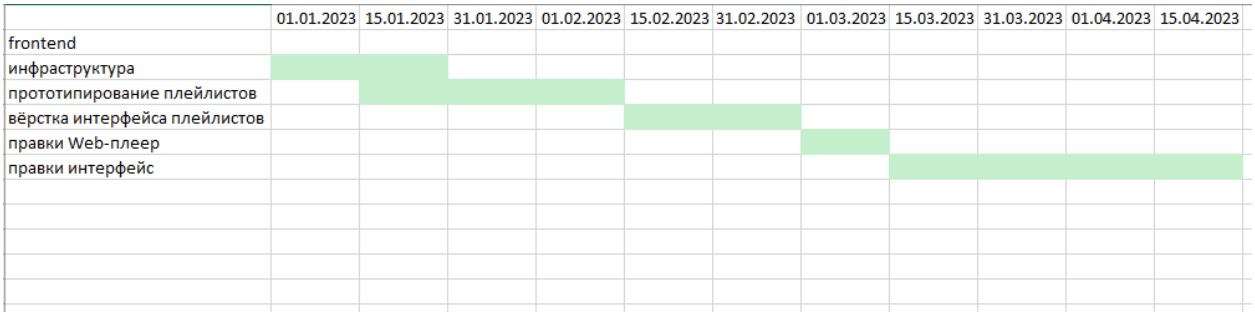

Рисунок 31 – Диаграмма Ганта Frontend с 01.01.2023

Возраст целевой аудитории Discord 12–45 лет, две трети из которых мужчины (65%/35%), а 93% состоят как минимум в двух сообществах [57].

Стейкхолдеры проекта. Внутренние стейкхолдеры – это основатели проекта, меценаты. Внешние – конкуренты, листинг ботов Discord.

Прямые экономические эффекты – уменьшение стоимости в сравнении с использованием доступных ресурсов хостинга. Удешевление разработки

ввиду использования архитектурных решений, позволяющих производить дешевое масштабирование продукта.

Косвенные экономические эффекты – рост количества активных пользователей позволит увеличить количество благотворителей проекта. Наличие активной аудитории, благоприятно воздействует на развитие проекта (добавление нового функционала, новых источников воспроизведения, улучшение метрик и инфраструктуры для поддержки большего количества пользователей). Удешевление себестоимости и уменьшение «холостой» работы проекта. Холостая работа характеризуется работой приложения без активных очередей проигрывания и использования бота внутри Discord.

Риски проекта. Ввиду начальной стадии масштабирования важно уделить внимание к получению новых пользователей продукта, что является ресурсозатратным мероприятием. Необходимо создать пиар-механизм и продумать способы распространения среди целевой аудитории. Чем быстрее происходит узнаваемость продукта, тем проще набирать новых пользователей.

Возможны претензии со стороны сервисов откуда берётся музыка для проигрывания, но ввиду внутреннего лицензирования и проверки подписки пользователей на используемые источники, приложение защищено от нелицензированного прослушивания и распространения музыкальных произведений. Риски отключения хостинга приложения или его перегрузки берёт на себя NetAngels, но даже в критической ситуации инфраструктура приложения может быть развёрнута на аналогичных сервисах хостинга в кратчайшие сроки.

Монетизация. Текущие ограничения для бесплатного использования – это количество хранимой информации в базе данных (плейлисты пользователей максимально 1 плейлист на 20 песен). Прорабатывается подписка на сервис, смысл которой подразумевает расширение хранимых данных для пользователя, возможен разный уровень подписки, открывающий разное количество расширений к основному функционалу и возможностям

приложения (расширение количества плейлистов, приоритет в использовании ресурсов сервера).

Меценатство со стороны пользователей (рисунок 32) по средствам сервиса платных подписок Boosty [56], помогающий и облегчающий сбор денежных пожертвований для различных проектов. Возможна реализация «изолированных решений», представляющих отдельную реплику Abobot для гильдии заказчика, что позволяет использовать отдельные мощности для сервера и не зависеть от всей инфраструктуры бота Abobot.

