РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ТЕХНОЛОГИЧЕСКОГО ОБРАЗОВАНИЯ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЭЛЕКТРОННОГО ОБУЧЕНИЯ

ДИПЛОМНАЯ РАБОТА

Разработка серверной части веб-приложения
для управления спортивным клубом

Студента 4 курса
дневного отделения
Ахмедова Эдгара Тимуровича

Руководитель: Старший преподаватель
Аксютин Павел Александрович

Санкт-Петербург
2024 г.

Оглавление

Введение

Актуальность темы

Спортивные клубы сегодня представляют собой слаженно функционирующие организации, которые требуют эффективного управления как на административном, так и на техническом уровнях это мы разберем в первой главе этой дипломной работы. Однако, несмотря на неоспоримую важность такого управления, многие клубы сталкиваются с проблемой неэффективного использования ресурсов из-за отсутствия современных технологических решений для автоматизации процессов. Противоречие между необходимостью эффективного управления и его отсутствием актуализирует необходимость разработки специализированного веб-приложения для взаимодействия с спортивным клубом.

Цель работы

Цель данной дипломной работы состоит в разработке серверной части данного веб-приложения, которая позволит оптимизировать процессы взаимодействия с спортивным клубом и повысить его эффективность.

Задачи исследования

Для достижения поставленной цели необходимо решить следующие задачи:

- Проанализировать существующие потребности и проблемы в управлении спортивным клубом.
- Предложить концепцию веб-приложения, учитывающую особенности спортивных клубов и их управления.
- Разработать серверную часть веб-приложения с использованием современных технологий.
- Протестировать разработанное веб-приложение, оценить его эффективность и подготовить для него документацию.

Тестирование результатов планируется осуществить путем развертывания разработанной серверной части веб-приложения для тестовых запросов от реальных людей, что позволит проверить работоспособность программы и отладить возникшие неожиданные ошибки.

Глава 1. Анализ существующих проблем и потребностей в управлении спортивным клубом

1.1 Идентификация основных проблем

1.1.1 Недостаточная автоматизация управленческих процессов

Одной из ключевых проблем, выявленных в ходе анализа, является недостаточная автоматизация управленческих процессов в спортивных клубах. Большинство клубов продолжают использовать устаревшие методы учета и анализа информации, такие как ручное ведение учета посещений, расписаний тренировок и финансовых операций. Это затрудняет оперативное принятие решений на основе актуальных данных и может привести к ошибкам в управлении. Например, недостоверные данные о посещаемости могут привести к неправильной оценке эффективности тренеров или неправильному планированию расписания тренировок

1.1.2 Ограниченные возможности взаимодействия с клиентами

Еще одной существенной проблемой является ограниченность возможностей взаимодействия спортивного клуба с его клиентами. В настоящее время потребители все больше ожидают удобства и доступности взаимодействия с услугами, включая возможность онлайн-записи на тренировки, оплаты абонементов, получения информации о расписании и проводимых мероприятиях. Отсутствие таких функциональностей может отпугнуть современных клиентов, привыкших к удобству онлайн-сервисов, и заставить их обратиться к конкурентам, предоставляющим более современные и удобные услуги.

1.1.3 Недостаточная эффективность управления ресурсами

Еще одной серьезной проблемой, выявленной в ходе анализа, является недостаточная эффективность управления ресурсами спортивного клуба. Нерациональное использование персонала или финансовых средств может привести к неоптимальному распределению нагрузки, неэффективному использованию ресурсов и, как следствие, снижению качества предоставляемых услуг. Например, неправильное распределение тренеров по группам может привести к недовольству клиентов и потере лояльности.

1.1.4. Неудовлетворенность клиентов

Как результат вышеупомянутых проблем, возникает неудовлетворенность клиентов спортивного клуба. Они могут испытывать неудовлетворение из-за неудобства взаимодействия с клубом, недостаточного качества предоставляемых услуг или неоправданных ожиданий. Неудовлетворенные клиенты могут не только отказаться от дальнейшего посещения клуба, но и довести свое негативное впечатление до других потенциальных клиентов, что может негативно сказаться на репутации клуба и его доходах.

Анализ этих проблем позволяет четко определить не только причины возникновения негативных явлений в работе клуба, но и выработать стратегию их решения с помощью разработки и внедрения специализированного веб-приложения.

1.2 Оценка доступных технологий для серверной разработки

При выборе технологий для разработки серверной части веб-приложения для управления спортивным клубом необходимо провести анализ доступных инструментов и фреймворков с учетом требований проекта и целей разработки. В данной главе будет представлено обширное исследование различных технологий с анализом их особенностей и преимуществ.

1.2.1 Java, фреймворк Spring и база данных MongoDB

Java - один из наиболее популярных языков программирования, широко используемый для создания серверных приложений. Он обладает мощной экосистемой инструментов и фреймворков, позволяющих разрабатывать сложные приложения любого масштаба. Фреймворк Spring является одним из наиболее распространенных инструментов для создания Java-приложений. Он предоставляет обширные средства для упрощения разработки и управления приложениями.

Преимущества:

- **Производительность и надежность**: Java известен своей высокой производительностью и надежностью, что делает его отличным выбором для критически важных приложений.
- **Широкая экосистема**: Существует огромное количество библиотек и инструментов, поддерживаемых сообществом Java, что обеспечивает широкие возможности для разработчиков.
- **Популярность**: Java и Spring являются одними из самых популярных технологий в области разработки серверных приложений, что означает наличие большого количества материалов и ресурсов для обучения и поддержки.

Недостатки:

- **Сложность**: Java может быть слишком сложным для некоторых разработчиков, особенно для тех, кто только начинает свой путь в программировании.
- **Большой объем кода**: В некоторых случаях разработка на Java требует написания большого объема кода, что может замедлить процесс разработки.

1.2.2. Python и фреймворк Django

Python - интерпретируемый язык программирования с простым и понятным синтаксисом, что делает его отличным выбором для разработки веб-приложений. Django - один из самых популярных фреймворков для создания веб-приложений на Python. Он предоставляет широкий набор инструментов и функциональности для упрощения разработки.

Преимущества:

- **Простота и удобство**: Python и Django известны своей простотой и удобством использования, что делает их отличным выбором для начинающих разработчиков и быстрого запуска проектов.
- **Большое количество готовых решений**: Django предоставляет множество готовых компонентов и модулей, которые позволяют разработчикам быстро создавать сложные приложения.
- **Широкая поддержка и активное сообщество**: Python и Django имеют огромное количество ресурсов и сообщества, которые готовы помочь в случае возникновения проблем.

Недостатки:

- **Некоторые ограничения**: В некоторых случаях Django может иметь ограничения в производительности или функциональности по сравнению с некоторыми другими фреймворками.
- **Гибкость**: В связи с тем, что Django предоставляет много "из коробки", в некоторых случаях может быть сложно настроить приложение под конкретные требования.

1.2.3 Node.js и фреймворк Express.js

Node.js позволяет использовать JavaScript для разработки серверных приложений. Express.js - один из самых популярных фреймворков для Node.js, предоставляющий минималистичные средства для создания веб-приложений и API.

Преимущества:

- **Высокая производительность**: Node.js известен своей высокой производительностью и масштабируемостью, благодаря асинхронной архитектуре.
- **Простота использования**: JavaScript является одним из самых популярных языков программирования, и многие разработчики уже знакомы с ним, что делает разработку на Node.js более доступной.
- **Многообразие модулей и плагинов**: Существует огромное количество сторонних модулей и плагинов для Node.js и Express.js, что позволяет расширить функциональность приложения по мере необходимости.

Недостатки:

- **Стабильность**: В сравнении с некоторыми другими технологиями Node.js может быть менее стабильным в работе, особенно при работе с большими объемами данных или высокой нагрузке.
- **Гибкость**: Express.js предоставляет минималистичные средства для разработки, что может быть недостаточно для некоторых проектов, требующих более сложной функциональности.

1.2.4 Golang и база данных PostgreSQL

Golang (или Go) - это язык программирования, разработанный компанией Google, который обеспечивает высокую производительность, надежность и эффективное использование ресурсов. Он широко применяется в создании высоконагруженных веб-сервисов и приложений, так как обеспечивает быстрое выполнение кода и эффективную работу с параллелизмом.

Преимущества:

- **Высокая производительность**: Golang компилируется в нативный код, что обеспечивает высокую скорость выполнения и эффективное использование ресурсов сервера.
- **Простота и чистота кода**: Синтаксис Go прост и понятен, что позволяет разработчикам быстро писать и поддерживать код.
- **Поддержка параллелизма**: Встроенная поддержка параллелизма и конкурентности делает Go отличным выбором для создания многопоточных приложений.

Недостатки:

- **Молодая экосистема**: По сравнению с Java, Python и Node.js, экосистема Go менее развита, что может ограничивать доступность некоторых инструментов и библиотек.
- **Отсутствие некоторых функциональностей**: Некоторые функции, доступные в других языках программирования и фреймворках, могут отсутствовать в Go, что может потребовать дополнительной разработки или использования сторонних библиотек.

PostgreSQL - это мощная и расширяемая система управления реляционными базами данных, которая широко используется в различных типах приложений, начиная от веб-приложений и заканчивая корпоративными системами. Использование PostgreSQL совместно с Golang предоставляет разработчикам удобные инструменты для создания производительных и надежных приложений.

Преимущества использования PostgreSQL с Go:

- **Нативная поддержка**: Golang имеет встроенную поддержку работы с базами данных через стандартный интерфейс database/sql, что делает интеграцию с PostgreSQL простой и удобной.
- **Высокая производительность**: PostgreSQL известен своей высокой производительностью и возможностями масштабирования, что делает его отличным выбором для разработки высоконагруженных приложений.
- **Богатый набор функций**: PostgreSQL предоставляет широкий спектр возможностей, включая поддержку JSON, полнотекстовый поиск, геопространственные запросы и многое другое, что позволяет разработчикам реализовывать разнообразные функциональные требования.

Недостатки использования PostgreSQL с Go:

- **Сложность настройки**: Настройка и управление PostgreSQL может потребовать некоторых усилий, особенно для больших и сложных баз данных.
- **Необходимость в знании SQL**: Для эффективной работы с PostgreSQL вам придется иметь навыки работы с SQL, что может быть вызовом для разработчиков, привыкших к ноу-коду и ORM-инструментам.

Сочетание Golang и PostgreSQL обеспечивает разработчикам мощный инструментарий для создания производительных и масштабируемых приложений, хотя это требует некоторого усилия при начальной настройке и обучении.

Ниже в таблице 1 предоставлено краткое сравнение всех вышеперечисленных решений.

Таблица 1 - сравнение доступных технологий для серверной разработки

Набор технологий	Плюсы	Минусы
Golang и база данных PostgreSQL	- Высокая производительность - Простота и чистота кода - Поддержка параллелизма - Нативная поддержка	- Молодая экосистема - Отсутствие некоторых функциональностей
Java, фреймворк Spring и база данных MongoDB	- Производительность и надежность - Широкая экосистема - Популярность	- Сложность - Большой объем кода
Python и фреймворк Django	- Простота и удобство - Большое количество готовых решений - Широкая поддержка и активное сообщество	- Некоторые ограничения - Гибкость - Медлительность
Node.js и фреймворк Express.js	- Высокая производительность - Простота использования - Многообразие модулей и плагинов	- Стабильность - Гибкость

1.3 Критерии выбора технологий: производительность, масштабируемость, удобство использования

В условиях современного спортивного бизнеса эффективное управление и взаимодействие с клиентами играют ключевую роль в успехе спортивного клуба. Разработка веб-приложения для управления спортивным клубом становится важным шагом для повышения эффективности административных процессов и улучшения опыта клиентов. При выборе технологий для создания серверной части этого приложения необходимо учитывать не только требования к функциональности, но и такие критерии, как производительность, масштабируемость, удобство использования и безопасность, гибкость и так далее.

По итогу на основании нижеперечисленных критериев был выбран набор технологий для серверной части приложения:

Производительность:

- Способность обрабатывать запросы пользователей с минимальными задержками и обеспечивать высокую отзывчивость приложения.
- Эффективное использование ресурсов сервера для обеспечения стабильной и быстрой работы приложения даже при высокой нагрузке.

Масштабируемость:

- Горизонтальное масштабирование приложения для поддержания его работоспособности при увеличении числа пользователей и объема данных.
- Вертикальное масштабирование для оптимизации производительности приложения путем увеличения мощности серверов или оптимизации кода.

Удобство использования:

- Простота и понятность синтаксиса выбранного языка программирования для ускорения разработки и обеспечения читаемости кода.
- Наличие обширной стандартной библиотеки или экосистемы инструментов, которые упрощают создание и поддержку приложения.
- Доступность документации, обучающих материалов и сообщества разработчиков для решения возникающих вопросов, и проблем.

Безопасность:

- Возможности для обеспечения безопасности передачи и хранения данных пользователей, включая механизмы аутентификации и авторизации.
- Защита от распространенных угроз, таких как атаки переполнения буфера, инъекции SQL и скрытых угроз безопасности.

Гибкость и расширяемость:

- Возможность легкого внесения изменений и добавления новой функциональности в приложение без необходимости переписывания существующего кода.
- Поддержка интеграции с другими системами и сервисами через API или стандартные протоколы обмена данными.

Сообщество и поддержка:

- Активность и размер сообщества разработчиков выбранной технологии, обеспечивающие доступность ресурсов и поддержку.
- Наличие официальной поддержки от разработчиков технологии, регулярные обновления и исправления ошибок.

Затраты на разработку и эксплуатацию:

- Оценка затрат на разработку серверной части приложения, включая время разработчиков и стоимость необходимых инструментов и ресурсов.
- Оценка затрат на эксплуатацию и поддержку приложения, включая затраты на хостинг, обновления и обслуживание серверов.

Каждый из перечисленных критериев выбора технологий играет важную роль в процессе разработки серверной части веб-приложения для спортивного клуба. Производительность, масштабируемость и удобство использования являются основными аспектами, которые определяют эффективность работы приложения и удовлетворение потребностей пользователей.

Производительность напрямую влияет на скорость обработки запросов и отклик приложения на действия пользователей. Быстрая и отзывчивая работа приложения не только улучшает пользовательский опыт, но и способствует повышению уровня доверия пользователей к продукту. Кроме того, эффективное использование ресурсов сервера позволяет сократить затраты на обслуживание и эксплуатацию приложения.

Масштабируемость, в свою очередь, обеспечивает гибкость при росте числа пользователей и объема данных. Система должна легко масштабироваться, чтобы сохранять стабильную работу приложения даже в условиях резкого увеличения нагрузки. Горизонтальное и вертикальное масштабирование позволяют оптимизировать работу приложения и обеспечивать его надежную работу в любых условиях.

Удобство использования является также важным аспектом при выборе технологий, поскольку влияет на скорость разработки, обучение новых сотрудников и поддержание кодовой базы. Простой и понятный синтаксис, наличие подробной документации и обширной стандартной библиотеки существенно упрощают работу разработчиков и позволяют быстро адаптироваться к новым задачам и требованиям проекта.

Кроме основных критериев, таких как производительность, масштабируемость и удобство использования, также необходимо учитывать безопасность приложения, его гибкость и расширяемость, а также доступность сообщества разработчиков и затраты на разработку и эксплуатацию. Все эти факторы в совокупности определяют выбор оптимальной технологии для реализации серверной части веб-приложения для спортивного клуба

1.4 Обоснование выбора конкретных технологий для реализации серверной части веб-приложения

Выбор конкретной технологии для реализации серверной части веб-приложения является ключевым этапом в разработке проекта. В данной главе будет представлено обоснование выбора Golang в качестве основной технологии для создания серверной части приложения для управления спортивным клубом.

- **Производительность**:

Golang известен своей высокой производительностью и эффективным использованием ресурсов. Статическая типизация и компиляция в машинный код позволяют Golang обрабатывать запросы быстрее и с меньшими нагрузками на сервер, что особенно важно для веб-приложений с высокими требованиями к производительности.

- **Масштабируемость**:

Golang обеспечивает простоту создания многопоточных приложений и эффективное управление параллелизмом. Это делает его идеальным выбором для создания масштабируемых веб-сервисов, способных обрабатывать большие объемы запросов и адаптироваться к росту пользовательской базы без потери производительности.

- **Удобство использования**:

Golang имеет простой и понятный синтаксис, который делает код читаемым и легко поддерживаемым. Благодаря обширной стандартной библиотеке и активному сообществу разработчиков, создание веб-приложений на Golang становится быстрым и эффективным процессом. Кроме того, инструменты разработки и документация хорошо поддерживаются, что облегчает работу с языком.

- **Экосистема и поддержка**:

Golang имеет развитую экосистему инструментов и библиотек, которые позволяют разработчикам быстро создавать различные компоненты веб-приложений. Благодаря активному сообществу разработчиков, всегда можно найти поддержку и решение возникающих проблем, а также вносить свой вклад в развитие языка и его экосистемы.

