


**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**


Факультет компьютерных наук  
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО

Руководитель, старший преподаватель  
департамента программной инженерии


  
\_\_\_\_\_  
« 16 » мая 2024 г. Н.А. Павлов

Соруководитель, приглашенный  
преподаватель департамента  
программной инженерии

  
\_\_\_\_\_  
« 16 » мая 2024 г. А.А. Топтунов

УТВЕРЖДАЮ

Академический руководитель  
образовательной программы  
«Программная инженерия» старший  
преподаватель департамента  
программной инженерии

  
\_\_\_\_\_  
« 16 » мая 2024 г. Н.А. Павлов

**Выпускная квалификационная работа  
(проектно-исследовательская)**

на тему: **Приложение для навигации в студенческих организациях  
Высшей Школы Экономики**

по направлению 09.03.04 «Программная инженерия»

ВЫПОЛНИЛА

студентка группы БПИ201  
образовательной программы  
09.03.04 «Программная инженерия»,

  
\_\_\_\_\_  
« 16 » мая 2024 г. Е.В.Дубровская

**Москва – 2024**

## **Реферат**

В Высшей школе экономики (ВШЭ) насчитывается более ста студенческих организаций, которые формируют активную и разнообразную внеучебную жизнь университета. Однако в настоящее время отсутствует единая платформа для обмена полной и актуальной информацией об этих сообществах, что усложняет развитие студенческих объединений. Такая нехватка информационных ресурсов затрудняет поиск студенческих организаций и препятствует вступлению в них новых участников.

Для упрощения взаимодействия студентов НИУ ВШЭ с университетскими сообществами был создан сервис "Higher search". Он предоставляет студентам эффективные инструменты для поиска подходящих сообществ, а организаторам - удобные средства для добавления и обновления информации о своих организациях. Разработка такой платформы потребовала решения ряда сложных задач, таких как создание быстрой, надежной и интуитивно понятной системы, которая бы отвечала потребностям всех пользователей.

**Работа содержит:** 31 страниц, 3 главы, 14 иллюстраций и 24 источников.

**Ключевые слова:** IT, Golang, backend-разработка, frontend-разработка, клиент-серверная архитектура.

## **Abstract**

The Higher School of Economics (HSE) has over a hundred student organizations that form an active and diverse extra-curricular life for the university. However, there is currently no centralized platform for sharing comprehensive and up-to-date information about these communities, which hinders the development of student organizations. This lack of information resources complicates the discovery of student groups and prevents new members from joining.

To address this challenge, "Higher Search" was developed as a service that aims to facilitate access to HSE student groups and provide convenient tools for students searching for a suitable community, as well as organizers adding and updating group information. Designing such a platform involves a range of complex tasks, including creating a fast, reliable, and intuitive system that meets the needs of all users. This article will discuss the steps necessary for launching the project, from analyzing requirements and collecting metrics to catching errors.

**The work contains:** 31 pages, 3 chapters, 14 figures и 24 sources

**Keywords:** IT, Golang, client-server, backend, frontend

## Содержание

Реферат.....	1
Abstract.....	2
<b>Определения, обозначения и сокращения.....</b>	<b>4</b>
<b>Введение.....</b>	<b>5</b>
<b>Глава 1. Обзор предметной области и анализ существующих решений.....</b>	<b>7</b>
1.1. Анализ существующих решений.....	7
1.1.1 Экстра.....	7
1.1.2. Сообщества Вконтакте и Telegram-каналы.....	8
1.1.3. Списки сообществ на официальном сайте ВШЭ.....	9
1.2. Постановка задачи и формирование списка требований.....	10
1.3. Выводы по главе.....	10
<b>Глава 2. Архитектура приложения и методология разработки.....</b>	<b>11</b>
2.1. Клиент-серверная архитектура.....	11
2.1.1. Frontend-разработка.....	12
2.1.2. Backend-разработка.....	12
2.1.3. Взаимодействие клиентской и серверной части.....	14
2.1.4. Devops-методология.....	15
2.2. Составные части системы.....	15
2.3. Выводы по главе.....	16
<b>Глава 3. Особенности программной реализации и результаты работы.....</b>	<b>17</b>
3.1. Бизнес-логика проекта.....	17
3.1.1. Личный кабинет. Функционал регистрации и авторизации.....	18
3.1.2. Система навигации.....	21
3.1.3. Система модерации.....	21
3.1.4. Страницы-визитки студенческих организаций: просмотр и администрирование.....	22
3.2. Технические аспекты.....	23
3.2.1. Protobuf-модели.....	23
3.2.2. Использование Protobuf-моделей со стороны серверной части приложения.....	23
3.2.3. Использование Protobuf-моделей со стороны frontend-a.....	24
3.2.4. Инициализация базы данных.....	24
3.2.5. Механизм авторизации пользователей.....	25
3.2.6. Хранение паролей.....	26
3.2.7. Клиентская часть.....	26
3.2.8. Логирование.....	27
3.3. Выводы по главе.....	27
<b>Заключение.....</b>	<b>28</b>
<b>Список использованных источников.....</b>	<b>29</b>

## **Определения, обозначения и сокращения**

**Студенческая организация, СО** – объединение студентов Высшей Школы Экономики, то же, что студенческое сообщество. СО, представленные на сайте, могут обладать или не обладать статусом официальной студенческой организации ВШЭ.

**Студенческое сообщество** – то же, что и студенческая организация.

**Центр поддержки студенческих инициатив, ЦПСИ** – подразделение ВШЭ, ответственное за работу со студенческими организациями.

## **Введение**

В настоящее время не существует единого сервиса, который позволит найти любую студенческую организацию ВШЭ и разместить рекламу или ссылку на ресурсы своей организации. Студенческие организации обычно выкладывают всю необходимую информацию (анонсы мероприятий, контакты организаторов, etc) в сообществах VK / Telegram-каналах, тем самым увеличивая количество источников, которые необходимо просмотреть, чтобы найти нужную организацию. За счет такого разрозненного хранения информации повышается порог входа в новые, малоизвестные организации; им не хватает единого публичного ресурса, на котором каждый студент мог бы создать и прорекламировать свою организацию, предоставить удобные условия для ее поиска.

В качестве решения этой проблемы предлагается создать единую платформу для навигации в студенческих организациях. Основным отличием этого проекта от существующих решений будет возможность организаторов самим обновлять информацию о своих сообществах – только эта опция позволит гарантировать полноту и актуальность данных.

Таким образом, цель настоящей выпускной квалификационной работы заключается в разработке сервиса, который предоставит удобные инструменты для поиска студенческих организаций, а также для их создания и управления.

Для достижения данной цели были поставлены следующие задачи:

1. Обзор и анализ существующих решений и аналогов;
2. Выбор и обоснование средств и методов разработки;
3. Проектирование архитектуры приложения;
4. Создание визуального прототипа приложения;
5. Разработка графического пользовательского интерфейса;
6. Разработка серверной части приложения;
7. Настройка системы логирования;
8. Настройка развертывания системы;
9. Разработка технической документации приложения.