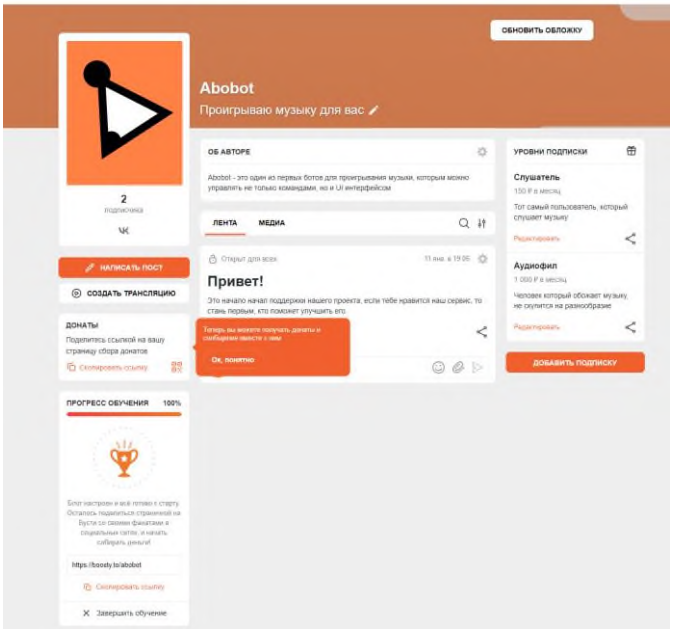

Рисунок 32 – Страница на сервисе boosty [56]

4.4 Выводы

Текущее состояние проекта – масштабирование и набор новой аудитории. Наибольшую ценность сейчас представляет узнаваемость и люди, использующие продукт. Необходимым этапом для масштабирования является публикация на листингах, для увеличения охвата аудитории. Добавление новых уникальных функций также благоприятно скажется на масштабировании. Уже составленный план развития упрощает понимание и прогнозирование развития. Совокупное количество потраченных ресурсов 380 человека-часов и 12000 рублей без учёта оплаты работы разработчиков (так

как проект появился ввиду инициативы разработчиков, трудозатраты не покрывались материальной компенсацией).

ЗАКЛЮЧЕНИЕ

В результате проделанной работы по разработке Web-приложения для бота Abobot, достигнута поставленная цель и выполнены поставленные задачи.

В первой главе магистерской диссертации рассмотрены аналоги и подобные публичные программные решения. Описана и проанализирована предметная область разработки, описаны преимущества и потенциальная выгода от разработанного продукта.

Во второй главе описаны технические требования, получена сравнительная характеристика доступных программных средств (фреймворков и библиотек) выбраны оптимальные базовые решения для разработки – фреймворк React, библиотека Mobx. Определен полный стек используемых технологий, с помощью которых произведена разработка Web-приложения Описан используемый фреймворк управления и программные средства разработки.

Третья глава посвящена описанию процесса разработки – прототипированию интерфейсов приложения, архитектуре и инфраструктуре, тестированию.

Четвертая глава посвящена экономической составляющей проекта, будущему развитию проекта – потраченным ресурсам, прямым и косвенным экономическим эффектам.

В качестве будущего развития проекта, помимо исправления багов, рассматриваются варианты добавления возможности перемешивания очереди, совмещения нескольких очередей или плейлистов друг с другом в один общий и перемотка композиции.

Основные тезисы и результаты исследования представлены на студенческой конференции ИНТЕР в секции «Искусственный интеллект и программная инженерия» (2023 г.)

Результаты работы отражены в публикациях:

Овчинников В.О., Гаев М.А. РУССКОЯЗЫЧНЫЙ DISCORD-БОТ ДЛЯ ПРОИГРЫВАНИЯ МУЗЫКИ С ПОДДЕРЖКОЙ WEB-ИНТЕРФЕЙСА / В.О. Овчинников, Гаев М.А. // «Конференция ИНТЕР – Информационные технологии и радиоэлектроника». Секция «Искусственный интеллект и программная инженерия»: сборник конференций. – Екатеринбург [УрФУ], 2016. – С. 31-34.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1.VK Музыка: офиц. сайт. – URL: https://music.vk.com/ (дата обращения: 22.05.2023).

2.Управление проектом по Agile методике // Хабр. – URL: https://habr.com/ru/companies/otus/articles/710034/ (дата обращения: 20.05.2023).

3.Чистая архитектура // Хабр. – URL: https://habr.com/ru/articles/269589/ (дата обращения: 17.05.2023).