- **Поддержка параллелизма и конкурентности**:

Golang предоставляет встроенную поддержку параллелизма и конкурентности, что особенно важно для серверных приложений, обрабатывающих множество одновременных запросов. Механизм горутин и

каналов позволяет эффективно управлять параллельными процессами и обеспечивать высокую отзывчивость приложения.

- **Простота развертывания и обслуживания**:

Golang обеспечивает легкость в развертывании и обслуживании приложений благодаря своей компактности и минимальным зависимостям. Компиляция в один исполняемый файл упрощает упаковку и доставку приложения на сервер, а стандартный набор инструментов для тестирования и профилирования помогает обнаруживать и устранять проблемы в работе приложения

На основе представленного обоснования выбора Golang для реализации серверной части веб-приложения для управления спортивным клубом можно сделать следующие выводы. Golang сочетает в себе высокую производительность, масштабируемость, удобство использования и поддержку параллелизма, что делает его оптимальным выбором для создания эффективных и надежных веб-сервисов.

Простота синтаксиса и обширная стандартная библиотека Golang упрощают процесс разработки и поддержки приложений, а активное сообщество разработчиков обеспечивает доступ к ресурсам и поддержке в решении возникающих задач. Благодаря своей экосистеме и инструментам разработки, Golang позволяет создавать высокопроизводительные и масштабируемые веб-приложения, способные эффективно обрабатывать большие объемы данных и запросов.

Таким образом, выбор Golang для реализации серверной части веб-приложения является обоснованным и обеспечивает оптимальное сочетание производительности, надежности и удобства использования, что способствует успешной реализации проекта и удовлетворению потребностей пользователей.

1.5 Выбор способа визуализации работы серверной части

При разработке серверной части веб-приложения для управления спортивным клубом существует множество вариантов визуализации работы, каждый из которых имеет свои уникальные особенности и преимущества. В данной главе мы рассмотрим различные варианты визуализации работы серверной части, с акцентом на их преимущества и возможности применения.

1.5.1 Веб-интерфейс

Веб-интерфейс представляет собой один из наиболее распространенных и удобных способов визуализации работы серверной части. Пользователи могут взаимодействовать с сервером через стандартный веб-браузер, получая доступ к различным функциям и данным. Веб-интерфейс обеспечивает простоту использования и широкий охват аудитории, поскольку большинство пользователей имеют опыт работы с веб-приложениями. Кроме того, веб-интерфейс позволяет создавать интерактивные и привлекательные пользовательские интерфейсы с использованием современных технологий веб-разработки, таких как HTML, CSS и JavaScript.

Одним из главных преимуществ веб-интерфейса является его универсальность и доступность. Пользователи могут получить доступ к приложению из любого места и с любого устройства, подключенного к интернету, что делает его идеальным выбором для приложений с широким кругом пользователей.

1.5.2 Мобильное приложение

Мобильное приложение представляет собой еще один популярный вариант визуализации работы серверной части. Мобильные приложения обеспечивают пользователям доступ к функциональности приложения на мобильных устройствах, таких как смартфоны и планшеты. Они часто используются в ситуациях, когда удобство и мобильность играют ключевую роль, например, при записи на тренировки в спортивном клубе или отслеживании статистики тренировок.

Одним из основных преимуществ мобильных приложений является возможность использования встроенных функций устройства, таких как камера, GPS и датчики движения, для создания уникальных и интерактивных пользовательских интерфейсов. Это позволяет разработчикам создавать

приложения с интуитивно понятным и удобным интерфейсом, а также использовать различные функции устройства для обогащения пользовательского опыта.

1.5.3 Telegram бот

Telegram бот представляет собой отличную альтернативу для визуализации работы серверной части, особенно если важен быстрый и удобный доступ к информации и функциональности приложения. Боты могут выполнять различные функции, от предоставления информации до выполнения определенных задач по запросу пользователя. В случае серверной части для управления спортивным клубом, Telegram бот может быть использован для предоставления информации о расписании тренировок, регистрации на тренировки, получения расписания залов, покупки абонементов и многое другое. Telegram бот отличается своей простотой и удобством использования. Пользователи могут взаимодействовать с ботом через мессенджер Telegram, который широко используется по всему миру и имеет удобный и интуитивно понятный интерфейс. Бот может быть легко настроен для выполнения различных задач и запросов пользователей, что делает его удобным инструментом для взаимодействия с серверной частью приложения.

1.5.4 Другие варианты

Помимо вышеперечисленных вариантов, существует множество других способов визуализации работы серверной части, таких как десктопные приложения, голосовые помощники, интеграция с умными устройствами и т. д. Каждый из этих вариантов имеет свои особенности и применимость в зависимости от конкретных потребностей и целей проекта.

1.5.5 Выбор Telegram бота

С учетом рассмотренных вариантов визуализации работы серверной части, Telegram бот выделяется своей простотой использования, широким охватом аудитории и удобством взаимодействия. Благодаря популярности мессенджера Telegram и его возможностям для создания ботов, Telegram бот становится оптимальным выбором для реализации серверной части веб-приложения для управления спортивным клубом.

Ниже в таблице 2 предоставлено краткое сравнение всех вышеперечисленных решений.

Таблица 2 – сравнение доступных решений для визуализации работы серверной части

Способ визуализации	Плюсы	Минусы
Веб-интерфейс	- Универсальность и доступность - Интерактивность - Широкий охват аудитории	- Зависимость от интернета - Безопасность - Сложность тестирования - Производительность
Мобильное приложение	- Удобство и мобильность - Использование встроенных функций устройства	- Разработка и поддержка - Ресурсоемкость - Сложность обновления - Проблемы с совместимостью
Telegram бот	- Простота использования - Широкий охват - Легкость настройки	- Ограниченный функционал - Ограниченные возможности интерфейса - Зависимость от платформы
Другие варианты (Десктопные приложения, голосовые помощники и т.д)	- Специфические возможности - Интеграция с другими устройствами	- Ограниченная аудитория - Сложность разработки - Зависимость от платформы - Высокая стоимость разработки - Проблемы с обновлением

1.6 Выбор инструмента развертывания приложения

В данном разделе мы рассмотрим две ключевые технологии, широко используемые для этих целей: Docker и Kubernetes. Мы обсудим их основные преимущества и недостатки, а также как они могут быть эффективно использованы вместе для создания высокопроизводительных приложений.

1.6.1 Docker

Docker - это платформа для контейнеризации, которая позволяет разработчикам паковать приложения и их зависимости в контейнеры, обеспечивая стабильную и предсказуемую работу на различных системах. Контейнеры изолируют приложения, что упрощает их развертывание и управление.

Преимущества:

- **Портативность**: Контейнеры Docker могут быть запущены на любой системе, поддерживающей Docker, что обеспечивает согласованность среды разработки, тестирования и производства.
- **Изоляция**: Контейнеры изолируют приложения и их зависимости, что снижает вероятность конфликтов между компонентами системы.
- **Быстрое развертывание**: Контейнеры можно запускать и останавливать за считанные секунды, что ускоряет процесс развертывания приложений и их обновлений.
- **Масштабируемость**: Docker упрощает горизонтальное масштабирование приложений, что важно для высоконагруженных систем.

Недостатки:

- **Сложность настройки**: Настройка и управление контейнерами могут потребовать значительных усилий, особенно для начинающих пользователей.
- **Безопасность**: Контейнеры разделяют ядро операционной системы, что может представлять угрозу безопасности при неправильной конфигурации.
- **Ограниченные ресурсы**: Контейнеры делят ресурсы хоста, что может привести к проблемам производительности при большой нагрузке.

1.6.2 Kubernetes

Kubernetes - это система оркестрации контейнеров с открытым исходным кодом, разработанная Google. Она автоматизирует развертывание, управление и масштабирование контейнерных приложений.

Преимущества:

- **Автоматическое масштабирование**: Kubernetes может автоматически масштабировать приложения в зависимости от нагрузки, обеспечивая их надежную работу под высокой нагрузкой.
- **Управление жизненным циклом**: Kubernetes обеспечивает автоматическое управление жизненным циклом контейнеров, включая их создание, обновление и удаление.
- **Высокая доступность**: Система поддерживает распределение нагрузки и автоматическое восстановление приложений, что повышает их отказоустойчивость.
- **Расширяемость**: Kubernetes имеет модульную архитектуру, что позволяет легко добавлять новые функции и интегрироваться с различными инструментами.

Недостатки:

- **Сложность**: Kubernetes имеет сложную архитектуру и требует значительных усилий для настройки и управления, что может быть проблемой для небольших команд.
- **Ресурсоемкость**: Система оркестрации требует значительных вычислительных ресурсов для управления кластером контейнеров.
- **Кривая обучения**: Освоение Kubernetes требует времени и усилий, особенно для разработчиков, незнакомых с контейнеризацией и оркестрацией.

1.6.3 Совместное использование Docker и Kubernetes

Использование Docker для контейнеризации приложений и Kubernetes для их оркестрации предоставляет разработчикам мощные инструменты для создания, развертывания и управления масштабируемыми и надежными приложениями.

Преимущества:

- **Эффективность разработки и развертывания**: Docker обеспечивает быструю и эффективную контейнеризацию приложений, а Kubernetes автоматизирует их развертывание и управление.
- **Гибкость и масштабируемость**: Комбинация этих технологий позволяет легко масштабировать приложения и управлять их жизненным циклом.
- **Высокая надежность**: Kubernetes обеспечивает автоматическое восстановление и балансировку нагрузки, что повышает надежность приложений.

Недостатки:

- **Сложность интеграции**: Совместное использование Docker и Kubernetes может потребовать значительных усилий для настройки и интеграции.
- **Требования к инфраструктуре**: Необходимо обеспечить подходящую инфраструктуру для работы с Kubernetes, что может включать в себя выделенные серверы или облачные ресурсы.
- **Затраты на обучение и поддержку**: Необходимость в обучении команды и поддержке инфраструктуры может потребовать дополнительных ресурсов.

С учетом масштаба нашего проекта и его потребностей было принято решение использовать Docker. Docker обеспечивает портативность, изоляцию и быструю настройку окружения для приложения, что идеально подходит для небольших проектов. Kubernetes, хотя и является мощной системой оркестрации, требует значительных ресурсов и имеет высокую сложность настройки, что не всегда оправдано для небольших приложений. Таким образом, Docker оказался оптимальным выбором для нашего проекта, обеспечивая необходимые возможности контейнеризации без излишней сложности и затрат.

Ниже в таблице 3 предоставлено краткое сравнение всех вышеперечисленных решений.

Таблица 3 Сравнение технологий для развертывания приложения

Инструмент	Преимущества	Недостатки
Docker	- Портативность - Изоляция - Быстрое развертывание - Масштабируемость	- Сложность настройки - Безопасность - Ограниченные ресурсы
Kubernetes	- Автоматическое масштабирование - Управление жизненным циклом - Высокая доступность - Расширяемость	- Сложность - Ресурсоемкость - Кривая обучения
Docker + Kubernetes	- Эффективность разработки и развертывания - Гибкость и масштабируемость - Высокая надежность	- Сложность интеграции - Требования к инфраструктуре - Затраты на обучение и поддержку

1.7 Выбор инструмента для разработки

При разработке серверной части веб-приложения выбор подходящего инструмента для разработки является ключевым фактором для обеспечения эффективности и удобства работы разработчиков. Рассмотрим несколько популярных инструментов, используемых для разработки на языке Go, и проанализируем их преимущества и недостатки.

1.7.1 Visual Studio Code (VSCode)

Visual Studio Code - это популярный редактор кода с открытым исходным кодом, разработанный Microsoft. Он поддерживает множество языков программирования и обладает широкими возможностями настройки через расширения.

Преимущества:

- **Расширяемость**: Большое количество доступных расширений для поддержки различных языков, инструментов и фреймворков.
- **Интеграция с системами контроля версий**: Встроенная поддержка Git и других систем контроля версий.
- **Кроссплатформенность**: Доступен для Windows, macOS и Linux.
- **Широкое сообщество**: Большое количество ресурсов и документации, доступных онлайн.

Недостатки:

- **Производительность**: При работе с большими проектами может возникать снижение производительности.
- **Конфигурация**: Настройка множества расширений может быть сложной и трудоемкой.

1.7.2 JetBrains GoLand

GoLand - это интегрированная среда разработки (IDE) от JetBrains, специально созданная для разработки на языке Go. Она предоставляет мощные инструменты для написания, отладки и тестирования кода.

Преимущества:

- **Глубокая интеграция с Go**: Полная поддержка синтаксиса Go, рефакторинг кода, автозаполнение и подсказки.
- **Интегрированные инструменты**: Встроенная поддержка работы с базами данных и Docker, что упрощает разработку и тестирование.

- **Инструменты для отладки**: Мощные инструменты для отладки кода, профилирования и тестирования.
- **Удобство использования**: Интуитивно понятный интерфейс и удобные инструменты для навигации по коду.

Недостатки:

- **Стоимость**: GoLand является платным продуктом, что может быть значительным фактором для небольших команд и проектов. При этом у нее есть пробная бесплатная версия.
- **Ресурсоемкость**: Требует больше системных ресурсов по сравнению с редакторами, такими как VSCode.

1.7.3 Sublime Text

Sublime Text - это легкий и быстрый текстовый редактор, популярный среди разработчиков благодаря своей простоте и производительности.

Преимущества:

- **Высокая производительность**: Быстрая работа даже с большими проектами.
- **Легкость**: Малое потребление системных ресурсов.
- **Расширяемость**: Поддержка пакетов и плагинов для добавления функциональности.

Недостатки:

- **Ограниченные встроенные функции**: Для полноценной работы с Go требуются дополнительные плагины и настройки.
- **Нет встроенной поддержки работы с базами данных и Docker**: Меньше интеграции с инструментами разработки по сравнению с IDE.

1.7.4 Выбор инструмента

После рассмотрения различных инструментов для разработки на Go было принято решение использовать JetBrains GoLand. Этот выбор обусловлен следующими факторами:

Глубокая интеграция с Go: GoLand предоставляет все необходимые инструменты для работы с языком Go, включая автозаполнение, рефакторинг, отладку и тестирование.

Интеграция с базами данных и Docker: Встроенные инструменты для работы с PostgreSQL и Docker упрощают настройку и управление окружением, что особенно важно для нашего проекта.

Интуитивно понятный интерфейс: Удобный интерфейс GoLand позволяет разработчикам быстро ориентироваться в коде и эффективно решать задачи.

Использование GoLand для работы с базой данных PostgreSQL и Docker позволяет значительно упростить процесс разработки, поскольку разработчики могут использовать встроенные инструменты для управления базами данных, создания и развертывания контейнеров, а также отладки кода в единой среде. Это сокращает время на настройку и переключение между различными инструментами, повышая производительность команды и качество разрабатываемого приложения.

Ниже в таблице 4 предоставлено краткое сравнение всех вышеперечисленных решений.

Таблица 4 – сравнение доступных инструментов для разработки

Инструмент	Плюсы	Минусы
Visual Studio Code (VSCode)	- Расширяемость - Интеграция с Git	- Производительность - Сложная конфигурация
JetBrains GoLand	- Интеграция с Go - Интеграция с базами данных и Docker	- Стоимость - Ресурсоемкость
Sublime Text	- Высокая производительность - Легкость	- Ограниченные функции - Нет встроенной поддержки работы с базами данных и Docker

Выводы к главе 1

В первой главе данной работы была доказана необходимость создания Telegram бота и серверной части для него с использованием таких технологий как: Golang, PostgreSQL и Docker. Эти компоненты представляют собой ключевые элементы системы, обеспечивающие взаимодействие с пользователями и обработку запросов. На основе анализа был сделан вывод о том, что Telegram бот является эффективным инструментом для реализации коммуникации с пользователями и предоставления им необходимой информации.

Глава 2. Разработка серверной части и telegram бота

На начальном этапе проектирования системы необходимо построить структурную и функциональную модель, которая определит основные компоненты системы и их взаимосвязи. Это позволит определить основные требования к разрабатываемой системе и обеспечить ее эффективное функционирование.

Также в первой главе был обоснован выбор веб-платформы для развертывания системы. Веб-интерфейс обеспечивает простой и удобный доступ к системе для пользователей, а также позволяет легко масштабировать и обновлять приложение.

В данной главе мы подробно рассмотрим процесс разработки серверной части и Telegram бота, начиная с построения моделей и заканчивая внедрением их в работу.

2.1 Требования к продукту

Для успешной разработки серверной части веб-приложения и Telegram бота необходимо четко определить требования к продукту. Эти требования выступят в качестве основы для дальнейшей работы над проектом и обеспечат соответствие конечного результата ожиданиям пользователей и заказчика. Рассмотрим основные категории требований:

2.1.1. Функциональные требования:

- **Авторизация и аутентификация**: Пользователи должны иметь возможность аутентифицироваться в системе с помощью своего телеграмм аккаунта.
- **Управление аккаунтом**: Пользователи должны иметь возможность регистрироваться в системе, изменять персональные данные.
- **Управление тренировками**: Пользователи должны иметь возможность просматривать расписание тренировок, записываться на тренировки, отменять запись и просматривать свою историю тренировок.
- **Информация о залах**: Пользователи должны иметь возможность видеть список залов их адреса и расписание для каждого зала.
- **Управление кошельком**: Пользователи должны иметь возможность пополнять свой кошелек, оплачивать с него занятия у тренеров или абонемент в зал.
- **Управление абонементом**: покупка желаемого абонемента, получения информации о нем и его продление.