Настоящая работа имеет следующую структуру:

В Главе 1 находится обзор и анализ смежных решений и аналогов разрабатываемого сервиса, выводятся нефункциональные требования.

В Главе 2 рассматривается архитектура приложения, особенности ее проектирования и реализации, методологии разработки.

В Главе 3 приводится описание программной реализации и продемонстрированы результаты работы.

В заключении приведены выводы по результатам проделанной работы и описан план дальнейшего развития проекта.

## **Глава 1. Обзор предметной области и анализ существующих решений**

В этой главе будут рассмотрены наиболее популярные инструменты, используемые студенческими организациями для размещения информации. Анализ недостатков и преимуществ поможет продумать пользовательские сценарии и проработать требования к проекту.

### **1.1. Анализ существующих решений**

Самым близким аналогом разрабатываемого продукта являлся сервис HSE.Family (<https://family.hse.ru/>). К моменту разработки и последние несколько лет сервис недоступен, и полноценно его проанализировать уже невозможно. В качестве альтернативы используются как официальные ресурсы Высшей Школы Экономики, так и сторонние платформы.

#### **1.1.1 Экстра**

Экстра — бренд внеучебной жизни, объединяющий все сервисы, проекты и продукты Вышки, которые развивают и поддерживают творческие, научные и профессиональные идеи студентов [1]. Экстра представляет собой крупнейшую медиа платформу студенческих сообществ ВШЭ и предоставляет доступ к подробному календарю мероприятий большого количества студенческих организаций. Рассылка, социальные сети и непосредственно календарь Экстра в текущий момент очень востребованы как источники информации о мероприятиях СО Вышки.

С точки зрения проблемы навигации в студенческих сообществах сервис Экстра не предоставляет полноценного решения. На сайте отсутствует полный список организаций, не предоставлены удобные инструменты для поиска, нет функционала для создания новых СО. Также недостатком является тот факт, что единственный способ распространить информацию об уже существующих организациях – регулярно вносить их события в календарь. Если же организация не проводит мероприятий в ближайшее время, узнать о ней через сервис Экстра невозможно.

Основной функцией Экстра является информирование о предстоящих мероприятиях студенческих организаций, тогда как цель разрабатываемого продукта – создать инструмент для удобного поиска СО. Эти сервисы дополняют, а не заменяют друг друга. Более наглядное сопоставление характеристик приведено в таблице 1.



Таблица 1: Сравнение функционала Экстра и разрабатываемого приложения

Экстра	Higher Search
Предоставляет календарь для поиска предстоящих мероприятий.	Предоставляет функционал для поиска студенческих организаций.
Предоставляет полный список мероприятий.	Предоставляет полный список студенческих организаций.
Позволяет рассказать о своем мероприятии в рассылке или внести его в календарь.	Позволяет выложить информацию о существующей студенческой организации или создать новую.
Дает возможность зарегистрироваться на мероприятие.	Дает возможность вступить в студенческую организацию.

Таким образом, сервис Экстра и разрабатываемый продукт не конкурируют с друг другом.

### 1.1.2. Сообщества Вконтакте и Telegram-каналы

Многие студенческие сообщества ВШЭ размещают информацию о мероприятиях на платформе Вконтакте [2]. Сообщества Вконтакте – удобный инструмент для отображения всех необходимых данных, таких как ссылок на мероприятия, архивов с фотографиями и прочее. Однако, несмотря на то, что во Вконтакте присутствует собственная система для навигации в сообществах, она не решает целиком поставленную задачу. Для того, чтобы найти организацию по запросу, пользователю приходится подбирать названия, пытаться угадать, как бы организаторы назвали соответствующее сообщество. Не все организаторы указывают университет в тегах или названии сообщества, отсутствует унифицированная система наименований, и это тоже усложняет поиск. Хотя ВКонтакте на текущий момент является наиболее удобной площадкой для активной деятельности сообщества (сервис предоставляет возможность публиковать посты, размещать опросы, комментировать и общаться в чатах), ВКонтакте не является подходящим инструментом для поиска, потому что не предоставляет полного и актуального списка организаций.

Другой популярной платформой для ведения деятельности СО является Telegram. Поскольку аудитория мессенджера не повторяет целиком аудиторию Вконтакте, некоторые организации поддерживают сразу несколько информационных ресурсов, на разных платформах. Такое разрозненное хранение информации существенно осложняет задачу поиска – студенту, желающему найти себе студенческую организацию с конкретной тематикой, необходимо подбирать и угадывать названия сразу на нескольких ресурсах.

И Вконтакте, и Telegram предоставляют удобные инструменты для ведения деятельности сообществ, и разрабатываемый сервис в текущий момент не может стать их альтернативой. Однако роль Higher Search не в этом – общая платформа должна представлять собой агрегатор, то есть сервис, объединяющий информацию из разных источников, и предоставляющий возможность перейти на них.

### **1.1.3. Списки сообществ на официальном сайте ВШЭ**

Некоторые страницы официального сайта ВШЭ также выполняют роль агрегаторов. Их ключевой недостаток заключается в том, что информация на них очень быстро становится неактуальной.

К примеру, наиболее популярная страница по теме “Студенческие сообщества ВШЭ” – это ссылка на “Почемучник” (брошюра для первокурсников) 2018 года [3]. К текущему времени эта информация устарела на 6 лет – большинство участников тех сообществ уже выпустились из университета. Устаевают контакты, меняются ссылки на актуальные источники информации, перестают существовать маленькие организации и появляются новые, а информация на сайте остается той же.

Другие списки расположены на страницах факультетов или кампусов, и содержат только локальные студактивы [4]. Информация на таких страницах также статична, и изменять ее могут только сотрудники университета. При этом общепринятого механизма, по которому можно сообщить об устаревании информации и гарантировать изменение данных сейчас нет.

Статичные списки сообществ никогда не могут соответствовать двум требованиям: актуальности и полноты данных. Их выполнение может достигаться только за счет возможности студентов самостоятельно обновлять информацию о сообществах. На официальном сайте Вышки такой возможности у них нет.

## **1.2. Постановка задачи и формирование списка требований**

Опираясь на анализ существующих решений, для новой платформы можем сформулировать требования об актуальности и полноте информации.

Полнота информации о студенческих организациях означает, что все студенческие организации Высшей Школы Экономики должны иметь возможность разместить информацию на платформе. Должен быть реализован функционал для добавления новых сообществ, сохранение информации в базе данных. Новые организации должны быть доступны для просмотра остальным пользователям.

Требование об актуальности информации означает, что организаторы студенческих сообществ должны быть способны изменять информацию, уже размещенную на сайте. Здесь появляется необходимость в системе аутентификации – у пользователей должна быть возможность регистрироваться и авторизовываться на платформе, чтобы за каждой студенческой организацией сохранялись пользователи, которые могут ее редактировать.