4.Что такое CI/CD? Разбираемся с непрерывной интеграцией и непрерывной поставкой // Хабр. – URL: https://habr.com/ru/companies/otus/articles/515078/ (дата обращения: 20.05.2023).

5.Яндекс Музыка: офиц. сайт. – URL: https://music.yandex.ru/home (дата обращения: 22.05.2023).

6.Abobot: офиц. сайт. – URL: http://www.abobot.ru/ (дата обращения: 24.05.2023).

7.Fork: офиц. сайт. – URL: https://fork.dev/ (дата обращения: 25.05.2023).

8.Fredboat | Multipurpose Discord Music Bot: офиц. сайт. – URL: http://fredboat.com (дата обращения: 22.05.2023).

9.Git: офиц. сайт. – URL: https://git-scm.com/ (дата обращения: 20.05.2023).

10.Gusic: офиц. сайт. – URL: https://gusic.xyz/ (дата обращения: 24.05.2023).

11.Enzyme: офиц. сайт. – URL: https://enzymejs.github.io/enzyme/ (дата обращения: 25.05.2023).

12.James E. McDonough. Automated Unit Testing with ABAP / James E. McDonough. – Apress, Berkeley, CA, 2021 – 508p.

13.JuniperBot: офиц. сайт. – URL: https://juniper.bot/ (дата обращения: 22.05.2023).

14.Meme Land: офиц. сайт. – URL: https://meme-land.site/ (дата обращения: 24.05.2023).

15.NestJS: офиц. сайт. – URL: https://nestjs.com/ (дата обращения: 24.05.2023).

16.Next.js: офиц. сайт – URL: https://nextjs.org/ (дата обращения: 26.05.2023).

17.OAuth 2.0: офиц. сайт. – URL: https://oauth.net/2/ (дата обращения: 18.05.2023).

18.Official results for js web frameworks benchmark. – URL: https://krausest.github.io/js-framework-benchmark/ (дата обращения: 13.05.2023).

19.LavaPlayer – Audio player library for Discord // sedmelluq. – 2023. – URL: https://github.com/sedmelluq/lavaplayer (дата обращения: 26.05.2023).

20.Dyno: офиц. сайт. – URL: https://dyno.gg/ (дата обращения: 24.05.2023).

21.MEE6: офиц. сайт. – URL: https://mee6.xyz/en (дата обращения: 24.05.2023).

22.The Official YAML Web Site: офиц. сайт. – URL: https://yaml.org/ (дата обращения: 20.05.2023).

23.VK Music Bot: офиц. сайт. – URL: https://vkmusicbot.megaworld.space/ (дата обращения: 22.05.2023).

24.Vue.js: офиц. сайт. – URL: https://ru.vuejs.org/ (дата обращения: 26.05.2023).

25.WebSockets Standard: офиц. сайт. – URL: https://websockets.spec.whatwg.org/ (дата обращения: 17.05.2023).

26.What is Agile? | Agile 101 | Agile Alliance: офиц. сайт. – URL: https://www.agilealliance.org/agile101/ (дата обращения: 20.05.2023).

27.YouTube Music: офиц. сайт. – URL: https://music.youtube.com/ (дата обращения: 22.05.2023).

28.A survey on reactive programming / E. Bainomugisha, A. L. Carreton, T. V. Cutsem [et al.] // ACM Computing Surveys. – 2013. – Vol. 45. – № 4. – P. 1–34.

29.Babel: офиц. сайт. – URL: https://babeljs.io/ (дата обращения: 17.05.2023).

30.Fielding R. RFC9112: офиц. сайт. – URL: https://httpwg.org/specs/rfc9112.html (дата обращения: 18.05.2023).

31.Figma: офиц. сайт. – URL: https://www.figma.com/files/recent?fuid=992681900070749836 (дата обращения: 22.05.2023).

32.MobX State Router: офиц. сайт. – URL: https://nareshbhatia.github.io/mobx-state-router/ (дата обращения: 17.05.2023).

33.Inc G. B. Groovy: офиц. сайт. – URL: https://groovy.bot/ (дата обращения: 22.05.2023).

34.Java Documentation: офиц. сайт. – URL: https://docs.oracle.com/en/java/ (дата обращения: 26.05.2023).

35.Jest: офиц. сайт. – URL: https://jestjs.io/ (дата обращения: 20.05.2023).