2.1.2. Нефункциональные требования:

- **Производительность**: Система должна обеспечивать высокую производительность и быстрый отклик на запросы пользователей даже при больших нагрузках.
- **Безопасность**: Система должна обеспечивать защиту персональных данных пользователей и обеспечивать безопасность авторизации и аутентификации.
- **Надежность**: Система должна быть надежной и стабильной, минимизируя возможность сбоев и ошибок в работе.
- **Масштабируемость**: Система должна быть легко масштабируемой и способной обрабатывать рост числа пользователей и объема данных.

2.1.3. Требования к Telegram боту:

- **Аутентификация**: Бот должен поддерживать механизм аутентификации, позволяя приложению проверять идентичность пользователей и их прав доступа к функциональности.
- **Взаимодействие с пользователем:** Бот должен предоставлять пользователю возможность получения информации о расписании тренировок, записи на тренировки и других важных событиях.
- **Обработка запросов:** Бот должен быть способен обрабатывать запросы от приложения, выполняя соответствующие действия в ответ на полученные команды и запросы.

Эти детальные требования к продукту будут играть важную роль на всех этапах разработки и тестирования системы. Они обеспечат четкое понимание функциональных и нефункциональных характеристик системы, что позволит команде разработки эффективно планировать, проектировать и реализовывать каждый компонент.

Ориентируясь на эти требования, разработчики смогут определить конкретные задачи, необходимые для достижения поставленных целей. Также они смогут активно использовать их в процессе тестирования, чтобы убедиться в соответствии системы заданным стандартам и ожиданиям пользователей.

Кроме того, эти требования будут служить отправной точкой для обратной связи с заказчиком и пользователями. Путем регулярной проверки соответствия разрабатываемой системы этим требованиям, команда сможет адаптировать свою работу в соответствии с изменяющимися потребностями и запросами пользователей, что в свою очередь повысит удовлетворенность конечных пользователей и обеспечит успешное завершение проекта.

Таким образом, эти требования к продукту будут играть ключевую роль в процессе разработки, обеспечивая ее целостность, качество и соответствие поставленным целям и ожиданиям всех заинтересованных сторон.

2.2 Системные требования

2.2.1 Требования для клиентской части приложения

При разработке бота, работающего на платформе Telegram, необходимо учитывать системные требования как для мобильных устройств, так и для компьютеров. Эти требования определяют минимальные характеристики устройств, на которых приложение с ботом должно корректно функционировать.

- **Требования для мобильных устройств**:
  - ***Операционная система***: Поддержка операционных систем Android (версия 5.0 и выше) и iOS (версия 9.0 и выше).
  - ***Процессор***: Минимальный процессор ARM Cortex-A53 для устройств Android и Apple A9 для устройств iOS.
  - ***Оперативная память***: Минимум 1 ГБ оперативной памяти для стабильной работы приложения.
  - ***Свободное место на диске***: Достаточное количество свободного места на устройстве для установки и хранения данных приложения.
  - ***Интернет-соединение***: Поддержка Wi-Fi или мобильного интернета для обмена данными с сервером Telegram.
- **Требования для компьютеров**:
  - Операционная система: Поддержка операционных систем Windows 7/8/10, macOS 10.10 и выше, Linux Ubuntu 16.04 и выше.
  - Процессор: Минимальный процессор Intel Core i3 или аналогичный для Windows и Linux, процессор Intel Core i5 для macOS.
  - Оперативная память: Минимум 2 ГБ оперативной памяти для стабильной работы приложения.
  - Свободное место на диске: Достаточное количество свободного места на диске для установки и хранения данных telegram.
  - Интернет-соединение: Поддержка подключения к Интернету для обмена данными с сервером Telegram.

Эти системные требования определяют минимальные характеристики устройств, на которых приложение с ботом Telegram должно быть установлено и работать стабильно. Учитывая эти требования, пользователи смогут оценить возможность запуска приложения на своих устройствах и обеспечить его оптимальную производительность.

2.2.2 Требования для серверной части приложения

При развертывании серверной части приложения для управления спортивным клубом необходимо учитывать системные требования, определяющие минимальные характеристики серверного окружения для корректной работы приложения.

- **Операционная система**:
  - Поддержка операционных систем: Linux (Ubuntu 16.04 и выше), CentOS 7 и выше, Debian 9 и выше.
- Процессор:
  - Минимальный процессор: Intel Xeon E3 или аналогичный для физических серверов, виртуальные серверы с минимальным выделенным количеством процессорных ядер (например, 1 ядро).
- Оперативная память:
  - Минимум 2 ГБ оперативной памяти для стабильной работы серверной части приложения.
- Свободное место на диске:
  - Достаточное количество свободного места на диске для установки и хранения данных приложения, а также для хранения логов и временных файлов.
- Интернет-соединение:
  - Поддержка постоянного высокоскоростного интернет-соединения для обмена данными с клиентскими устройствами и внешними сервисами.

Эти системные требования позволяют определить необходимое серверное окружение для развертывания и работы серверной части приложения для управления спортивным клубом. Учитывая эти требования, системные администраторы смогут подобрать подходящее серверное оборудование или виртуальное окружение для обеспечения стабильной и надежной работы приложения, это примерная нагрузка, используемая при расчётах 400 одновременных пользователей.

2.3 Функциональная модель

Функциональная модель представляет собой описание функций и возможностей системы, которые позволяют пользователям взаимодействовать с ней и выполнять необходимые операции. В контексте разрабатываемого бота для управления спортивным клубом через платформу Telegram, функциональная модель определяет основные задачи, которые пользователи могут выполнять с помощью бота, а также основные функции и возможности, доступные через интерфейс бота.

- Функции Telegram Бота:
  - /help: Показать доступные команды.
  - /start: Начать диалог с ботом.
  - /support: Показать контактную информацию службы поддержки.
  - /u_create <first_name> <last_name> <phone> <email>: Создать профиль пользователя с указанными именем, фамилией, номером телефона и адресом электронной почты.
  - /u_profile: Показать профиль пользователя.
  - /u_update <first_name> <last_name> <phone> <email>: Обновить профиль пользователя, можно использовать символ "-" для пропуска определенных полей.
  - /u_change_subscription <subscription_level>: Изменить уровень подписки пользователя.
  - /u_change_current_gym <gym_id>: Изменить текущий зал пользователя.
  - /w_deposit <amount>: Пополнить баланс кошелька указанной суммой.
  - /w_balance: Показать текущий баланс кошелька.
  - /w_history <start_date> <end_date>: Показать историю транзакций кошелька за указанный период времени. Формат даты: ДД.ММ.ГГГГ_ЧЧ:ММ.
  - /s_extend <months_count>: Продлить подписку на указанное количество месяцев.
  - /s_data: Показать данные о текущей подписке пользователя.
  - /g_list: Показать список доступных залов.
  - /g_schedules <gym_id>: Показать расписание тренировок для указанного зала.
  - /t_list_for_gym <gym_id>: Показать список тренеров для указанного зала.

- /t_available_booking_list <trainer_id>: Показать доступные слоты для бронирования у выбранного тренера.
- /t_current_booking_list: Показать текущие бронирования выбранного тренера.
- /t_booking <slot_id>: Забронировать слот у выбранного тренера.
- /t_unbooking <booking_id>: Отменить бронирование выбранного тренера.

Функциональная модель предоставляет общее представление о том, как пользователи будут взаимодействовать с системой через Telegram бота. Это основа для дальнейшей детализации и проектирования пользовательского интерфейса и функциональности системы.

2.4 Обоснование архитектуры и планирование разработки

В данном разделе представлено обоснование выбора микросервисной архитектуры для серверной части приложения спортивного зала, а также детальное планирование процесса разработки. Здесь проводится анализ принятых решений по выбору архитектуры, выявляются основные преимущества данного подхода, обсуждаются факторы, учтенные при принятии решения, и рассматривается роль микросервисной архитектуры в обеспечении требуемых функциональных и нефункциональных характеристик системы. Также представлен план разработки, включая этапы работы, распределение ролей и ответственности, а также выбранные инструменты и технологии для реализации проекта.

2.4.1 Анализ архитектур серверной части

В процессе выбора архитектуры серверной части приложения для спортивного клуба были рассмотрены несколько архитектурных подходов, каждый из которых имел свои преимущества и недостатки. Один из наиболее распространенных подходов — монолитная архитектура, которая предполагает создание приложения как единого целого, включающего все компоненты и функциональные модули в одном приложении. Этот подход обладает простотой в разработке и развертывании, поскольку все компоненты находятся внутри одной системы. Однако, с ростом размера и сложности приложения возникают проблемы с масштабируемостью и поддержкой, так как любые изменения или модификации требуют пересборки и перезапуска всего приложения.

В свете этих ограничений было решено выбрать микросервисную архитектуру. Этот подход предполагает разделение приложения на небольшие автономные сервисы, каждый из которых выполняет конкретную функцию. Такая архитектура позволяет обеспечить более гибкое масштабирование и развертывание приложения, так как каждый сервис может быть развернут и масштабирован независимо от остальных. Кроме того, микросервисы обеспечивают легкость в поддержке и модификации, поскольку изменения в одном сервисе не влияют на остальные. Этот подход также способствует улучшению надежности системы, так как отказ одного сервиса не приводит к полной недоступности приложения.

Помимо принципиальных преимуществ микросервисной архитектуры, также были учтены и некоторые её недостатки. Например, развертывание и управление множеством микросервисов требует дополнительных усилий в сравнении с монолитной архитектурой. Кроме того, необходимо обеспечить

надежную коммуникацию и согласованность между сервисами, что также является нетривиальной задачей.

Таким образом, выбор микросервисной архитектуры был обоснован необходимостью обеспечения масштабируемости, гибкости и надежности приложения для спортивного зала, а также учетом требований к функциональности и производительности системы.

2.4.2 Схемы отображающие функциональность приложения

Use Case схема доступа к функциональности веб-приложения

В процессе разработки была сформирована схема Use Case, охватывающая функциональности как для авторизованных, так и для неавторизованных пользователей увидеть её можно на рисунке 1.

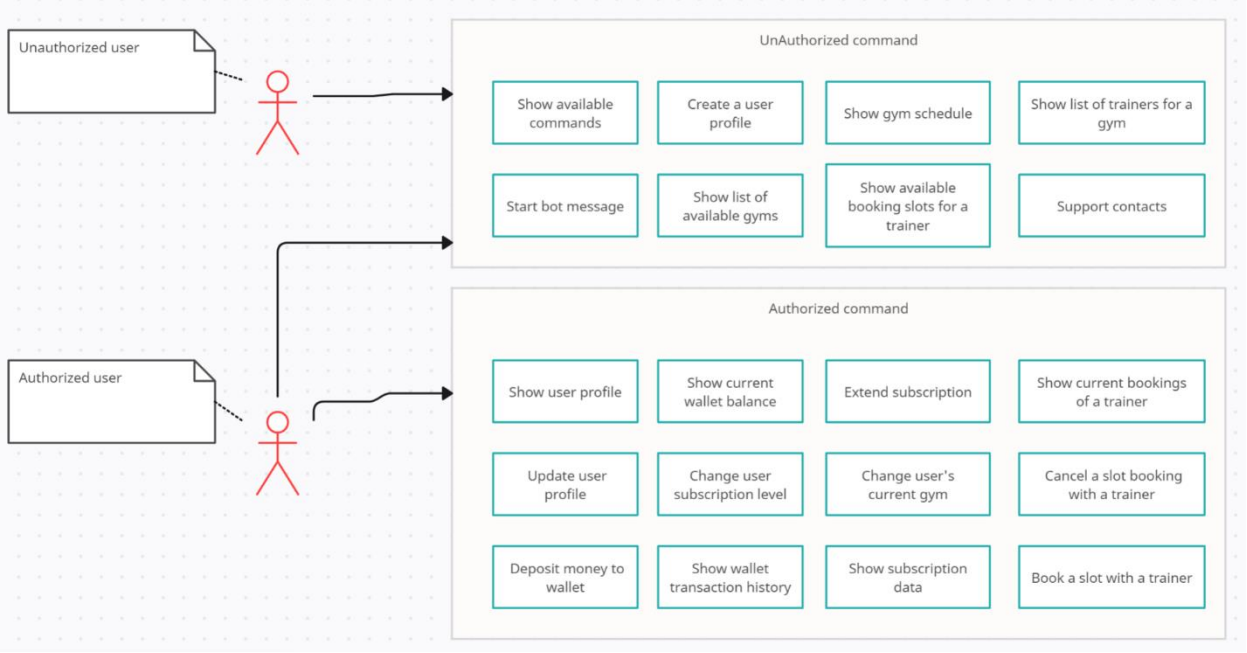

Рисунок 1 – Use Case схема доступа к функциональности веб-приложения

Эта схема представляет собой детальное описание того, как пользователи будут взаимодействовать с системой. Какие команды доступны до регистрации в приложении и после. Эта схема помогает разработчикам глубже понять требуемую функциональность конечной версии приложения.

- Функциональность для авторизованных пользователей:

Схема Use Case для авторизованных пользователей подробно описывает широкий спектр действий, доступных после успешной аутентификации. Включая функциональности, такие как управление персональным кабинетом,

просмотр и редактирование личной информации, управление абонементами, бронирование тренировок и другие возможности, предоставляемые приложением.

- Функциональность для неавторизованных пользователей:

Схема Use Case для неавторизованных пользователей описывает доступные действия без необходимости входа в систему. Это включает в себя просмотр общедоступной информации о зале и его услугах, просмотр расписания тренировок, возможность регистрации в системе и другие функциональности, которые могут быть полезными для потенциальных клиентов.

Такая схема помогает улучшить понимание требований к приложению и обеспечить полное покрытие функциональных возможностей для пользователей, независимо от их статуса в системе.

UML-схема взаимодействия микросервисов

Учитывая выбранную микросервисную архитектуру приложения для спортивного зала, была разработана UML-схема, которая наглядно отображает взаимодействие между различными компонентами системы. Изображена схема на рисунке 2.

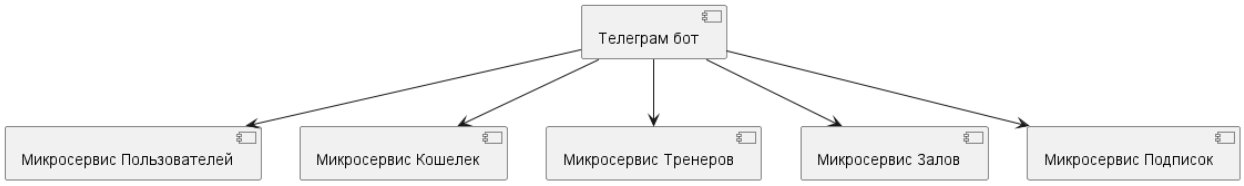

Рисунок 2 – UML-схема взаимодействия микросервисов

На схеме представлены основные микросервисы, составляющие приложение. Каждый из них выполняет определенную функцию, необходимую для обеспечения работоспособности и функциональности приложения. Например, сервис управления пользователями отвечает за регистрацию, аутентификацию и управление профилями пользователей, сервис тренеров занимается управлением расписанием тренировок и занятий, а сервис управления платежами обрабатывает платежные операции и абонентские платежи.

Взаимодействие между микросервисами происходит с помощью системы удаленного вызова процедур gRPC с использованием прокотолов protobuf. Например, сервис telegram бота может отправлять запросы на сервис управления платежами для подтверждения оплаты за бронирование занятия. Эти взаимодействия между сервисами обеспечивают выполнение бизнес-

логики и обеспечивают пользователей доступом к необходимым функциональным возможностям.

Такая UML-схема позволяет лучше понять внутреннюю структуру и взаимодействие компонентов системы, что помогает разработчикам и архитекторам принимать обоснованные решения при проектировании и разработке приложения. Она также служит важным инструментом для документирования архитектуры и обеспечения ее понимания всеми участниками проекта.

UML-схема классов для спортивного клуба

Для серверной части приложения была разработана UML-схема классов, которая наглядно отображает взаимодействие Docker контейнеров внутри серверной части. На данной схеме представлены основные классы и компоненты, включая Docker контейнеры, которые взаимодействуют между собой для обеспечения работы приложения. Увидеть схему можно на рисунке 3.