Пользователи также должны иметь способ сохранять себе понравившиеся организации – это необходимо для того, чтобы отслеживать изменения в сообществах.

## **1.3. Выводы по главе**

В данной главе были рассмотрены текущие альтернативы разрабатываемому проекту. На основе сделанного анализа, можно сделать вывод, что ни одно из современных решений не обладает необходимыми качествами для платформы по навигации. Если данные размещены на официальном портале ВШЭ, обычные студенты не могут их изменять, из-за чего не выполняется требование об актуальности информации. При свободном выборе платформы студенты используют те инструменты, которые им удобны и привычны, и информация хранится разрозненно – не выполняется требование о полноте. Предлагаемое решение будет обладать обеими характеристиками за счет возможности студентов самостоятельно размещать и обновлять данные.

Необходимо учитывать, что новая платформа не требует полного ухода со старых площадок – она является агрегатором, то есть позволяет найти информацию, размещенную сразу в нескольких местах, и при необходимости перейти на более удобную площадку – такую как VK, Telegram или личный сайт сообщества.

## **Глава 2. Архитектура приложения и методология разработки**

В этой главе мы подробно рассмотрим архитектуру разрабатываемого приложения, включая основные компоненты системы и принципы их взаимодействия. Также будет представлен стек технологий, используемых в проекте – языки программирования, фреймворки, базы данных и другие инструменты, определяющие техническую основу приложения.

### **2.1. Клиент-серверная архитектура**

Веб-приложения, как правило, реализуют классическую клиент-серверную архитектуру, которая обеспечивает эффективное и масштабируемое взаимодействие между клиентскими устройствами и серверными ресурсами. Одним из основных преимуществ клиент-серверной архитектуры является разделение ответственности между клиентской и серверной частями. Клиент отвечает за взаимодействие с пользователем, отправку запросов и отображение результатов, сервер же берет на себя основную логику обработки данных, хранение информации и предоставление ответов клиентам.

Благодаря разделению ролей, веб-приложения на основе клиент-серверной архитектуры легко масштабируются. Серверная часть может быть распределена на несколько мощных машин, способных обрабатывать большие объемы запросов, в то время как клиентские устройства остаются относительно простыми. Это позволяет обеспечить высокую доступность и производительность приложения даже при значительном увеличении числа пользователей.

В текущем проекте также было решено использовать именно клиент-серверную модель архитектуры. Разработка подобного приложения включает в себя 4 ключевых процесса:

1. Frontend – разработка клиентской части и проектирование веб-интерфейса;
2. Backend – разработка серверной части, взаимодействие с базой данных;
3. DevOps – deploy проекта, контейнеризация;
4. Тестирование – написание unit- и интеграционных тестов, тестирование веб-интерфейса, логирование ошибок.

Эти процессы сильно завязаны друг на друга, поэтому выполняются одновременно.. Добавление новой функциональности включает в себя изменение в каждом отдельном

блоке. Далее мы рассмотрим список технологий и подходов к разработке в каждом отдельном процессе.

### **2.1.1. Frontend-разработка**

Для создания клиентской части платформы в проекте используется набор передовых технологий и методологий, обеспечивающих эффективную и качественную разработку пользовательского интерфейса.

Для установки, управления и обновления зависимостей проекта используется Node.js [5]. В качестве инструмента сборки выбран Vite [6] - быстрый и производительный инструмент, который позволяет разработчикам получать мгновенную обратную связь при внесении изменений в код. Это способствует повышению скорости разработки и улучшению общей эффективности процесса.

Для структурирования и организации frontend-части приложения применяется фреймворк Remix [7]. Он предоставляет удобные механизмы для управления навигацией между страницами, а также для эффективного управления состоянием приложения.

В качестве основной библиотеки для построения пользовательского интерфейса используется React [8]. Эта популярная JavaScript-библиотека обеспечивает разработчикам мощные инструменты для создания модульных, динамичных и высокопроизводительных компонентов. Применение React способствует повышению гибкости и масштабируемости frontend-части проекта.

Для ускорения создания визуального интерфейса и обеспечения единообразного дизайна применяется фреймворк Semantic-UI [9]. Он предоставляет готовые компоненты и стили, что позволяет быстро создавать современный и отзывчивый пользовательский интерфейс, улучшая общий пользовательский опыт.

Для повышения надежности и удобства разработки используется язык программирования TypeScript [10]. Он добавляет статическую типизацию к JavaScript, что позволяет выявлять и исправлять ошибки на этапе разработки, а также облегчает сопровождение и масштабирование проекта в долгосрочной перспективе.

### **2.1.2. Backend-разработка**

Ключевым инструментом backend-разработки выбран язык программирования Go [11], известный своей эффективностью и производительностью. Backend представляет

собой Go-модуль и расположен в папке `backend` в корневой директории. Для удобства дальнейшей разработки и поддержки кода в проекте использованы общепринятые практики и подходы к написанию кода.

Проект структурирован в соответствии со стандартным Go макетом [12]. Внешние инструменты расположены в папке `pkg` – их при необходимости можно будет импортировать в другие проекты. В папке `internal` расположены внутренние инструменты проекта (чтение файлов с конфигурацией, взаимодействие с базой данных, инициализация основных компонентов сервиса, роутинг запросов) и бизнес-логика. Запуск приложения осуществляется из файла `main`, директория `cmd/higher_search`.

Паттерн проектирования `Application` является ключевым элементом архитектуры данного проекта. Он позволяет структурировать приложение таким образом, чтобы каждая его часть была легко заменяема и расширяема. Это достигается за счет следующих особенностей:

1. **Модульность.** Приложение разделено на модули, каждый из которых отвечает за определенную функциональность. Такое разделение позволяет легко добавлять, удалять или обновлять отдельные компоненты, не затрагивая остальную часть системы.
2. **Слабая связанность.** Изменения в одном модуле не оказывают существенного влияния на другие. Это достигается за счет использования четко определенных интерфейсов между модулями.
3. **Расширяемость.** Благодаря модульной архитектуре и слабой связанности, приложение легко расширяется новыми функциями. Для этого достаточно добавить новый модуль, реализующий требуемую функциональность, и интегрировать его с существующей системой.
4. **Тестируемость.** Разделение приложения на независимые модули упрощает процесс тестирования. Каждый модуль может быть протестирован отдельно при помощи технологий mock-тестирования, что позволяет быстро выявлять и устранять ошибки.

Таким образом, применение паттерна `Application` позволяет придерживаться принципов чистой архитектуры: код разделен на независимые слои (презентация, бизнес-логика, инфраструктура), что повышает гибкость, тестируемость и масштабируемость системы.