36.Lighthouse: офиц. сайт. – URL: https://developer.chrome.com/docs/lighthouse/overview/ (дата обращения: 17.05.2023).

37.Spotify: офиц. сайт. – URL: https://www.spotify.com/int/why-not-available/ (дата обращения: 22.05.2023).

38.Material UI: офиц. сайт. – URL: https://mui.com/material-ui/getting-started/overview/ (дата обращения: 17.05.2023).

39.Postman: офиц. сайт. – URL: https://www.postman.com (дата обращения: 20.05.2023).

40.React: офиц. сайт. – URL: https://react.dev/ (дата обращения: 13.05.2023).

41.Rythm: офиц. сайт. – URL: https://rythm.app/ (дата обращения: 22.05.2023).

42.Socket.IO: офиц. сайт. – URL: https://socket.io/ (дата обращения: 17.05.2023).

43.webpack: офиц. сайт. – URL: https://webpack.js.org/ (дата обращения: 17.05.2023).

44.WebStorm: офиц. сайт. – URL: https://www.jetbrains.com/webstorm/ (дата обращения: 20.05.2023).

45.Highsmith J. A. Agile software development ecosystems / J. A. Highsmith. – Boston : Addison-Wesley, 2002. – 454 с. – URL: http://archive.org/details/agilesoftwaredev0000high (дата обращения: 20.05.2023).

46.Bot Verification and Data Whitelisting: офиц. сайт. – URL: https://support.discord.com/hc/en-us/articles/360040720412-Bot-Verification-and-Data-Whitelisting (дата обращения: 25.05.2023).

47.GitLab Runner: офиц. сайт. – URL: https://docs.gitlab.com/runner/ (дата обращения: 20.05.2023).

48.JavaScript | MDN: офиц. сайт. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 24.05.2023).

49.Лист Ботов Дискорд: офиц. сайт. – URL: https://server-discord.com/ (дата обращения: 21.05.2023).

50.Скайп: офиц. сайт. – URL: https://www.skype.com/ru// (дата обращения: 24.05.2023).

51.Akemi: офиц. сайт. – URL: https://akemi.life/ (дата обращения: 22.05.2023).

52.JeggyBot: офиц. сайт. – URL: https://jeggybot.xyz/ (дата обращения: 22.05.2023).

53.MobX: офиц. сайт. – URL: https://mobx.js.org/index.html (дата обращения: 17.05.2023).

54.REST: офиц. сайт. – URL: https://habr.com/ru/articles/590679/ (дата обращения: 18.05.2023).

55.iTunes: офиц. сайт. – URL: https://www.apple.com/ru/itunes/ (дата обращения: 22.05.2023).

56.Boosty.to: офиц. сайт. – URL: https://boosty.to/ (дата обращения: 29.05.2023).

57.Статистика Discord в 2023 году. – URL: https://xmldatafeed.com/statistika-discord-v-2022-godu-aktivnye-polzovateli-oczenka-dohody-i-tendenczii/ (дата обращения: 20.05.2023).

58.Серверный или клиентский рендеринг на вебе: что лучше использовать у себя в проекте и почему. – URL: https://tproger.ru/translations/rendering-on-the-web/ (дата обращения: 20.05.2023).

59.Service Locator – антипаттерн // Хабр. – URL: https://habr.com/ru/companies/otus/articles/694458/ (дата обращения: 20.05.2023).