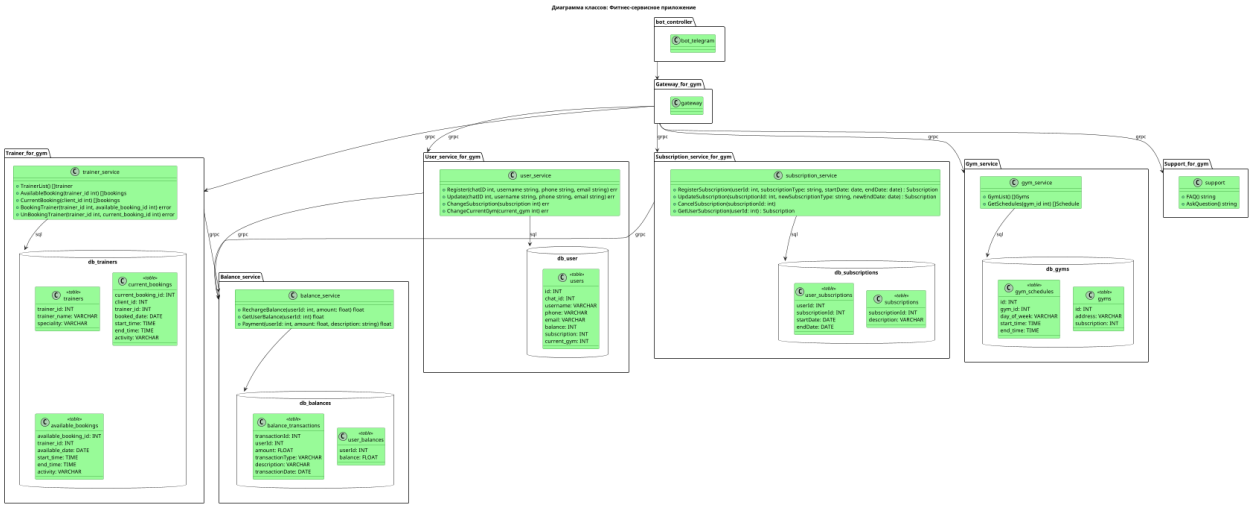

Рисунок 3 – UML-схема классов для спортивного клуба

Кроме того, на схеме также описано, какие таблицы использовались в базах данных для хранения данных, методы, используемые для выполнения запросов к базам данных, и типы данных, используемые в этих запросах. Такое детальное описание позволяет понять структуру данных и логику взаимодействия между компонентами приложения, что существенно облегчает процесс разработки, отладки и сопровождения приложения

2.5 Описание основных этапов работы пользователя с системой

По результатам разработки серверной части и Telegram бота были реализованы следующие функции.

Команда /start

Стандартная команда для начала взаимодействия с Telegram ботами, которая приветствует пользователя, рассказывает о себе, также отправляет в сообщении предложение использовать команду /help чтобы увидеть весь список команд. Это также предоставлено на рисунке 4.

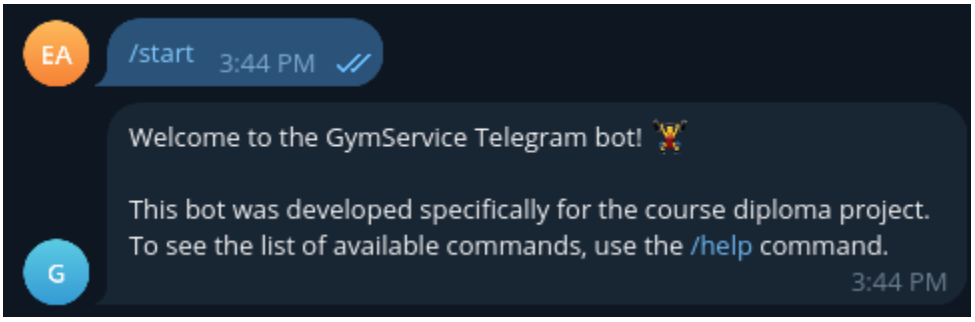

Рисунок 4 Ответ бота на команду /start

Команда /help

Стандартная команда для получения списка команд при взаимодействии с Telegram ботами. Эта команда показывает список всех актуальных команд, с которыми бот уже умеет работать, а также аргументы для каждой команды, для которой они нужны. Это также предоставлено на рисунке 5.

Рисунок 5 Ответ бота на команду /help

Команда /u_create

Данная команда нужна приложению для создания профиля пользователя, при вызове данной команды, введенные после команды аргументы валидируются, и при успешной валидации бот вызывает микросервисы пользователя, кошелька и подписок, в них он создает записи о пользователе, с учетом ранее полученных данных, после этого возвращает информацию о успешности выполнения операции. Ответ бота на эту команду предоставлен на рисунке 6.

Команда /u_profile

Данная команда нужна приложению для предоставления информации о пользователе, если пользователь есть в базе данных, то бот предоставит ответ со всеми данными для обратившегося пользователя, это продемонстрировано на рисунке 6. В противном случае бот ответит, что профиль не был найден и порекомендует пользователю воспользоваться командой /u_create для создания аккаунта, это также продемонстрировано на рисунке 7

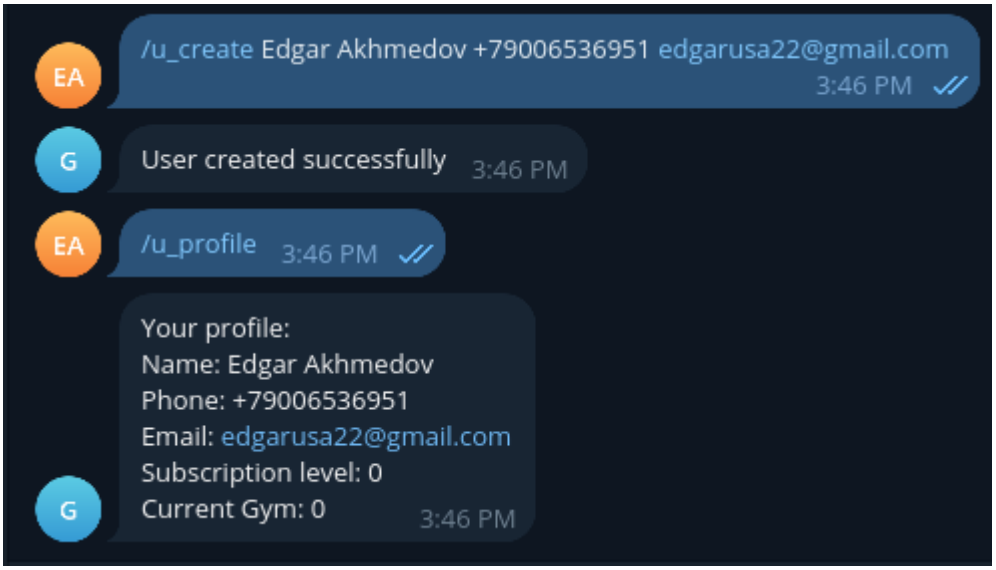

Рисунок 6 Ответы бота на команды /u_create и /u_profile

Рисунок 7 Неуспешный ответ бота на команду /u_profile

Команда /u_update

Данная команда нужна приложению для обновления личных данных пользователя, пользователь может использовать прочерк «-» для того чтобы пропустить какие-либо данные если он не хочет их обновлять. Работа команды продемонстрирована на рисунке 8

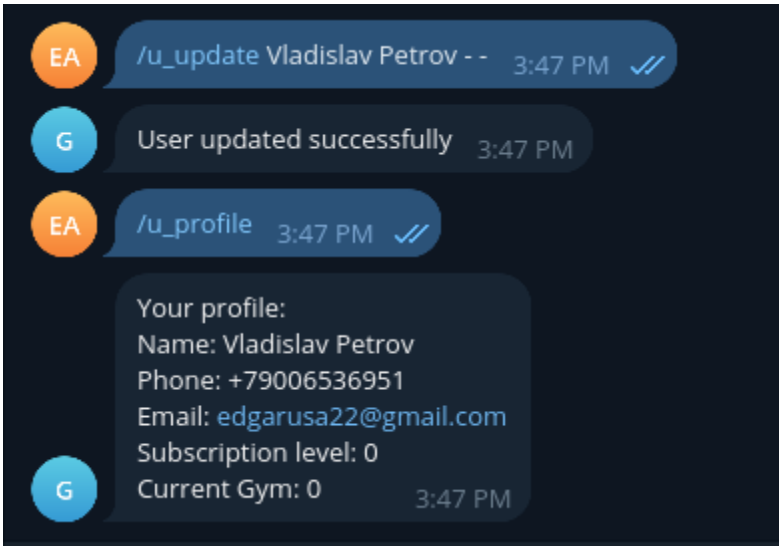

Рисунок 8 Ответ бота на команды /u_update и /u_profile

Команда /u_change_subscription

Данная команда нужна для того чтобы изменить уровень подписки при смене со счета спишется необходимая для уровня подписки сумма если её не будет, бот расскажет, что с помощью команды /w_deposit можно пополнить баланс. Это также показано на рисунке 9.

Команда /w_deposit

Данная команда нужна для того чтобы пополнить кошелек в приложении после этого отправится сообщение с актуальным балансом, в рамках курсовой работы кошелек не был привязан ни к какой платежной системе, однако даже сейчас можно воспользоваться всеми функциями кошелька. Демонстрация работы предоставлена на рисунке 9.

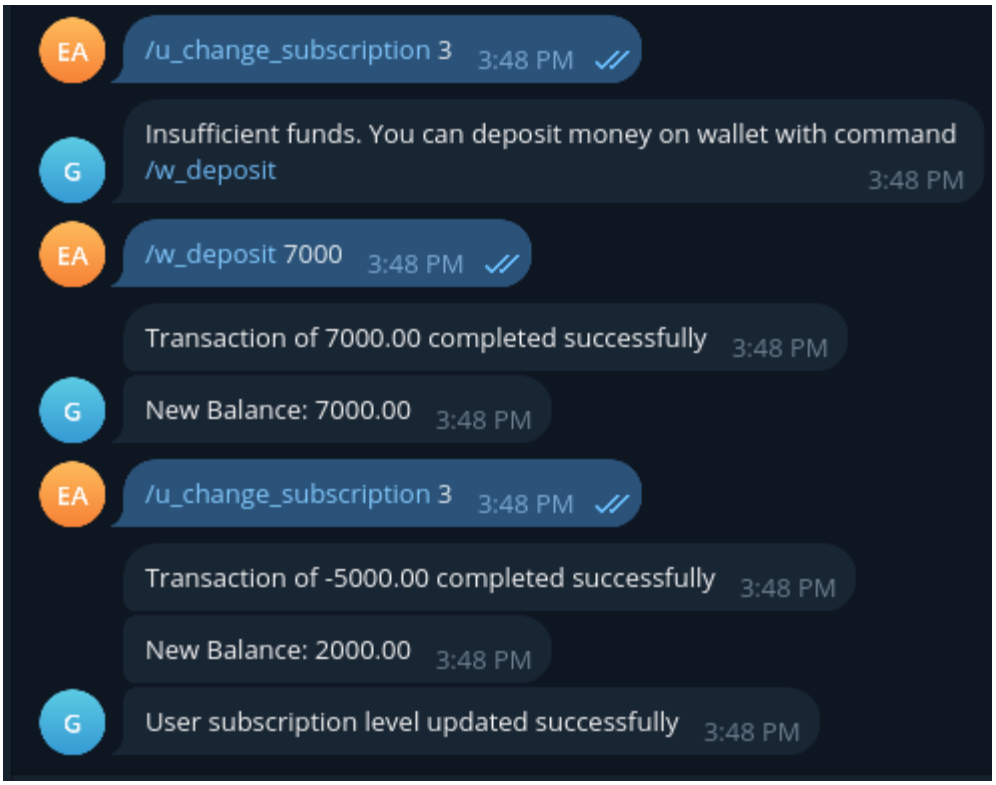

Рисунок 9 Ответ бота на команды /u_change_subscription и /w_deposit

Команда /g_list

Данная команда предоставляет ответ от бота, в котором прописаны все адреса, спортивных клубов в виде списка. Ответ бота на эту команду предоставлен на рисунке 10

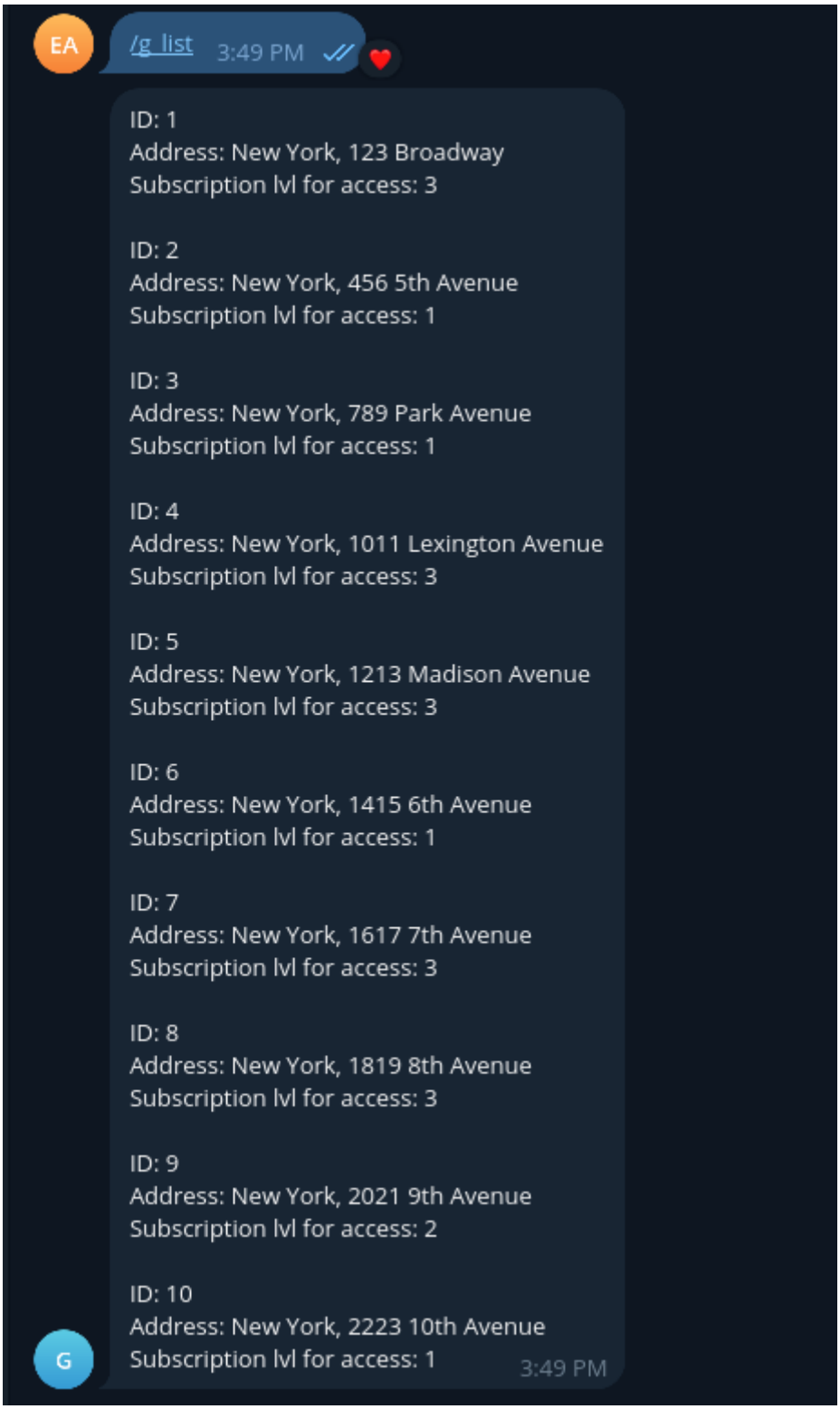

Рисунок 10 Ответ на команду /g_list

Команда /g_schedule

Данная команда нужна в приложении для предоставления расписания для определенного зала, она пишет день недели, время открытия и время закрытия. Это показано на рисунке 11

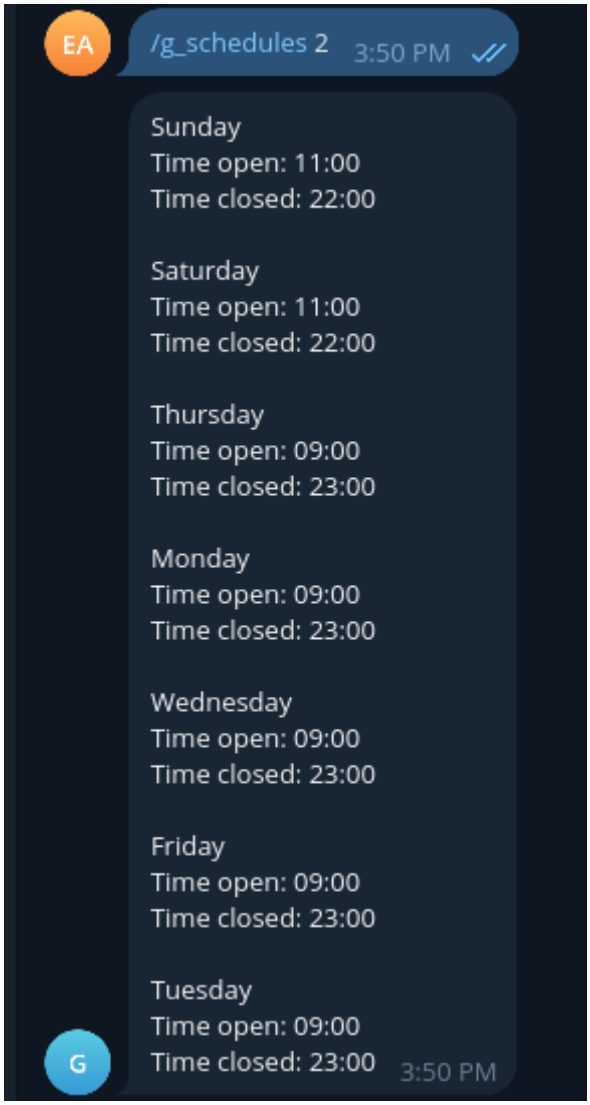

Рисунок 11 Ответ бота на команду /g_schedules

Команда /u_change_current_gym

Данная команда нужна боту для того, чтобы изменить текущий зал пользователя, она не даст этого сделать если уровень подписки ниже допустимого уровня зала, если же уровень выше, то зал будет изменен. Это показано на рисунке 12.