Используются многие другие идиоматические практики языка Go: завертывание и последующая обработка ошибок, передача контекста для упрощенного взаимодействия

между компонентами, использование интерфейсов для разделения слоев абстракции и так далее. Для написания более эффективного, надежного и читаемого кода в процессе разработке используются различные утилиты: технология go vet помогает обнаруживать потенциальные ошибки и не оптимальные конструкции в коде; go fmt форматирует код в соответствии с общепринятым стилем; go lint анализирует код на предмет нарушений указанных стандартов и рекомендует изменения для улучшения качества кода. Эти и некоторые другие инструменты объединены в общий скрипт в директории backend, что упрощает их запуск и ускоряет процесс разработки.

В качестве системы управления базами данных в проекте используется PostgreSQL [13]. Она предлагает SQL-совместимость, ACID-свойства через транзакции и расширяемость для обработки различных типов данных. Создание контейнера с PostgreSQL в Docker позволяет легко развернуть базу данных в любой среде без установки на хост-систему. Интеграция PostgreSQL через контейнеризацию упрощает развертывание, масштабирование и управление базой данных.

Для запросов к базе данных используется библиотека sqlx [14], которая предоставляет набор расширений к стандартной go библиотеке database/sqlx. Ключевым преимуществом этой библиотеки является автоматическая сериализация полей базы данных в заданную структуру.

Для обеспечения безопасности и аутентификации пользователей используется технология JWT [15] (JSON Web Tokens), обеспечивающий безопасную передачу данных между клиентом и сервером. Новый JWT токен формируется на стороне сервера когда пользователь авторизован и хранится на стороне клиента в файлах Cookie.

Конфигурационные данные хранятся в .env-файле, поскольку его удобно передавать в docker-контейнеры. Для обработки конфигурационных файлов также используется специальная go-библиотека – Clean Env [16].

Эффективное сочетание этих технологий, подходов и паттернов в backend-разработке проекта обеспечивает создание надежной, масштабируемой и безопасной серверной части приложения, готовой к эффективной работе и развитию.

### **2.1.3. Взаимодействие клиентской и серверной части**

Для взаимодействия клиентской и серверной части было решено использовать систему gRPC [17] и технологию Protocol Buffers [18]. Ключевым аргументом использования gRPC в данном проекте является удобство использования protobuf-моделей.

Backend и frontend постоянно обмениваются моделями (содержащими данными студенческих организаций, пользователей и так далее). Технология protobuf позволяет использовать кодогенерацию, и автоматически создавать модели на языках Go и TypeScript. Это существенно упрощает разработку и делает процесс передачи данных между frontend-ом и backend-ом более прозрачным.

Protobuf-модели хранятся в папке proto в корне проекта. Для того, чтобы конвертировать protobuf-модели в модели на Go и TypeScript, используются библиотеки protoc-gen-go-grpc [19] и protoc-gen-ts [18] соответственно.

Для того, чтобы gRPC сервер умел обрабатывать HTTP запросы, нужно использовать дополнительную библиотеку grpc-web [20]. С помощью этой технологии поднимается HTTP-сервер, совместимый с web-клиентами.

#### **2.1.4. Devops-методология**

Для развертывания сервиса используются технологии Docker [22] и docker-compose [23]. Для backend-а и frontend-а написаны Dockerfile-ы (хранятся в соответствующих директориях). Docker Compose дает возможность управлять многоконтейнерными приложениями, определяя структуру приложения в одном конфигурационном файле. В данном проекте с помощью docker-compose можно развернуть одновременно сразу несколько контейнеров (backend, frontend и базу данных), запустив всю систему одной командой.

Описание конфигурации хранится в файле .env и передается внутрь docker-compose.

Для того, чтобы было возможно раздавать статику с клиентской части и обрабатывать запросы с серверной части на одном домене одновременно, необходимо использовать обратный прокси.

Исходные коды проекта поддерживаются системой контроля версий git и имеют удаленный репозиторий на сервисе github.

## **2.2. Составные части системы**

Исходя из нефункциональных требований, рассмотренных в предыдущей главе, можно сформировать более детальное представление о функционале системы. Выделим крупные блоки функциональности:



**1. Личный кабинет.** Предоставляет возможности для изменения личной информации, просмотра списка своих организаций, выход из организаций. Должны быть реализованы механизмы для авторизации и регистрации.

**2. Система навигации.** Включает в себя поиск организаций по названию и расширенный поиск по тегам. Результатом поиска является список организаций. Позволяет переходить на страницы конкретных организаций.

**3. Страницы-визитки студенческих организаций.** Отображают всю необходимую информацию о студенческих организациях.

**4. Администрирование студенческой организации.** Включает в себя создание новой организации, изменение информации о существующей организации (только для пользователей, являющихся организаторами).

**5. Система модерации.** Позволяет пользователям с соответствующими правами скрывать студенческие организации и оставлять комментарии к ним.

### **2.3. Выводы по главе**

В этой главе была рассмотрена высокоуровневая структура проекта, методология разработки и набор используемых технологий. Были перечислены основные процессы разработки и ключевые блоки функциональности. Таким образом, мы получили общее представление о проекте, его архитектуре и используемом стеке технологий.

Следующим шагом будет более детальное рассмотрение каждого из компонентов системы, их взаимодействия друг с другом, а также описание реализации основных сценариев использования.

### Глава 3. Особенности программной реализации и результаты работы

В результате работы была разработано веб-приложение, отвечающее всем поставленным требованиям. Исходный код проекта размещен в открытом доступе на платформе GitHub: [https://github.com/DubrovEva/higher\\_search](https://github.com/DubrovEva/higher_search).

Итоговый проект может быть представлен с двух различных аспектов: как с точки зрения процессов разработки (backend, frontend, devops, взаимодействие с базой данных, protobuf-модели), и как с точки зрения блоков функциональности (личный кабинет пользователя, администрирование сообщества, и т.д.).

Относительно блоков функциональности проект может быть представлен как набор взаимосвязанных функций, каждая из которых выполняет определенные задачи. Личный кабинет обеспечивает доступ к информации об авторизованном пользователе, администрирование сообщества обеспечивает управление данными конкретной организации, и так далее. Рассматривая каждый отдельный блок, можно говорить об общем наборе реализованных методов. Их мы рассмотрим в первой половине главы.

С точки зрения процессов разработки, проект может быть рассмотрен как сложная система, состоящая из взаимосвязанных компонентов, каждый из которых выполняет определенные функции. Backend обеспечивает обработку данных, frontend обеспечивает пользовательский интерфейс, devops обеспечивает автоматизацию процессов, а взаимодействие с базой данных и protobuf-модели обеспечивают эффективное хранение и передачу данных. Во второй части главы мы последовательно рассмотрим реализацию каждого отдельного слоя абстракции.

В заключительной части данной главы мы уделим внимание анализу перспектив дальнейшего развития проекта, а также рассмотрим возможные пути его совершенствования как с позиции расширения функциональных возможностей системы, так и с точки зрения повышения удобства и комфорта для пользователей.

#### 3.1. Бизнес-логика проекта

В пункте 2.2 были рассмотрены составные части системы, определяющие базовую функциональность. В следующих подпунктах мы подробнее рассмотрим, как страницы были разработаны, и какие функции реализованы.