ПРИЛОЖЕНИЕ А

Конфигурация pipeline GitLab CI/CD

```
image: node:16.13.1
stages:
  - install_deps
  - build
  - publish
  - deploy

variables:
  TAG_LATEST: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:latest
  TAG_COMMIT:
$CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:$CI_COMMIT_SHORT_SHA

cache:
  paths:
    - node_modules/

install_dependencies:
  stage: install_deps
  before_script:
    - apt-get update -qy
    - apt-get install yarn -y
    - yarn global add node-gyp
  script:
    - yarn
  artifacts:
    paths:
      - node_modules/

build_app:
  stage: build
  script:
    - CI=false yarn build

publish:
  image: docker:20
  stage: publish
  variables:
    DOCKER_DRIVER: overlay2
    DOCKER_TLS_CERTDIR: ""

  services:
    - name: docker:20-dind
      alias: docker
      command: ["--tls=false"]

  script:
    - docker build -t $TAG_COMMIT -t $TAG_LATEST .
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN
$CI_REGISTRY
    - docker push $TAG_COMMIT
```

```
    - docker push $TAG_LATEST

deploy:
  image: alpine:latest
  stage: deploy
  tags:
    - deployment
  script:
    - chmod og= $ID_RSA
    - apk update && apk add openssh-client
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no
$SERVER_USER@$SERVER_IP "docker login -u gitlab-ci-token -p
$CI_BUILD_TOKEN $CI_REGISTRY"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no
$SERVER_USER@$SERVER_IP "docker pull $TAG_COMMIT"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no
$SERVER_USER@$SERVER_IP "docker container rm -f abobot || true"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no
$SERVER_USER@$SERVER_IP "docker run -d -p 8080:80 --name abobot
nginx $TAG_COMMIT"
  environment:
    name: production
    url: http://213.189.221.42
  only:
    - main
```

ПРИЛОЖЕНИЕ Б

Список конечных точек приложения Abobot

```
export const routes = [
  {
    name: 'main',
    pattern: '/',
  },
  {
    name: 'howToStart',
    pattern: '/how-to-start',
  },
  {
    name: 'auth',
    pattern: '/auth',
  },
  {
    name: 'guilds',
    pattern: '/guilds',
  },
  {
    name: 'requireAuth',
    pattern: '/require-auth',
  },
  {
    name: 'player',
    pattern: '/guilds/:guildId',
  },
  {
    name: 'notFound',
    pattern: '/not-found',
  },
];
```

ПРИЛОЖЕНИЕ В

Реализация web-socket соединения с сервером

```
export enum SocketConnection {
  Queue = 'queue',
}
export type WsConnectionListener = (response: any) => void;

// Класс отвечающий за транспортировку socket запросов
export class SocketsStore {
  connections: Map<string, Socket> = new Map();
  rootStore: RootStore;

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
  }

// Установка слушателей для канала соединения
  addEventListener(connectionName: string, evt: string,
listener: WsConnectionListener) {
    const connection = this.connections.get(connectionName);
    connection?.on(evt, (response) => {
      if (isNumber(response) || response?.success) {
        listener(response);
        return;
      }
      console.log(response, evt);
      this.rootStore.alertService.addNotification({ type:
AlertType.Error }, response?.data?.message);
    });
  }
// Функция отправки сообщений в канал соединения
  emitMessage(connectionName: string, event: string, message?:
any) {
    const connection = this.connections.get(connectionName);
    connection?.emit(event, message);
  }
// Функция создания соединения
  createConnection(connectionName: string, uri: string, options:
Partial<ManagerOptions & SocketOptions> | undefined) {
    const connection = io(uri, options);
    this.addDefaultSubscriptions(connection);
    this.connections.set(connectionName, connection);
    return connection.id;
  }
//Функция удаления соединения
  deleteConnection(name: string) {
    const connection = this.connections.get(name);
    connection?.disconnect();
    this.connections.delete(name);
  }
// Функция инициализации стандартных подписок для канала
соединения
```

```
  private addDefaultSubscriptions(connection: Socket) {
    connection.on('connect', () => console.info(connection.id, '
connected'));
    connection.on('connect_error', (error) =>
console.error(`${connection.id} error ${error}`));
    connection.on('disconnect', (reason, desc) => {
      let message;
      if (isError(desc)) {
        message = desc.message;
      } else {
        message = desc?.description;
      }
      this.rootStore.alertService.addNotification({ type:
AlertType.Error }, message);
      console.info(`${connection.id} disconnected by reason
${reason} (${message})`);
    });
  }
}
```

ПРИЛОЖЕНИЕ Г

Реализация сервиса логирования

```
// Класс логирования ошибок
export class LoggerService extends ILoggerService {
  constructor(dataCtx: IAPITransportService) {
  super(dataCtx)
  }
  public warn(message: string) {
    console.warn(message);
//отправка ошибки на сервер
    this.dataCtx.sendWarn(API_WARN_ENDPOINT)
  }

  public error(message: string) {
    console.error(message);
//отправка ошибки на сервер
    this.dataCtx.sendWarn(API_ERROR_ENDPOINT)
  }

  public info(message: string) {
    console.info(message);
  }
}
```