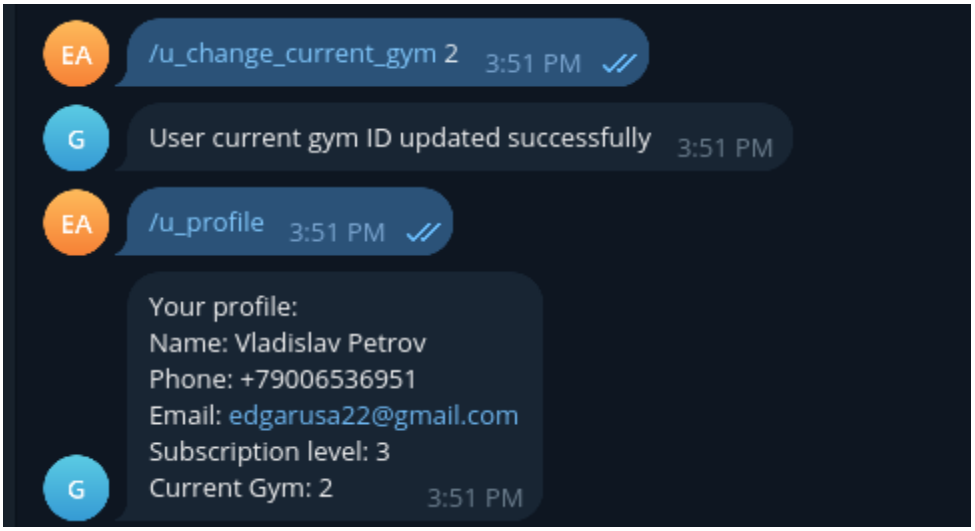

Рисунок 12 Ответ бота на команду /u_change_current_gym

Команда /w_balance

Данная команда нужна в приложении для того чтобы пользователь мог получить информацию о текущем балансе своего кошелька. Демонстрация работы показана на рисунке 13

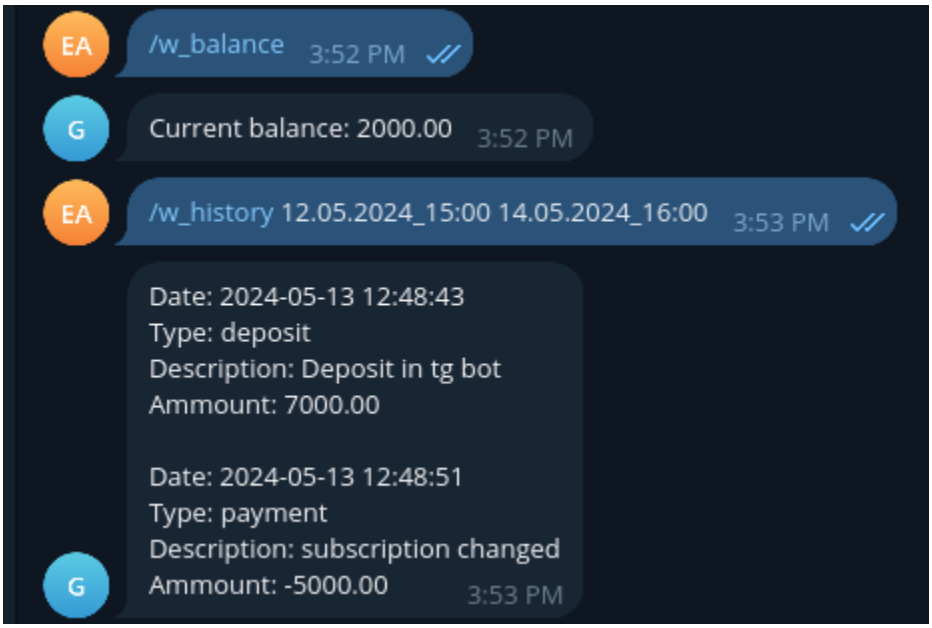

Рисунок 13 Ответ бота на команды /w_balance и /w_history

Команда /w_history

Данная команда нужна для получения информации о совершенных операциях за определенный период времени, о их типе, а также получить информацию о платеже. Это также предоставлено на рисунке 13.

Команда /s_data

Данная команда предоставляет информацию о уровне подписки, а также начало её действия и окончания. Ответ бота предоставлен на рисунке 14.

Команда /s_extend

Данная команда нужна для продления действия подписки на определенное количество месяцев, если для этого достаточно средств в кошельке. Демонстрация работы предосталена на рисунке 14.

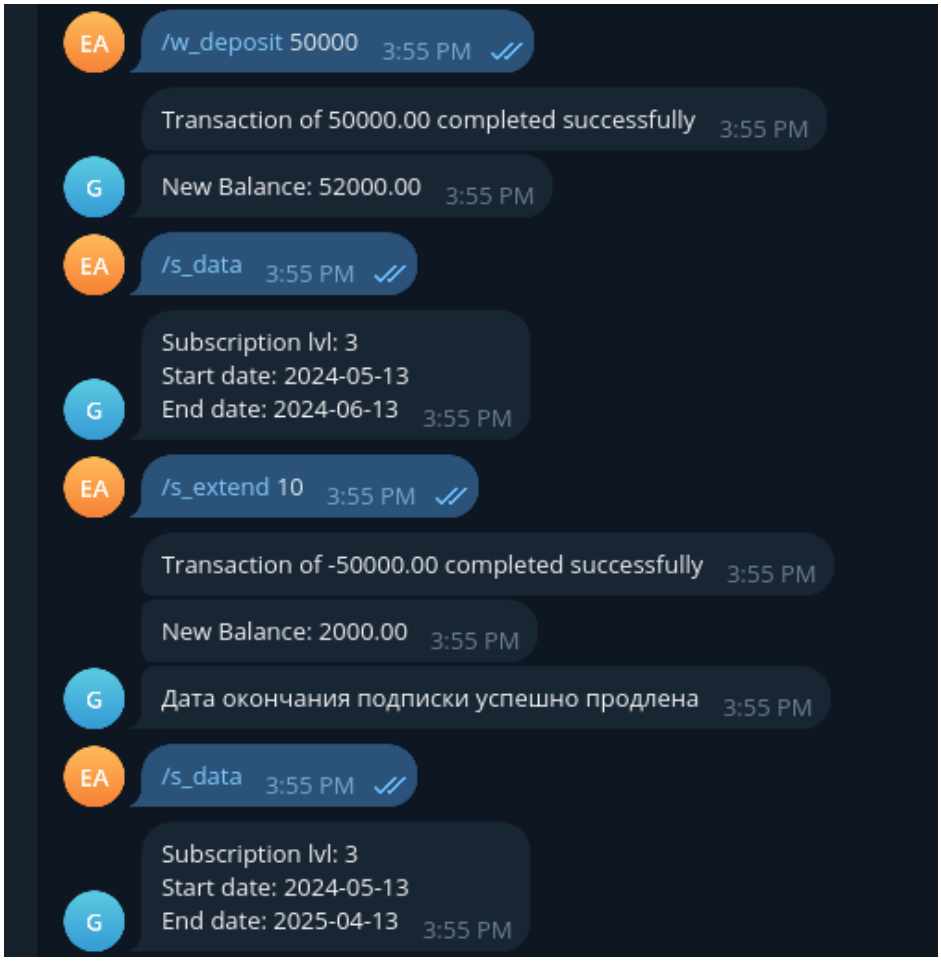

Рисунок 14 Ответы бота на команды /s_data и /s_extend

Команда /t_list_for_gym

Данная команда нужна для получения информации о тренерах для определенного зала. Бот отвечает списком из тренеров где для каждого из них есть уникальный id, имя, а также специальность. Пример предоставлен на рисунке 15

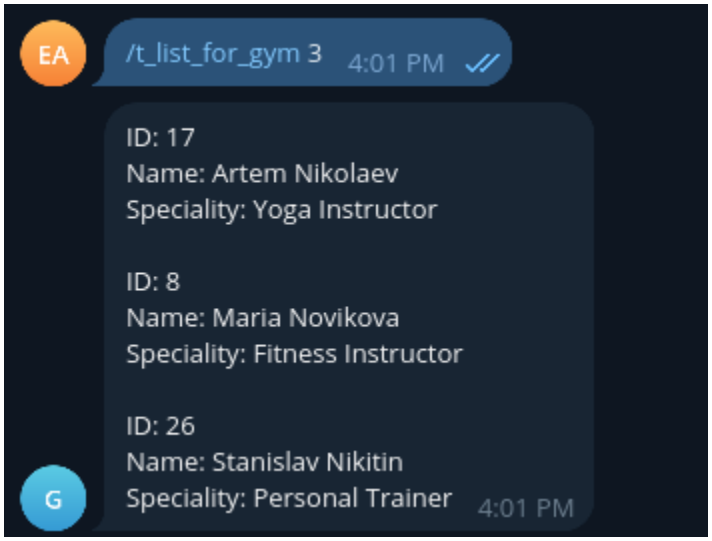

Рисунок 15 Ответ бота на команду /t_list_for_gym

Команда /t_available_booking_list

Данная команда показывает доступные слоты для определенного тренера, бот присылает список из слотов, в каждом из них есть уникальный id, название активности, время начала и конца. Демонстрация работы изображена на рисунке 16.

Команда /t_booking

Данная команда нужна для того, чтобы пользователи могли бронировать один из доступных слотов по id который получается с помощью команды /t_available_booking_list, при этом если на балансе будет недостаточно средств появится ошибка. Демонстрация работы предоставлена на рисунке 17.

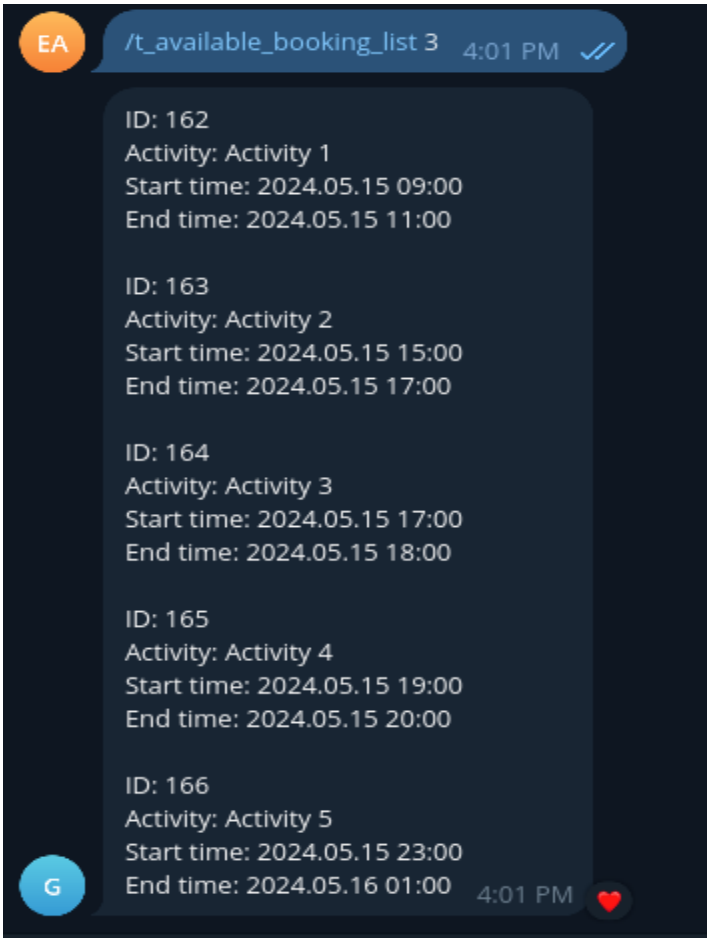

Рисунок 16 Ответ бота на команду /t_available_boking_list

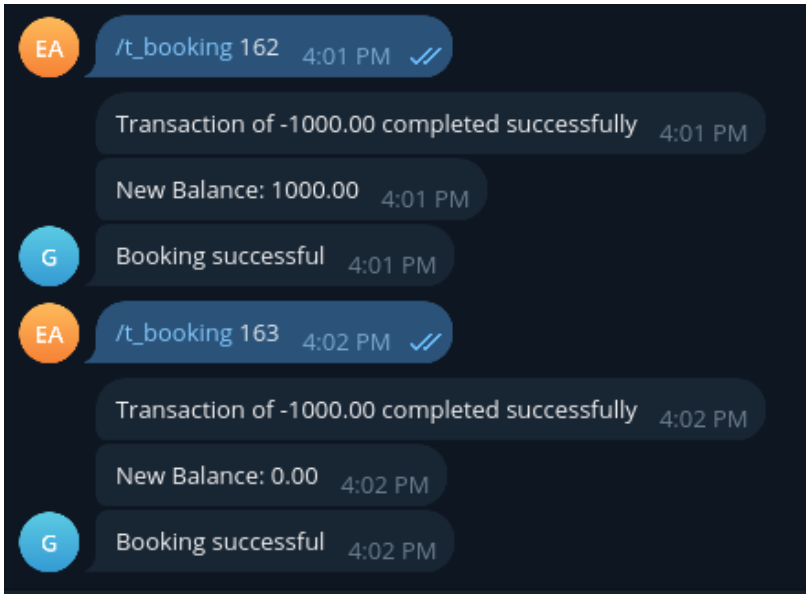

Рисунок 17 Ответы бота на команду /t_booking

Команда /t_current_booking_list

Данная команда показывает активные записи для пользователя, бот присылает список записей, в которых предоставлены уникальные id записей, название активностей, их начала и конца. Ответы предоставлены на рисунке 17

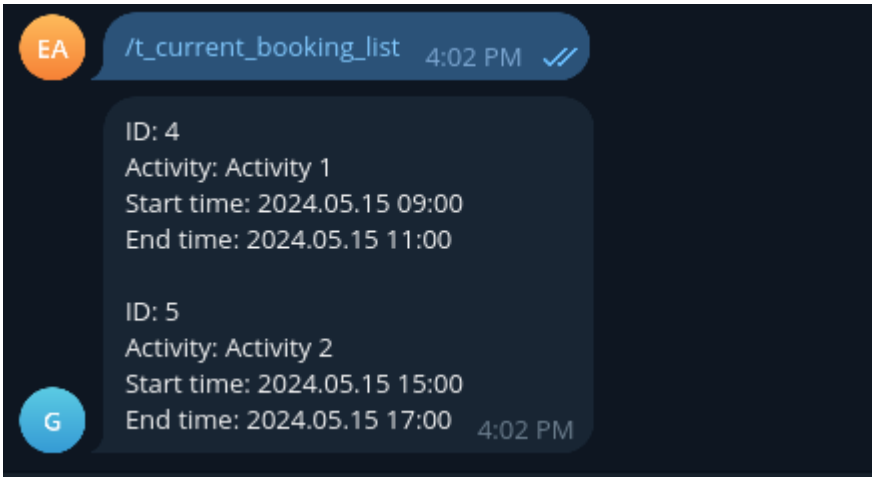

Рисунок 18 Ответы бота на команду /t_current_booking_list

Комадна /t_unboking

Данная команда нужна для того, чтобы пользователь мог отменить свою определенную запись к тренеру, по id его текущих записей, также при это плата за тренировку возвращается пользователю на счет. Демонстрация ответа бота предоставлена на рисунке 19.

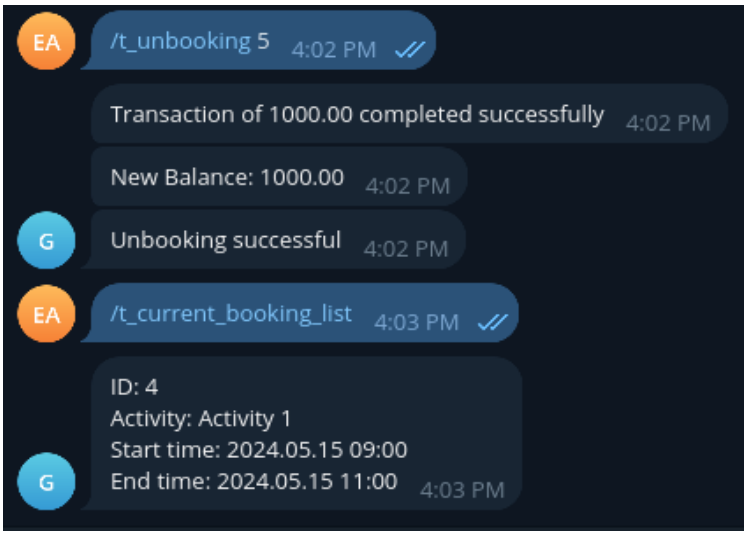

Рисунок 19 Ответы бота на команду /t_unbooking

Команда /support

Данная команда предоставляет ссылку на службу поддержки. Демонстрация работы показана на рисунке 19.