### 3.1.1. Личный кабинет. Функционал регистрации и авторизации.

На стартовой странице приложения (рис. 1) расположены кнопка “Войти”, направляющая пользователя на страницу авторизации (рис. 2), и кнопка “Зарегистрироваться”, направляющая на страницу регистрации (рис. 3). После прохождения соответственно авторизации или регистрации пользователь оказывается в личном кабинете. На клиентской стороне пользователя формируется JWT-токен. Он будет храниться в файлах Cookie. При генерации страниц на серверной части осуществляется валидация токена. Если пользователь будет авторизован, на страницах будет отображен дополнительный/измененный функционал – к примеру, у авторизованного пользователя на стартовой странице не отображаются кнопки “Войти” и “Зарегистрироваться”.

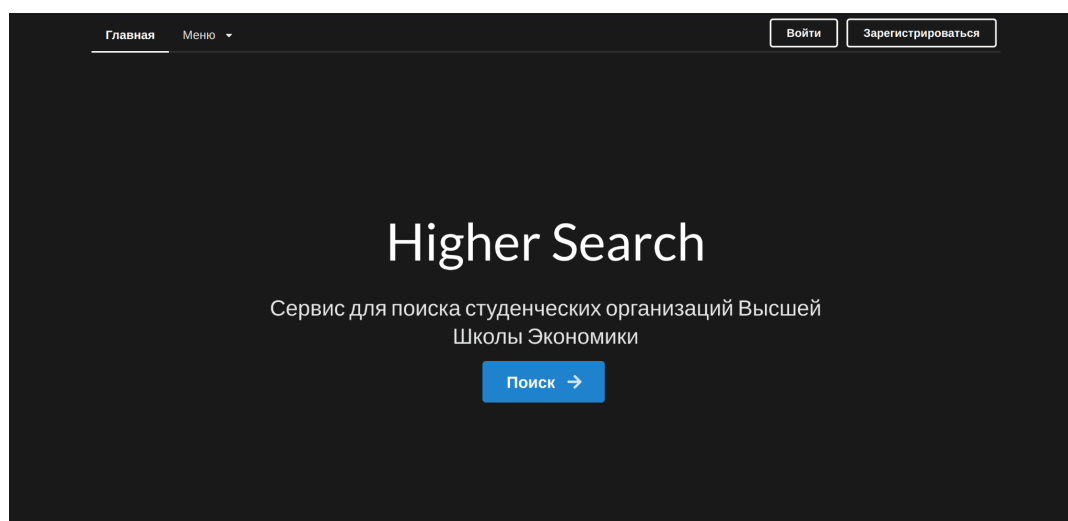
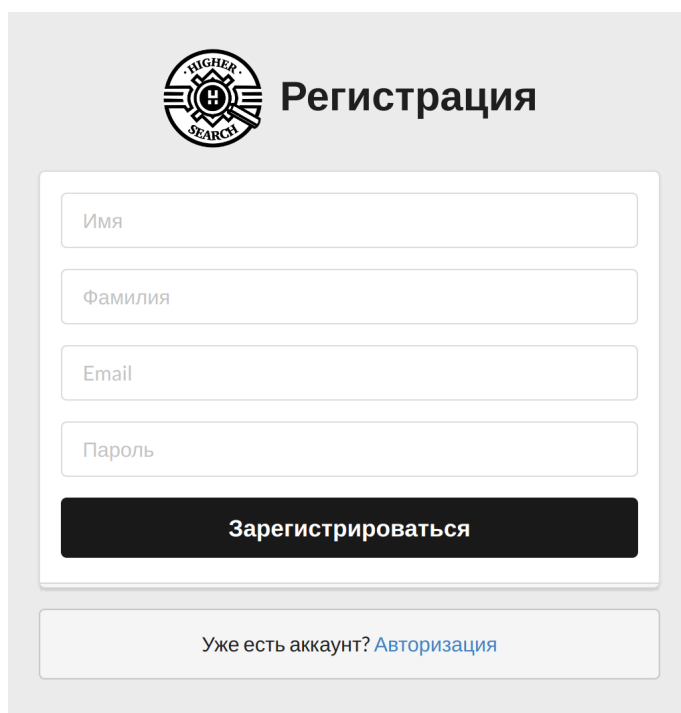


Рис. 1: Стартовая страница

The image shows the authorization form. At the top left is a circular logo with 'HIGHER SEARCH' and a stylized 'H'. To the right of the logo is the title 'Авторизация' (Authorization) in a bold, black font. Below the title is a white form box with rounded corners. Inside the form, there are two input fields: the first is labeled 'E-mail address' with a person icon, and the second is labeled 'Password' with a lock icon. Below these fields is a black button with the text 'Login' in white. At the bottom of the form box, there is a link that says 'Нет аккаунта? Зарегистрироваться' (No account? Register).

Рис. 2: Форма авторизации




The image shows a registration form titled "Регистрация" (Registration) with a logo for "HIGHER SEARCH" featuring a gear and a magnifying glass. The form contains four input fields: "Имя" (Name), "Фамилия" (Surname), "Email", and "Пароль" (Password). Below these fields is a black button with the text "Зарегистрироваться" (Register). At the bottom, there is a link that says "Уже есть аккаунт? [Авторизация](#)" (Already have an account? [Authorization](#)).

Рис. 3: Форма регистрации

Личный кабинет предоставляет следующие возможности:

1. Обновить личные данные (поменять имя, фамилию, отчество, указать пол, факультет, дополнительную информацию об обучении, создать короткое описание);
2. Посмотреть, как личные данные пользователя будут отображаться на стандартной карточке (рис. 4). Такие же карточки будут отображаться, когда этот пользователь будет указан как контактное лицо студенческой организации;
3. Просмотреть свой список студенческих организаций (рис. 5). На карточках студенческих организаций отображается статус конкретного пользователя в организации, дата вступления в организацию.
4. Перейти на страницу редактирования организации (для организаторов и главы сообщества)
5. Покинуть организацию (для участников сообщества)
6. Перейти на страницу-визитку организации

## Личная информация



**Ева Дубровская**  
Факультет компьютерных наук, 4 курс, БПИ201  
Разработчик проекта Higher Search  
3 студенческих организации

**Фамилия**  
Дубровская

**Имя**  
Ева

**Отчество**  
Отчество

**Пол**  
Женский

**Факультет**  
Факультет компьютерных наук


**Информация об обучении**  
4 курс, БПИ201

**Описание**  
Разработчик проекта Higher Search

Сохранить


Рис. 4: Личный кабинет: личная информация

## Организации




**Студактив ФКН**  
Участник с 5/15/2024  
Глава

Редактировать



**Шахматный клуб**  
Участник с 5/15/2024  
Организатор

Редактировать



**Just Dance**  
Участник с 5/15/2024  
Участник

Покинуть

Рис. 5: Личный кабинет: студенческие организации

### 3.1.2. Система навигации

Система навигации включает в себя полный список организаций (рис. 6), поиск по названию и расширенный поиск (рис. 7). Результаты поиска представлены как список организаций в формате карточек-превью. С каждой карточки можно перейти на страницу-визитку, на которой будет представлена полная информация об организации.