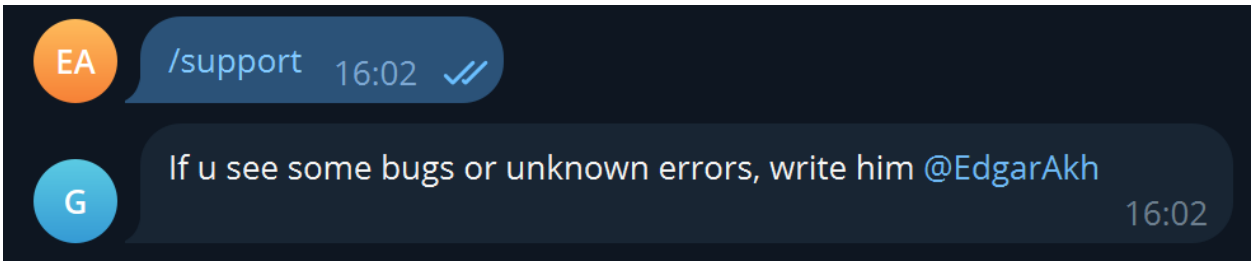

Рисунок 20 Ответы бота на команду /support

Все вышеуказанные команды демонстрируют работу бота и приложения в целом, почти в каждой команде telegram бот обращается к сервисной части для получения данных или их изменения. В будущем для работы с сервисной части можно подключить микросервис с веб-сайтом и это не повлияет на текущую работоспособность приложения.

2.6 Реализация микросервисов и демонстрация файлов Docker

Для реализации микросервисной архитектуры нашего приложения спортивного клуба мы использовали контейнеризацию с помощью Docker. Ниже на рисунке 21 приведен общий шаблон Dockerfile, который используется в каждом микросервисе.

```
# Используем официальный образ Go как базовый
FROM golang:1.21.0-alpine as builder

# Устанавливаем рабочую директорию внутри контейнера
WORKDIR /app

# Копируем исходники приложения в рабочую директорию
COPY . .
# Скачиваем все зависимости
RUN go mod tidy

# Собираем приложение
RUN go build -o main ./cmd

# Начинаем новую стадию сборки на основе минимального образа
FROM alpine:latest

# Добавляем исполняемый файл из первой стадии в корневую директорию контейнера
COPY --from=builder /app/main /main
COPY --from=builder /app/.env /.env


# Запускаем приложение
CMD ["/main"]

```

Рисунок 21 Шаблон Dockerfile для всех микросервисов.

Для эффективного управления размером образов Docker и оптимизации процесса развертывания мы используем двухэтапную сборку. Вот почему:

Минимизация размера образа:

- Первый этап сборки, основанный на образе Golang, включает в себя только необходимые зависимости и компилирует исходный код приложения. Это позволяет нам избежать включения в итоговый образ лишних компонентов, таких как инструменты сборки и исходные файлы.
- Второй этап, использующий минимальный образ Alpine Linux, только копирует исполняемый файл и необходимые файлы конфигурации из первого этапа. Это уменьшает объем итогового образа, так как Alpine Linux представляет собой легковесный дистрибутив.

Улучшение безопасности:

- Использование минимального образа Alpine Linux во втором этапе снижает поверхность атаки и уменьшает риски безопасности, так как в образе отсутствуют ненужные компоненты и службы.

Оптимизация времени сборки:

- Разделение процесса сборки на два этапа позволяет уменьшить время сборки образа. В первом этапе выполняется компиляция исходного кода, а во втором — создание минимального исполняемого образа, что сокращает общее время сборки.

Использование двухэтапной сборки в Docker помогает нам создавать более компактные, безопасные и быстрые образы, что облегчает процесс развертывания и управления нашими микросервисами.

Ниже предоставлены примеры docker-compose файлов для текущего приложения

На рисунке 22 представлен процесс развертывания сервиса тренеров. В контейнере trainer запущен сервер Go. Сервис взаимодействует с базой данных db_trainers PostgreSQL для хранения расписания тренировок.

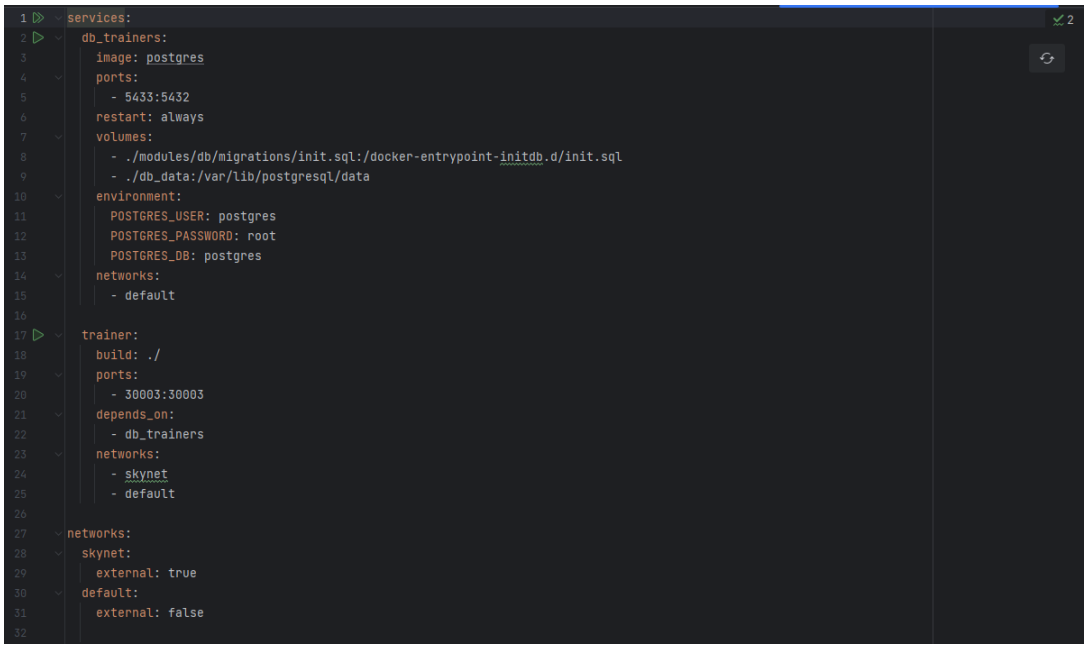

Рисунок 22 Развертывание сервиса тренеров.

На рисунке 23 показан процесс развертывания сервиса управления платежами. В контейнере используется сервер Go, который обрабатывает платежные операции. Для хранения данных о платежах используется база данных PostgreSQL.

```yaml
services:
  db_wallet:
    image: postgres
    restart: always
    volumes:
      - ./modules/db/migrations/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./db_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
      POSTGRES_DB: postgres
    networks:
      - default

  wallet:
    build: ./
    ports:
      - 30001:30001
    depends_on:
      - db_wallet
    networks:
      - skynet
      - default

networks:
  skynet:
    external: true
  default:
    external: false
```

Рисунок 23 Развертывание сервиса управления платежами

На рисунке 24 показан процесс развертывания сервиса telegram бота bot

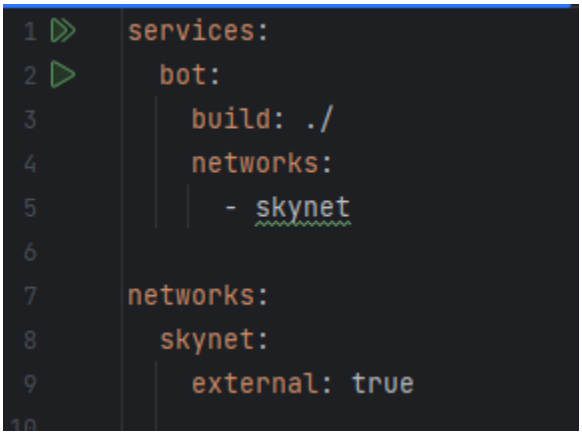

Рисунок 24 Развертывание сервиса telegram бота

2.7 Инструкция для развертывания приложения.

Для успешного развертывания приложения потребуется выполнить следующие шаги:

1. Клонирование репозитория:

Склонируйте репозиторий с микросервисами из Git на свой локальный компьютер.

2. Переход в директорию микросервиса:

В терминале перейдите в директорию каждого микросервиса с помощью команды cd.

3. Запуск скрипта сборки:

В каждой директории микросервиса запустите скрипт build.sh, который содержит необходимые команды для сборки образа Docker и запуска контейнера с помощью Docker Compose.

4. Ожидание завершения процесса:

Дождитесь завершения процесса сборки и запуска контейнера. После успешного выполнения всех шагов приложение будет доступно для использования.

5. Проверка работоспособности:

После завершения развертывания проверьте работоспособность приложения, перейдите в бота или выполните необходимые тесты для подтверждения корректной работы.

Эти простые шаги позволят быстро и эффективно развернуть приложение с помощью Docker Compose и начать его использование без лишних затрат времени и усилий.

Выводы к главе 2

При создании серверной части веб-приложения для спортивного клуба было выполнено следующее:

1. спроектирована структурная схема программного продукта
2. разработана сервисная часть, telegram бот и база данных веб-приложения
3. проведена апробация программного продукта среди студентов РГПУ им. А.И. Герцена;
4. разработана сопроводительная документация к программному продукту

В результате разработано полностью готовое к внедрению и тиражированию программное решение.

Заключение

При выполнении данной дипломной работы был разработан программный продукт «Микросервисное приложение для спортивного клуба», которое позволяет пользователям взаимодействовать с системами спортивного зала.

В процессе выполнения работы были решены следующие задачи:

- Проанализированы существующие потребности и проблемы в управлении спортивным клубом.
- Предложена концепция веб-приложения, учитывающая особенности спортивных клубов и их управления.
- Разработана серверная часть веб-приложения с использованием современных технологий.
- Протестировано разработанное веб-приложение и оценен его эффективность.

Таким образом, цель дипломной работы полностью реализована.

Одной из важнейших перспектив развития системы является интеграция с другими клиентскими веб-приложениями, в том числе разработка веб-сайта.

Кроме того, планируется разработка новых функций, например: сервис нотификации и улучшение микросервиса поддержки.

Литература

Книги

1.) Мартин Фаулер. "Микросервисы: архитектура будущего". – O'Reilly, 2016. – 256 с.
2.) Сэм Ньюман. "Building Microservices". – O'Reilly, 2015. – 208 с.
3.) Ричард К. Симпсон. "Designing Distributed Systems". – O'Reilly, 2018. – 256 с.
4.) Майкл Т. Фокс. "Clean Architecture: A Craftsman's Guide to Software Structure and Design". – Addison-Wesley, 2017. – 352 с.
5.) Роберт С. Мартин. "Clean Code: A Handbook of Agile Software Craftsmanship". – Prentice Hall, 2008. – 384 с.
6.) Мартин Фаулер. "Domain-Driven Design: Tackling Complexity in the Heart of Software". – Addison-Wesley, 2003. – 528 с.
7.) Джонатан Шварц. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation". – Addison-Wesley, 2010. – 448 с.
8.) Джонатан Хьюз. "Designing Data-Intensive Applications". – O'Reilly, 2017. – 528 с.
9.) Майкл Т. Фокс. "The Pragmatic Programmer: From Journeyman to Master". – Addison-Wesley, 1999. – 288 с.
10.) Джонатан Хьюз. "The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise". – O'Reilly, 2009. – 528 с.
11.) Джонатан Хьюз. "The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win". – IT Revolution, 2013. – 304 с
12.) Джонатан Хьюз. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations". – IT Revolution, 2016. – 528 с.

Электронные ресурсы:

1.) Martin Fowler's Blog - https://martinfowler.com/
2.) Microservices.io - https://microservices.io/
3.) Google Cloud Architecture - https://cloud.google.com/architecture/
4.) Distributed Systems for Fun and Profit - http://book.distributed-systems.net/
5.) The Twelve-Factor App - https://12factor.net/
6.) The Reactive Manifesto - https://reactivemanifesto.org/

7.) The SOLID Principles - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
8.) Clean Code Blog - https://blog.cleancoder.com/
9.) InfoQ - https://www.infoq.com/
10.)ThoughtWorks Tech Radar - https://www.thoughtworks.com/technology-radar

Приложения

ПРИЛОЖЕНИЕ 1

Telegram бот

UML ДИАГРАММА ВАРИАНТОВ (USE CASE)