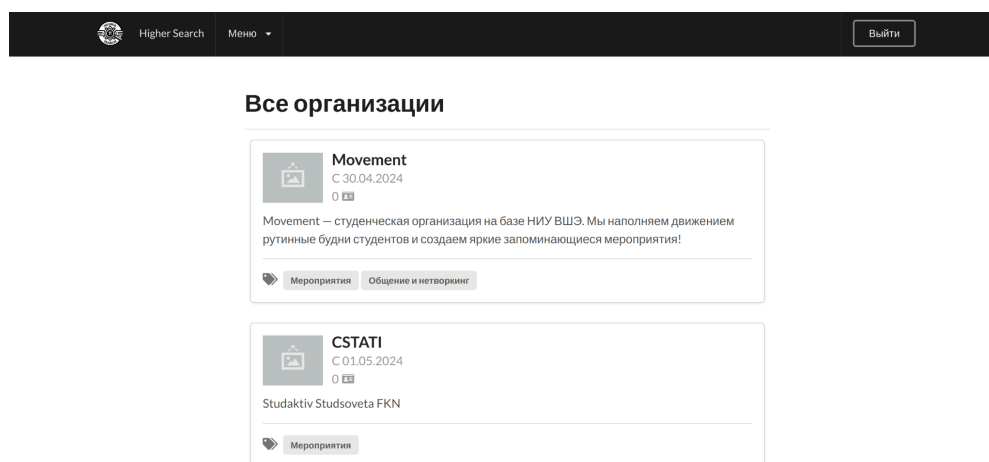


Рис. 6: Полный список организаций

Рис. 7: Расширенный поиск

### 3.1.3. Система модерации

Важным условием запуска продукта является наличие у сотрудников ЦПСИ следить за деятельностью сообществ и при необходимости выступать в роли модераторов, то есть скрывать страницы сообществ, которые нарушают правила университета. Эта

задача решается с помощью различных уровней доступа – сотрудникам ЦПСИ ставится особая роль в системе, с которой они могут не просто просматривать страницы, но и убирать некоторые организации в архив с дополнительными комментариями. У студентов-организаторов будет возможность просмотреть эти комментарии, изменить поля организаций и подать заявку на выход из архива.

### 3.1.4. Страницы-визитки студенческих организаций: просмотр и администрирование

На страницах-визитках организаций отображена вся необходимая информация: название организации, описание, кампус, факультет, основной язык, контакты организаторов, ссылки на любые дополнительные ресурсы и профиль главы организации.

Администрирование студенческих организаций доступно только пользователям, которые являются главами или организаторами представленных на сайте сообществ. На специальной странице они могут выбрать одно из своих сообществ и изменить его поля (рис. 8). Форма создания организации аналогична форме редактирования существующего сообщества.

The screenshot shows a web form titled "Создание студенческой организации" (Creating a student organization). The form is set against a dark header with a logo, "Higher Search", a "Меню" (Menu) dropdown, and a "Выйти" (Logout) button. The form fields include: a "Название" (Name) field with a red asterisk and placeholder "Новая организация"; "Кампус" (Campus) and "Факультет" (Faculty) dropdown menus with "Санкт-Петербург" and "Факультет гуманитарных наук" selected respectively; a "Язык" (Language) dropdown menu with "Русский" selected; a "Краткое описание" (Short description) text area with a placeholder "Опиши организацию в 2-3 предложениях. Это короткое превью, которое будет отображаться в поиске."; an "Описание" (Description) text area with a placeholder "Полное описание организации, будет отображено на отдельной странице."; and a "Категории" (Categories) section with buttons for "Кино", "Юмор", and "Общение и нетворкинг", all of which are currently selected.

Рис. 8: Создание новой организации

## 3.2. Технические аспекты

### 3.2.1. Protobuf-модели

В корневой директории расположена папка proto. В ней находятся две папки: в папке api расположена модель роутера, а в папке models представлены основные модели, которыми обмениваются backend и frontend.

В роутере описаны непосредственно gRPC-методы (рис. 9). В папке models находятся как сами модели, так и enum-ы – структуры данных, задающие возможные значения для некоторых полей (рис. 10).

```
service Router {  
    rpc GetUser (user.UserID) returns (UserResponse);  
    rpc GetUsers (user.UserIDs) returns (UsersResponse);  
    rpc InsertUser (user.UserInfo) returns (UserResponse);  
    rpc UpdateUser (user.User) returns (UserResponse);  
}
```

Рис. 9: Пример gRPC-методов

```
7 message StudorgID {  
8     int64 ID = 1;  
9 }  
10  
11 message StudorgInfo {  
12     common.Campus campus = 1;  
13     string createdAt = 2;  
14     string description = 3;  
15     common.Faculty faculty = 4;  
16     common.Language language = 5;  
17     repeated common.Links links = 6;  
18     string logo = 7;  
19     string name = 8;  
20     string shortDescription = 9;  
21     StudorgStatus status = 10;  
22     repeated string tags = 11;  
23     repeated Contact contacts = 14;  
24 }  
25
```

Рис. 10: Пример proto-моделей

### 3.2.2. Использование Protobuf-моделей со стороны серверной части приложения

С помощью Makefile в корневой директории из proto-файлов генерируются модели на языке Go. Они сохраняются в директорию backend/pkg/proto/...



При запуске приложения в папке application.go вызывается в том числе метод `initServer()`, который инициализирует gRPC-сервер. Дополнительно поднимается HTTP-сервер для того чтобы принимать HTTP запросы.

Все модели, пришедшие по gRPC, конвертируются с помощью специальных функций во внутренние модели базы данных.

```
51 func (r *Router) GetUser(_ context.Context, userID *proto.UserID) (*service.UserResponse, error) {
52     userDB, err := r.User.Get(userID.GetID())
53     if err != nil { return nil, err }
54
55     result, err := userDB.ToProtoUser()
56     if err != nil { return nil, err }
57
58     return &service.UserResponse{Response: &service.UserResponse_User{User: result}}, nil
59 }
```

Рис. 11: Пример реализации gRPC-метода

### 3.2.3 Использование Protobuf-моделей со стороны frontend-a

Генерация моделей на языке TypeScript осуществляется через скрипт `proto-gen-grpc-web.sh` в корневой директории.

При запуске frontend-a также инициализируется клиент, который посылает запросы по GRPC, дополнительно обрабатывая входящие запросы.

### 3.2.4. Инициализация базы данных

Инициализация базы данных осуществляется один раз, когда создается `docker-volume`. Скрипт, инициализирующий базу данных, расположен в директории `deployments` – в нем 5 sql-команд, создающих таблицы:

1. **users** – пользователи;
2. **studorgs** – студенческие организации;
3. **tags** – иерархия тегов студенческих организаций;
4. **studorg2tag** – many-to-many таблица отношений организаций к тегам;
5. **user2studorg** – many-to-many таблица отношений пользователей к организациям.

ER-диаграмма таблиц приведена на рис. 12.

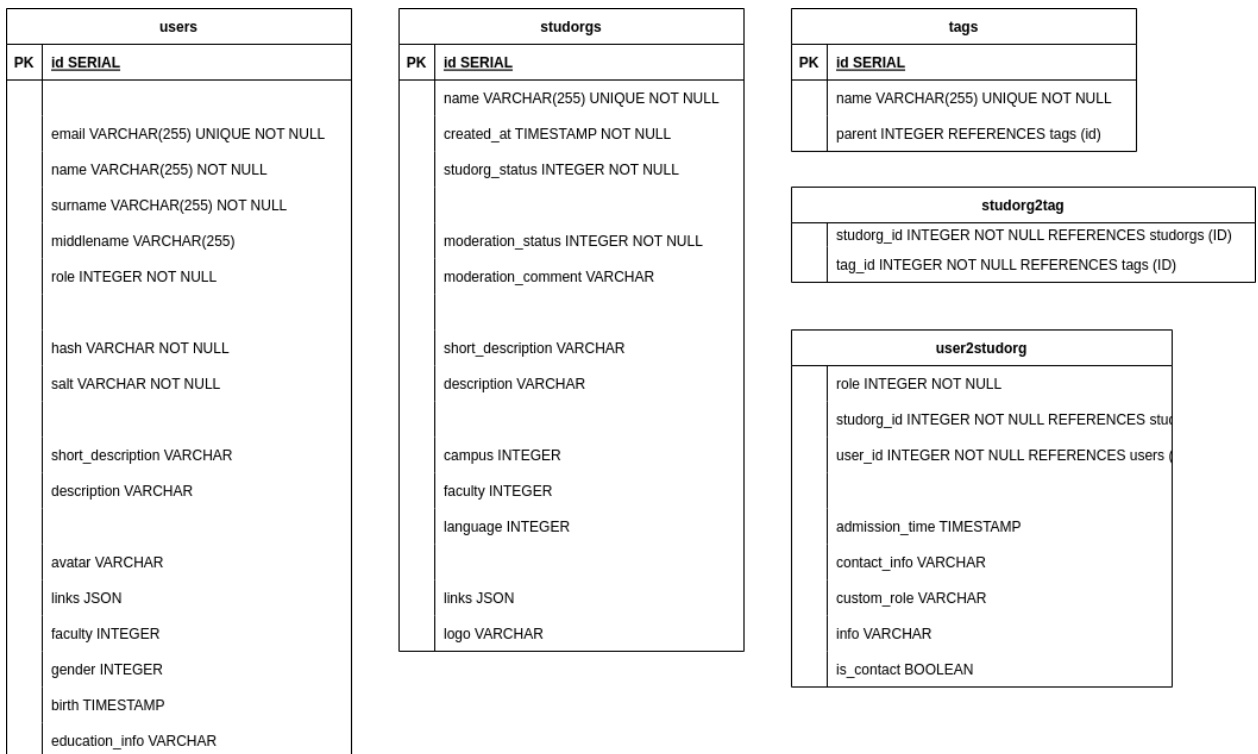


Рис. 12: ER-Диаграмма базы данных

### 3.2.5. Механизм авторизации пользователей

Авторизация пользователей осуществляется с помощью технологии JWT.

Когда пользователь входит в систему, его учетные данные (email и пароль) проверяются на сервере. Если данные верны, сервер генерирует JWT-токен.

JWT-токен состоит из трех частей, разделенных точкой: заголовка (header), полезной нагрузки (payload) и подписи (signature). Заголовок содержит информацию о типе токена и алгоритме шифрования. Полезная нагрузка содержит данные о пользователе – в данном случае, уникальный идентификатор пользователя в базе данных. Подпись создается с помощью секретного ключа, известного только серверу, и используется для проверки целостности токена.

После генерации JWT-токен отправляется клиенту и сохраняется в cookie-файле. Когда сервер получает запрос с JWT-токеном, он проверяет его подпись, используя свой секретный ключ. Если подпись верна, сервер извлекает данные из полезной нагрузки токена и авторизует пользователя.

Обновление токена: Для повышения безопасности JWT-токены имеют ограниченный срок действия. Когда токен истекает, клиент должен получить новый.

Таким образом, JWT обеспечивает безопасную передачу данных между клиентом и сервером без необходимости хранения сессионной информации на сервере.

### **3.2.6. Хранение паролей**

Для авторизации на сайте пользователям необходимо ввести пароль. Однако хранить пароли напрямую в базе данных небезопасно: если база данных будет скомпрометирована, злоумышленники получат доступ ко всем паролям пользователей, что может привести к массовому взлому аккаунтов [25].

Вместо этого, правильным подходом является хранение не самих паролей, а их хэшей. Хэширование - это процесс преобразования пароля в фиксированную последовательность символов (хэш) с помощью криптографической хэш-функции. Это позволяет проверять правильность пароля при авторизации, не раскрывая сам пароль.

Однако, простое хранение хэшей также не является достаточно безопасным. Злоумышленники могут использовать методы перебора (brute-force) для восстановления паролей по их хэшам. Поэтому необходимо использовать соленые хэши.

Соль - это случайная строка, которая добавляется к паролю перед хэшированием. Это усложняет атаки перебора и радужных таблиц, так как каждый пароль будет иметь уникальный хэш. Соли хранятся вместе с хэшами в базе данных.

Таким образом, в базе данных хранятся не сами пароли пользователей, а соль, сгенерированная для каждого отдельного пользователя случайным образом, и хэш от пароля с солью. Для генерации хэшей используется `go` библиотека [golang.org/x/crypto/pbkdf2](https://golang.org/x/crypto/pbkdf2).