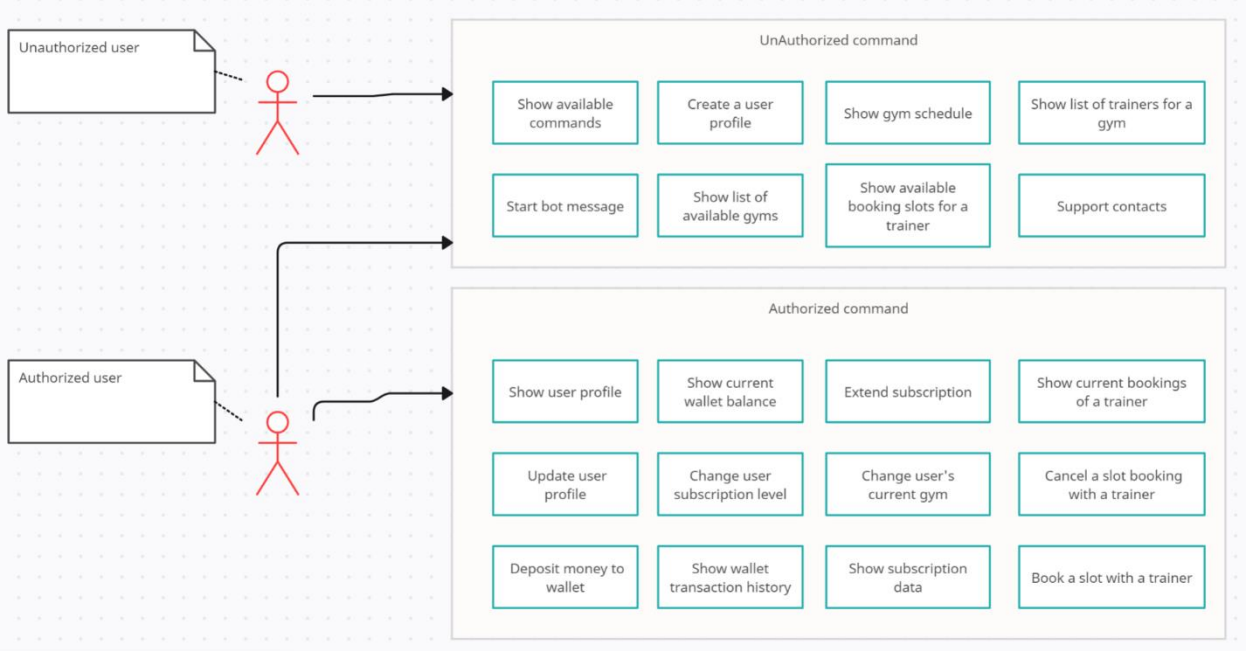

Рисунок 1 – Use Case схема доступа к функциональности веб-приложения

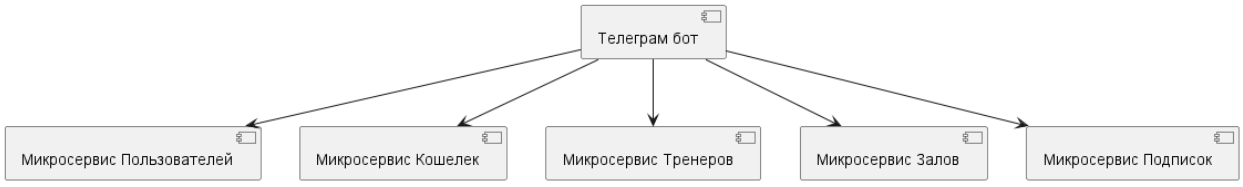

Рисунок 2 – UML-схема взаимодействия микросервисов

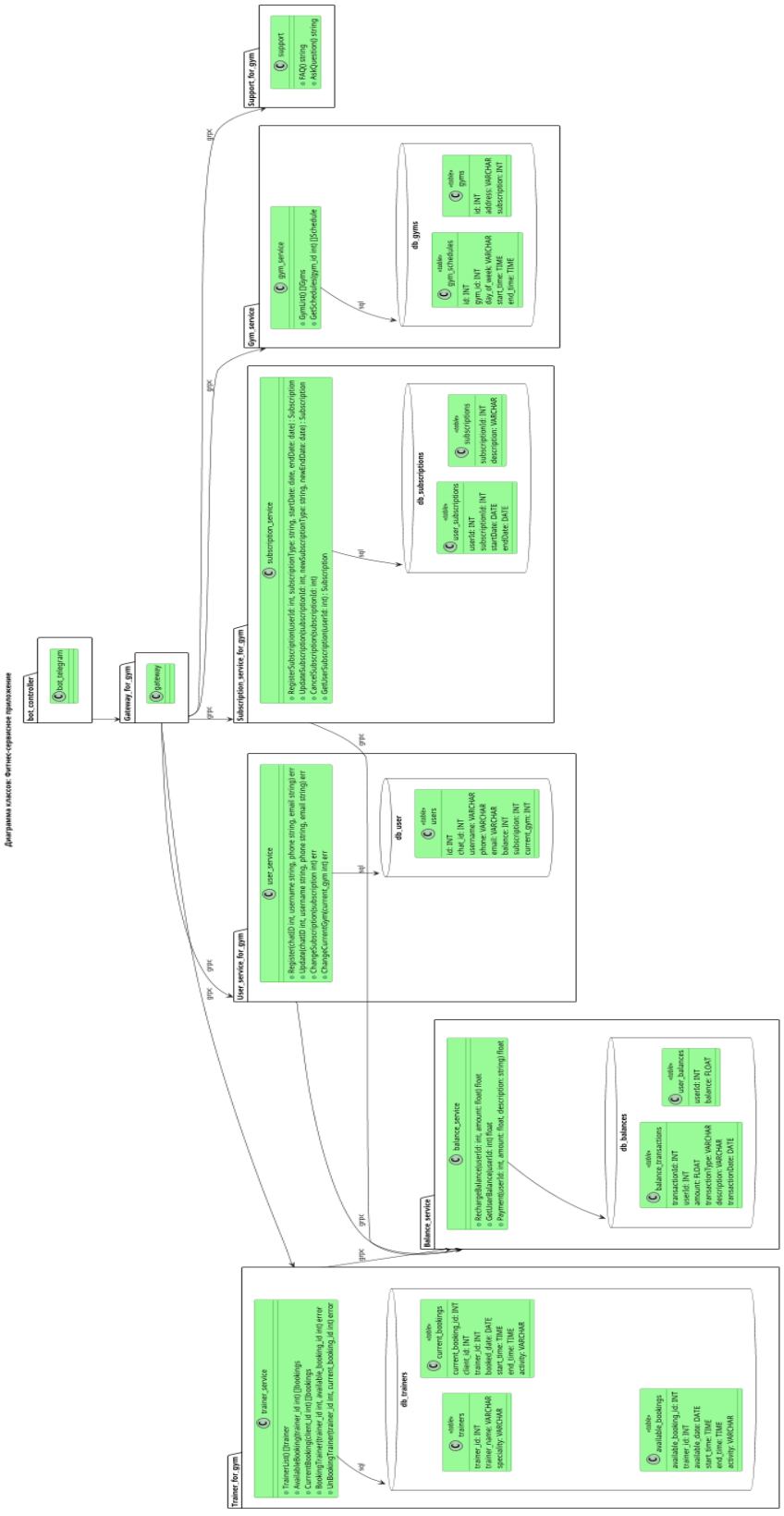

Рисунок 3 – UML-схема классов для спортивного клуб

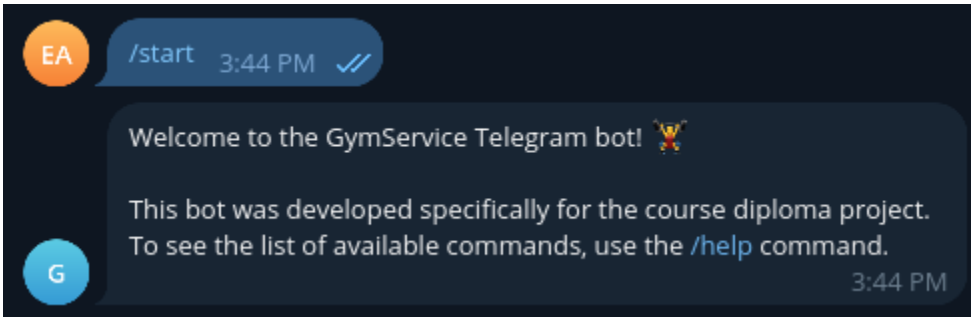

Рисунок 4 Ответ бота на команду /start

Рисунок 5 Ответ бота на команду /help

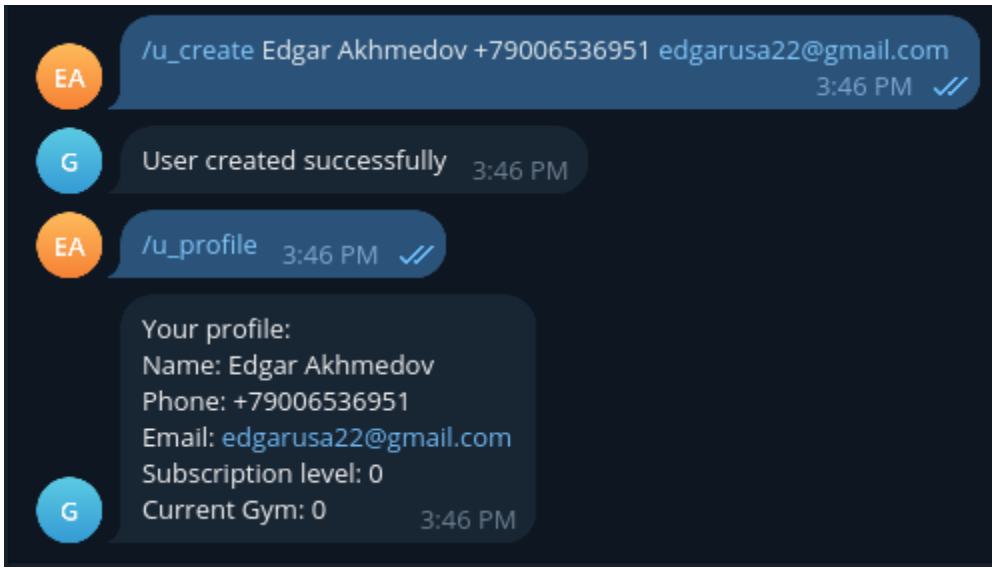

Рисунок 6 Ответы бота на команды /u_create и /u_profile

Рисунок 7 Неуспешный ответ бота на команду /u_profile

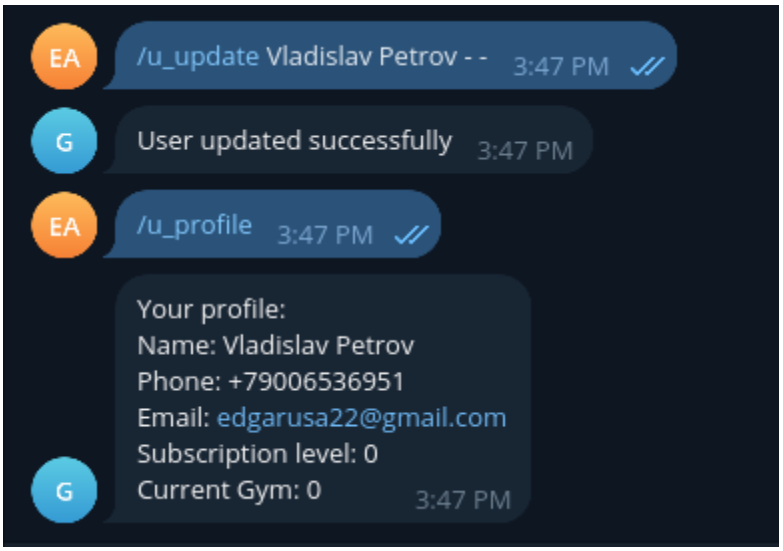

Рисунок 8 Ответ бота на команды /u_update и /u_profile

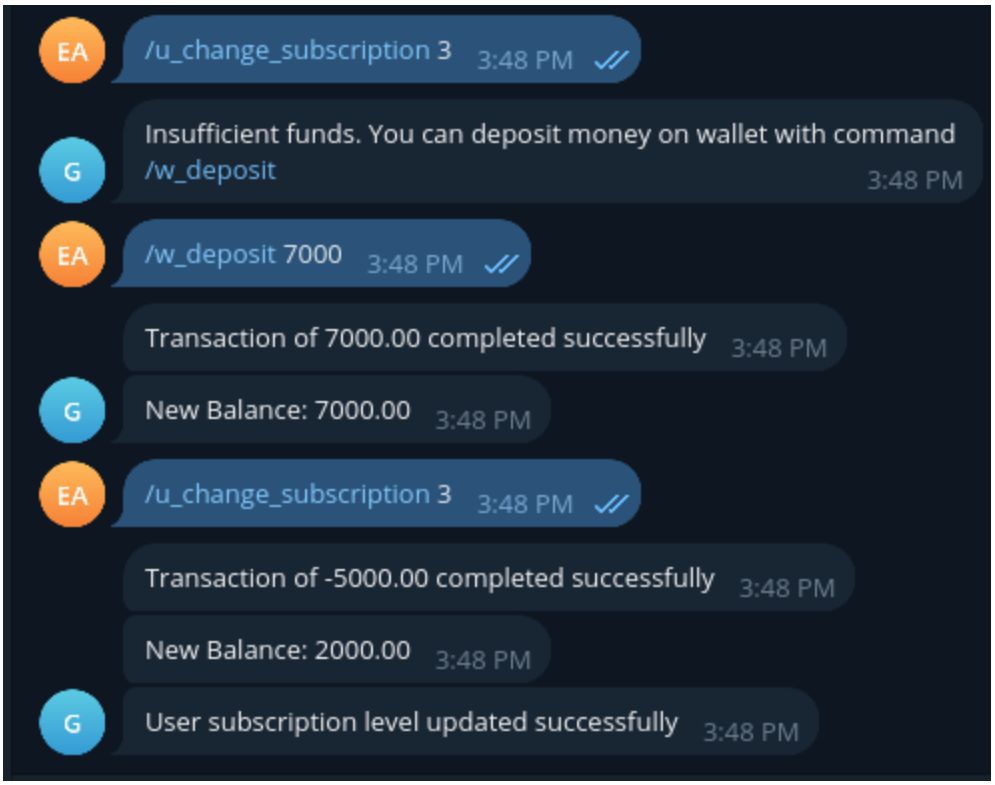

Рисунок 9 Ответ бота на команды /u_change_subscription и /w_deposit

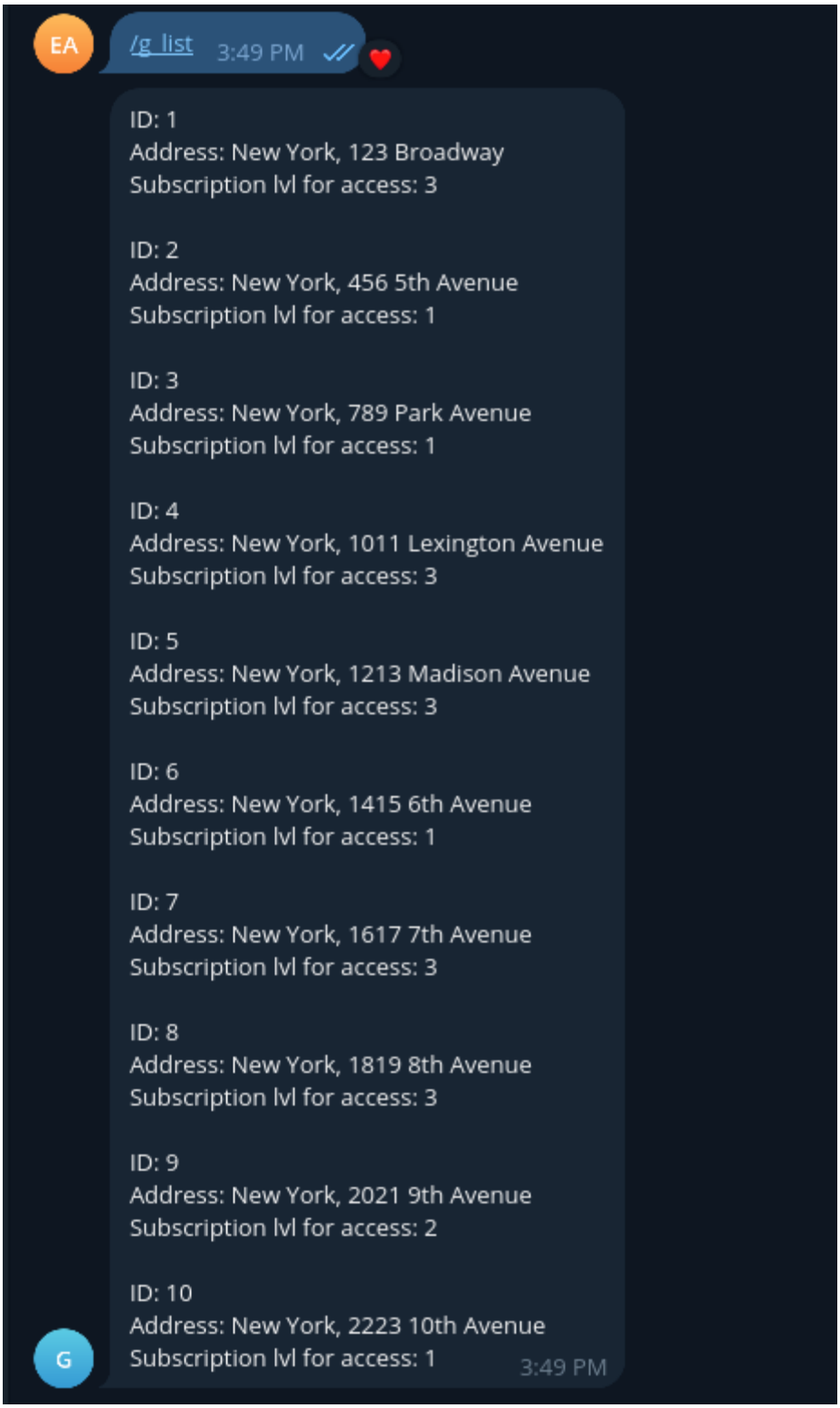

Рисунок 10 Ответ на команду /g_list

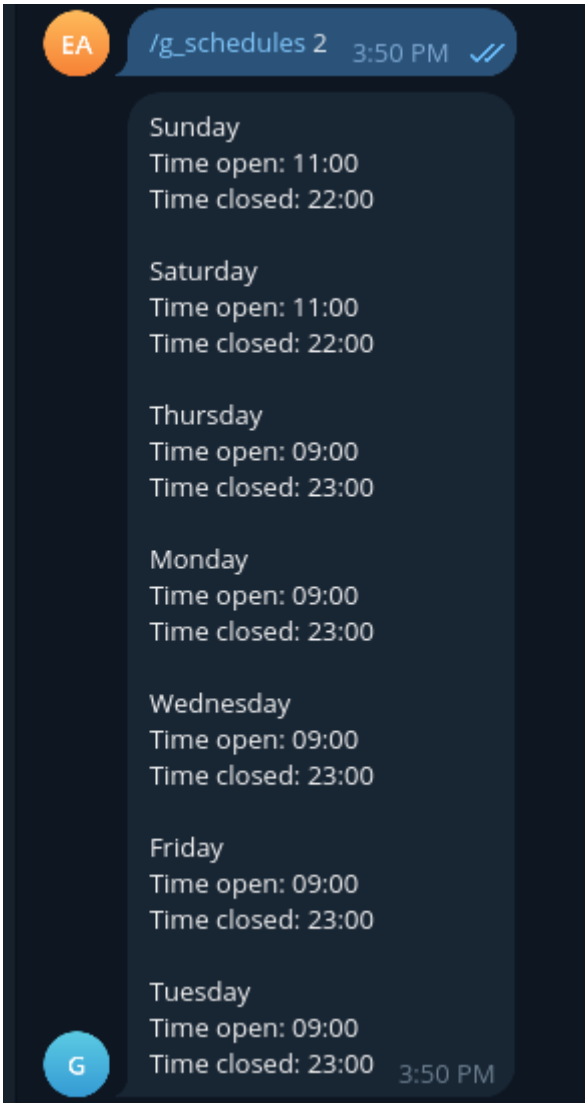

Рисунок 11 Ответ бота на команду /g_schedules

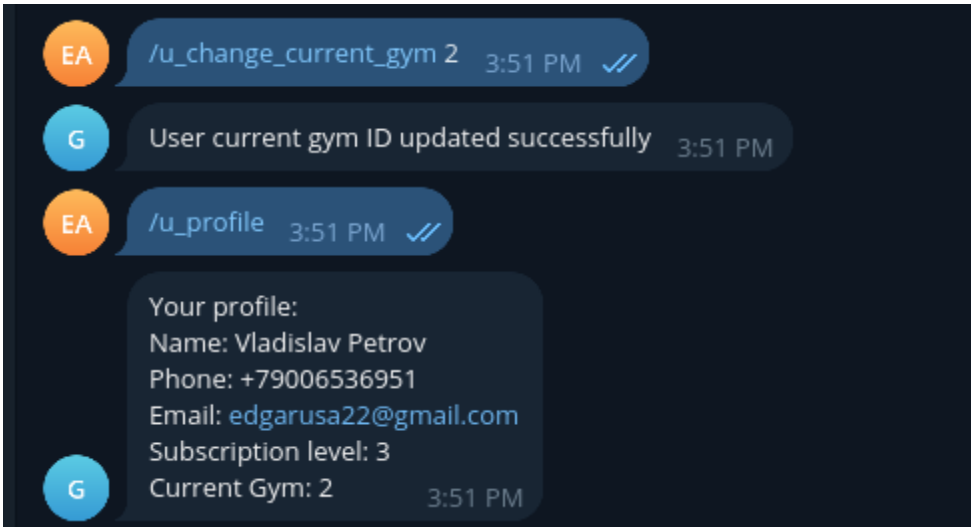

Рисунок 12 Ответ бота на команду /u_change_current_gym

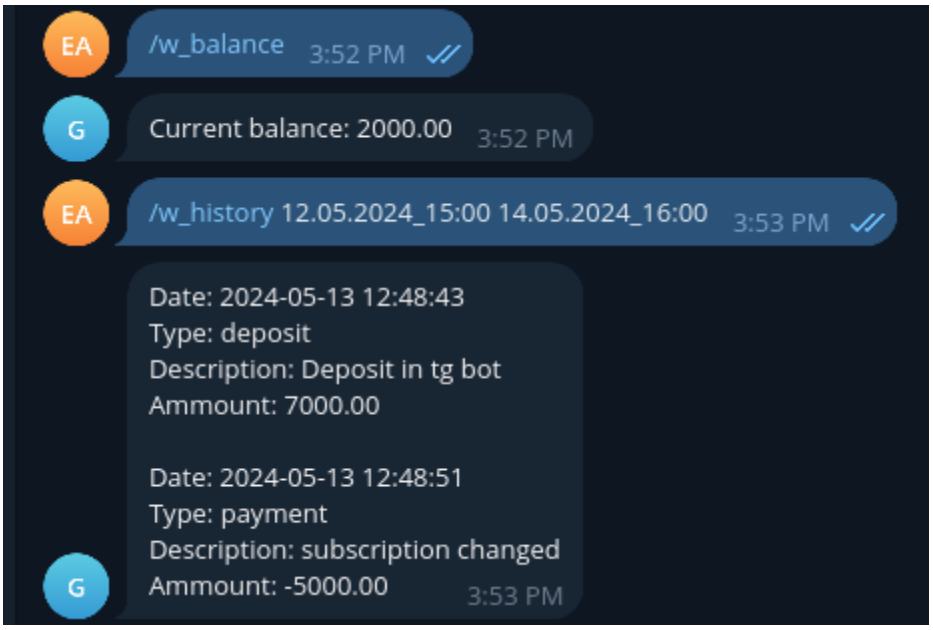

Рисунок 13 Ответ бота на команды /w_balance и /w_history

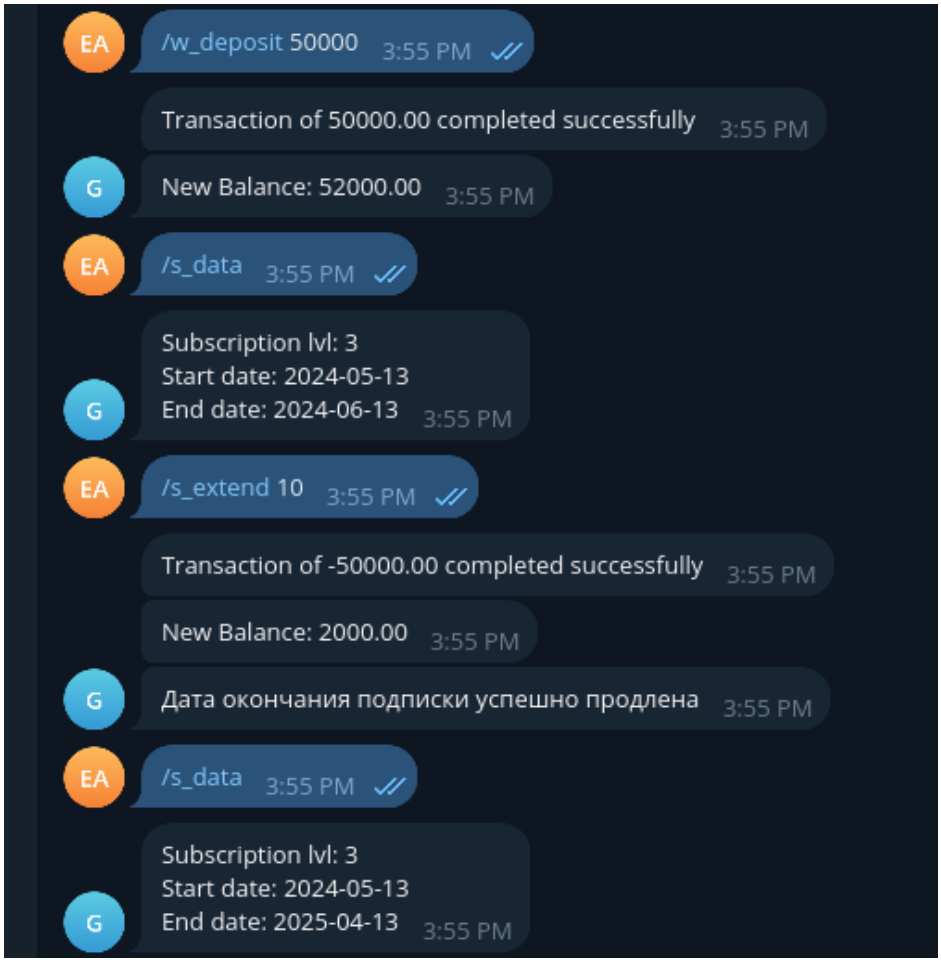

Рисунок 14 Ответы бота на команды /s_data и /s_extend

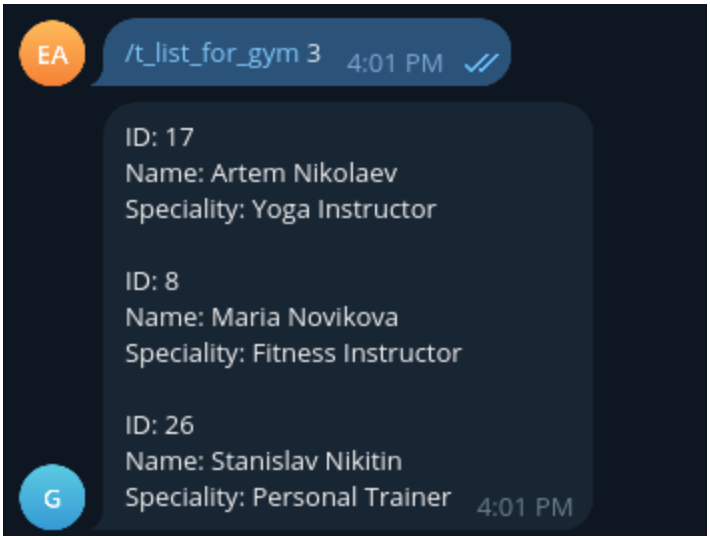

Рисунок 15 Ответ бота на команду /t_list_for_gym

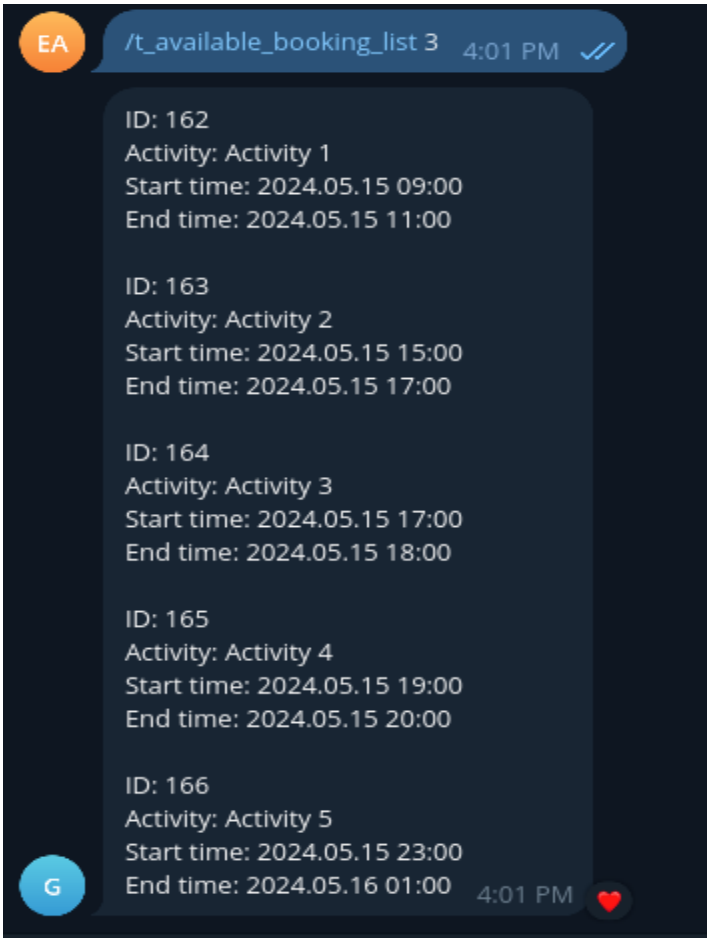

Рисунок 16 Ответ бота на команду /t_available_boking_list

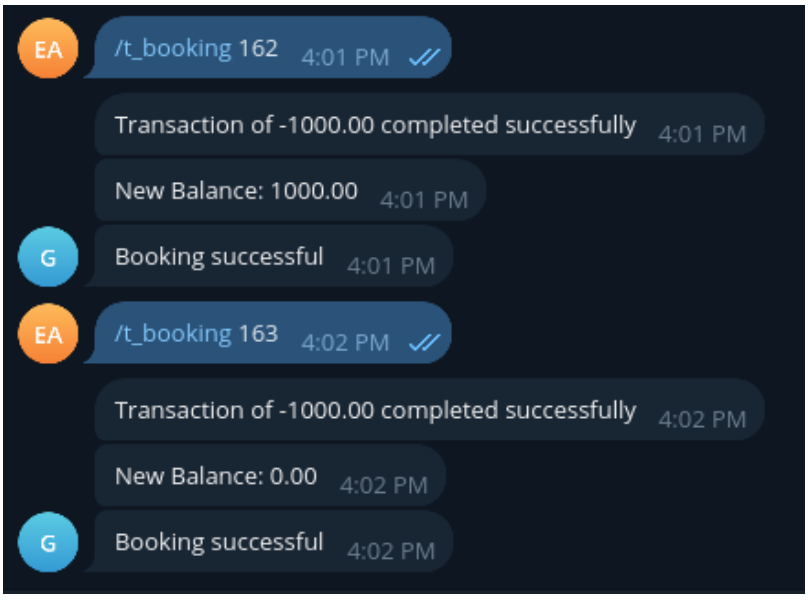

Рисунок 17 Ответы бота на команду /t_booking

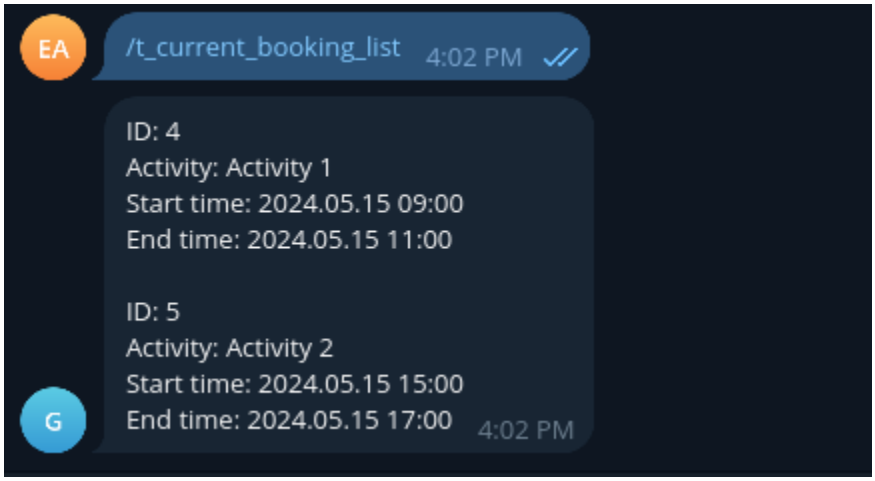

Рисунок 18 Ответы бота на команду /t_current_booking_list

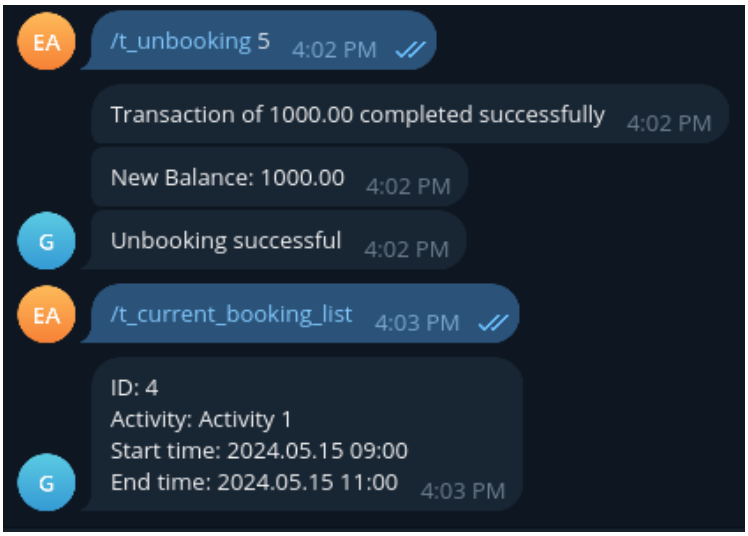

Рисунок 19 Ответы бота на команду /t_unbooking

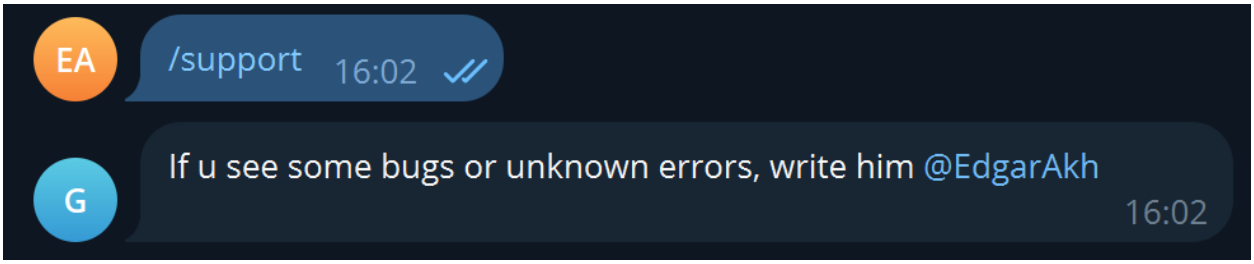

Рисунок 20 Ответы бота на команду /support

```dockerfile
# Используем официальный образ Go как базовый
FROM golang:1.21.0-alpine as builder

# Устанавливаем рабочую директорию внутри контейнера
WORKDIR /app

# Копируем исходники приложения в рабочую директорию
COPY . .
# Скачиваем все зависимости
RUN go mod tidy

# Собираем приложение
RUN go build -o main ./cmd

# Начинаем новую стадию сборки на основе минимального образа
FROM alpine:latest

# Добавляем исполняемый файл из первой стадии в корневую директорию контейнера
COPY --from=builder /app/main /main
COPY --from=builder /app/.env /.env


# Запускаем приложение
CMD ["/main"]

```

Рисунок 21 Шаблон Dockerfile для всех микросервисов.

```yaml
services:
  db_trainers:
    image: postgres
    ports:
      - 5433:5432
    restart: always
    volumes:
      - ./modules/db/migrations/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./db_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
      POSTGRES_DB: postgres
    networks:
      - default

  trainer:
    build: ./
    ports:
      - 30003:30003
    depends_on:
      - db_trainers
    networks:
      - skynet
      - default

networks:
  skynet:
    external: true
  default:
    external: false
```

Рисунок 22 Развертывание сервиса тренеров.

```yaml
services:
  db_wallet:
    image: postgres
    restart: always
    volumes:
      - ./modules/db/migrations/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./db_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: root
      POSTGRES_DB: postgres
    networks:
      - default

  wallet:
    build: ./
    ports:
      - 30001:30001
    depends_on:
      - db_wallet
    networks:
      - skynet
      - default

networks:
  skynet:
    external: true
  default:
    external: false
```

Рисунок 23 Развертывание сервиса управления платежами

```yaml
services:
  bot:
    build: ./
    networks:
      - skynet

networks:
  skynet:
    external: true
```

Рисунок 24 Развертывание сервиса telegram бота