### **3.2.7. Клиентская часть**

Компоненты клиентской части по большей части представляют собой готовые компоненты из библиотеки `semantic-ui`. Некоторые повторяющиеся компоненты (такие как меню, карточки, отображающие, студенческие организации и пользователей, и так далее) вынесены отдельно в папку `components` как экспортированные функции (рис. 13).

```

export function OrganizationsMenu() : JSX.Element { 1+ usages  ▴ Dubrovskaya Eva *
  return (
    <Dropdown text={"Меню"} simple>
      <DropdownMenu>
        <DropdownItem href="/list" active> Полный список </DropdownItem>
        <DropdownItem href="/search"> Расширенный поиск </DropdownItem>
        <DropdownItem href="/studorg/create"> Новая организация </DropdownItem>
      </DropdownMenu>
    </Dropdown>
  );
}

```

Рис. 13: Пример повторяющегося компонента

### 3.2.8. Логирование

Для упрощенного тестирования приложения дописан дополнительный обработчик, который логирует вызовы gRPC методов (рис. 14).

```

func NewInterceptor() grpc.UnaryServerInterceptor { 1 usage  ▴ Dubrovskaya Eva
  return func(ctx context.Context, req interface{}, info *grpc.UnaryServerInfo,
    handler grpc.UnaryHandler) (resp interface{}, err error) {
    log.Println(v...: "Processing request")

    defer func() {
      if r := recover(); r != nil {
        err = status.Errorf(codes.Internal, format: "panic: %s", r)
        log.Printf(format: "Panic detected: %s", r)
      }
    }()

    resp, err = handler(ctx, req)
    errCode := status.Code(err)
    if errCode == codes.Unknown || errCode == codes.Internal {
      log.Printf(format: "Request handler returned an internal error: %s", err)
      return
    }

    log.Println(v...: "Request finished successfully")
    return
  }
}

```

Рис. 14: Обработчик, логирующий вызовы gRPC методов

## 3.3. Выводы по главе

В этой главе рассматривались технические аспекты проекта, а также представлены окончательные результаты разработки приложения. В описании этих результатов были подробно рассмотрены особенности реализации каждого компонента приложения: клиентской части и серверной части. Кроме того, был полностью продемонстрирован функционал проекта, включая работу интерфейса и получение результатов.

## **Заключение**

В ходе выполнения выпускной квалификационной работы была рассмотрена проблема разрозненности информации о студенческих организациях ВШЭ и отсутствия единого ресурса для их поиска и рекламы. Для решения этой проблемы была предложена и в дальнейшем успешно реализована концепция создания единой платформы для навигации в студенческих сообществах.

Основные результаты и достижения:

1. Была разработана платформа, предоставляющая удобные инструменты для поиска, создания и модерации студенческих организаций.
2. Основным отличием проекта стало предоставление организаторам возможности самостоятельного обновления информации о своих сообществах, обеспечивая актуальность и полноту данных.
3. В ходе работы были выполнены все поставленные задачи, начиная от постановки задачи и выявления требований, и заканчивая разработкой технической документации приложения.

Дальнейшее развитие проекта предполагает расширение функционала, улучшение пользовательского опыта, интеграцию с другими платформами и сервисами, а также продвижение среди студенческого сообщества. Новый функционал, такой как сбор статистики, рассылки, регистрация на мероприятия и система оценок организаций, позволит улучшить взаимодействие студентов с платформой и повысит ее популярность и эффективность. Разработанный сервис открывает новые возможности для улучшения информационной среды студенческих сообществ и способствует развитию студенческой активности и самоорганизации.

## Список использованных источников

- 1) EXTRA [Электронный ресурс] //URL: <https://extra.hse.ru/> (Дата обращения: 20.04.2024, режим доступа: свободный).
- 2) Студенческие организации ВШЭ [Электронный ресурс] //URL: [https://vk.com/@hse\\_fes\\_2025-studencheskije-organizacii-vshe](https://vk.com/@hse_fes_2025-studencheskije-organizacii-vshe) (Дата обращения: 20.04.2024, режим доступа: свободный).
- 3) Почемучник для первокурсников 2018 [Электронный ресурс] //URL: <https://www.hse.ru/pochemuchnik2018/studlife/> (Дата обращения: 20.04.2024, режим доступа: свободный).
- 4) Внеучебная жизнь в питерской Вышке [Электронный ресурс] //URL: <https://spb.hse.ru/pochemuchnik/studlife/> (Дата обращения: 20.04.2024, режим доступа: свободный).
- 5) Node.js [Электронный ресурс] //URL: <https://nodejs.org/en> (Дата обращения: 24.04.2024, режим доступа: свободный).
- 6) Vite [Электронный ресурс] //URL: <https://vitejs.dev/> (Дата обращения: 22.04.2024, режим доступа: свободный).
- 7) Remix [Электронный ресурс] //URL: <https://remix.run/> (Дата обращения: 22.04.2024, режим доступа: свободный).
- 8) React [Электронный ресурс] //URL: <https://react.dev/blog/2023/03/16/introducing-react-dev> (Дата обращения: 22.04.2024, режим доступа: свободный).
- 9) Semantic-UI [Электронный ресурс] //URL: <https://react.semantic-ui.com/> (Дата обращения: 22.04.2024, режим доступа: свободный).
- 10) TypeScript [Электронный ресурс] //URL: <https://www.typescriptlang.org/> (Дата обращения: 22.04.2024, режим доступа: свободный).
- 11) Go [Электронный ресурс] //URL: <https://go.dev/> (Дата обращения: 23.04.2024, режим доступа: свободный).
- 12) Go Layout [Электронный ресурс] //URL: <https://github.com/golang-standards/project-layout> (Дата обращения: 23.04.2024, режим доступа: свободный).
- 13) PostgreSQL [Электронный ресурс] //URL: <https://www.postgresql.org/> (Дата обращения: 23.04.2024, режим доступа: свободный).

- 14) sqlx [Электронный ресурс] //URL: <https://github.com/jmoiron/sqlx> (Дата обращения: 23.04.2024, режим доступа: свободный).
- 15) JWT [Электронный ресурс] //URL: <https://jwt.io/> (Дата обращения: 23.04.2024, режим доступа: свободный).
- 16) Clean Env [Электронный ресурс] //URL: <https://github.com/ilyakaznacheev/cleanenv> (Дата обращения: 25.04.2024, режим доступа: свободный).
- 17) gRPC [Электронный ресурс] //URL: <https://grpc.io/> (Дата обращения: 25.04.2024, режим доступа: свободный).
- 18) Protocol Buffers [Электронный ресурс] //URL: <https://protobuf.dev/> (Дата обращения: 25.04.2024, режим доступа: свободный).
- 19) protoc-gen-go-grpc [Электронный ресурс] //URL: <https://pkg.go.dev/google.golang.org/grpc/cmd/protoc-gen-go-grpc> (Дата обращения: 24.04.2024, режим доступа: свободный).
- 20) protoc-gen-ts [Электронный ресурс] //URL: <https://www.npmjs.com/package/protoc-gen-ts> (Дата обращения: 25.04.2024, режим доступа: свободный).
- 21) grpc-web [Электронный ресурс] //URL: <https://www.npmjs.com/package/protoc-gen-ts> (Дата обращения: 27.04.2024, режим доступа: свободный).
- 22) Docker [Электронный ресурс] //URL: <https://www.docker.com/> (Дата обращения: 29.04.2024, режим доступа: свободный).
- 23) docker-compose [Электронный ресурс] //URL: <https://docs.docker.com/compose/> (Дата обращения: 29.04.2024, режим доступа: свободный).
- 24) Sriramya P., Karthika R. A. Providing password security by salted password hashing using bcrypt algorithm //ARPN journal of engineering and applied sciences. – 2015. – Т. 10. – №. 13. – С. 5551-5556.