СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных условиях цифровизации бизнеса и широкого распространения удаленного режима работы критически важным фактором успеха компании является эффективность внутренних коммуникаций. Разрозненность информационных потоков, использование множества неинтегрированных инструментов (мессенджеры, файлообменники, электронная почта) приводят к потере важной информации, снижению управляемости и замедлению адаптации новых сотрудников.

Решением данной проблемы является внедрение единого корпоративного портала – веб-ориентированной информационной системы, которая консолидирует управление организационной структурой, внутренними новостями и базой знаний (документами). Такая система позволяет создать единое информационное пространство, обеспечить прозрачность иерархии и разграничить доступ к информации в соответствии с ролью сотрудника.

Целью курсового проекта является проектирование и разработка программного средства «Корпоративный портал» для автоматизации управления сотрудниками, командами и корпоративным контентом.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ АНАЛОГОВ

1.1 Понятие и роль внутреннего порта компании

Внутренний портал компании представляет собой единую, защищённую точку доступа к информационным ресурсам и сервисам компании, ориентированную на сотрудников. Его ключевая задача – объединение разрозненных данных и бизнес-процессов в целостную информационную среду, что способствует повышению эффективности управления и операционной деятельности.

Роль внутреннего портала в современном бизнесе определяется решением ряда задач, одни из которых:

- повышение эффективности сотрудников за счёт централизованного и быстрого доступа к необходимым для работы инструментам и данным;
- улучшение коммуникаций между отделами, руководством и коллективом в целом;
- автоматизация рутинных процессов, таких как согласование документов;
- формирование и поддержание корпоративной культуры через публикацию новостей, ведение блогов руководителей и обсуждения во внутренних сообществах.

1.2 Анализ существующих аналогов

Для формирования требований к разрабатываемой системе был проведён анализ существующих готовых решений: «Microsoft SharePoint», «Confluence» и «Bitrix24».

1.2.1 «Microsoft SharePoint» – платформа от «Microsoft», предназначенная для организации внутреннего пространства компании. Она позволяет создавать интерактивные сайты-страницы, управлять документами, настраивать рабочие процессы и делиться информацией между сотрудниками. «Microsoft SharePoint» особенно ценится за глубокую интеграцию с экосистемой «Microsoft 365»: документы «Word», «Excel» и «PowerPoint» можно редактировать прямо в браузере, а доступ к материалам легко регулируется через Active Directory. Благодаря гибкости и масштабируемости, «Microsoft SharePoint» широко используется в крупных организациях, где

важны безопасность, контроль доступа и возможность кастомизации [1]. На рисунке 1.1 изображён интерфейс платформы.

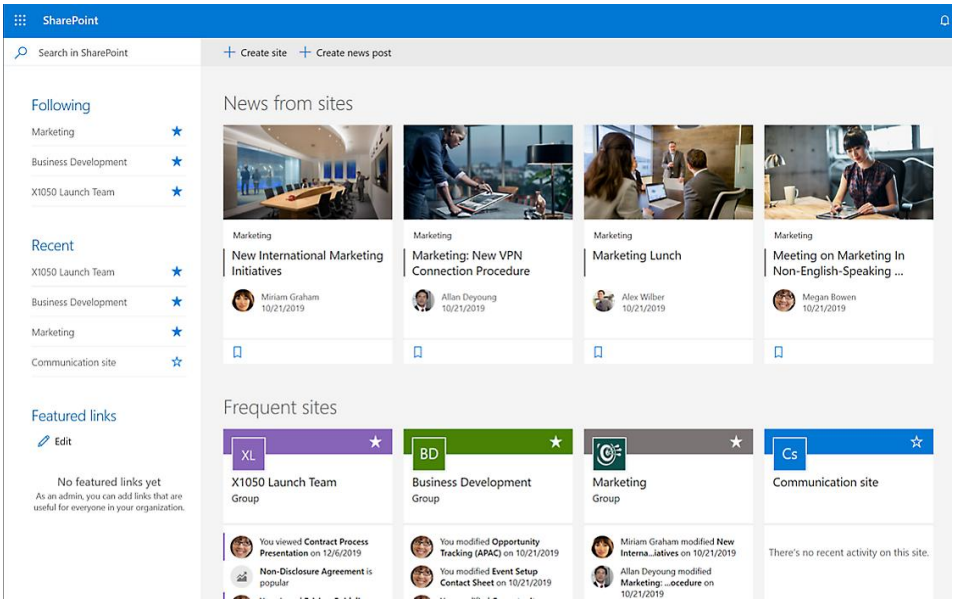

Рисунок 1.1 – Интерфейс платформы «Microsoft SharePoint»

1.2.2 «Confluence» от «Atlassian» – это платформа ориентированная на совместную работу и ведение документации. Она построена по принципу Wiki: сотрудники создают страницы, делятся знаниями, комментируют и редактируют материалы в реальном времени. «Confluence» особенно популярна среди команд разработчиков и проектных менеджеров, благодаря тесной интеграции с «Jira» и другими инструментами «Atlassian». Интерфейс интуитивно понятен, а структура страниц легко адаптируется под нужды конкретной команды. Это делает «Confluence» идеальным решением для компаний, где важно централизованное хранение знаний и прозрачность коммуникации[2]. На рисунке 1.2 изображён интерфейс платформы «Confluence».

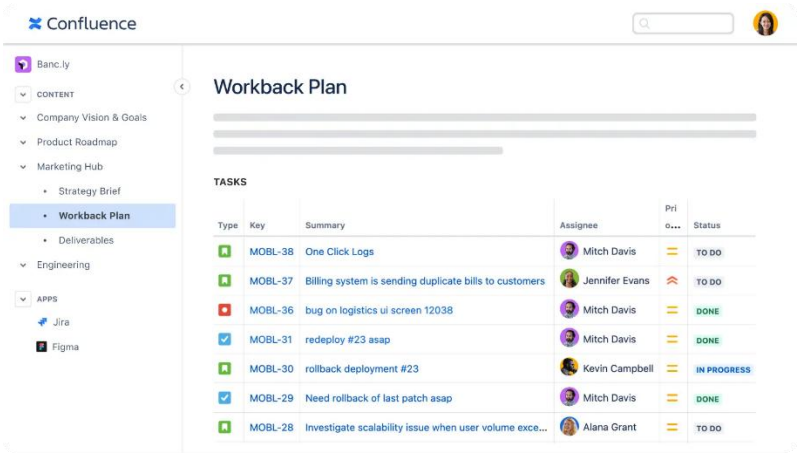

Рисунок 1.2 – Интерфейс платформы «Confluence»

1.2.3 «Bitrix24» – это универсальная платформа, которая объединяет в себе функции CRM, управления задачами, коммуникаций и внутреннего портала. Она предлагает богатый набор инструментов: от чатов и видеозвонков до документооборота и автоматизации бизнес-процессов. «Bitrix24» доступна как в облачной версии, так и в коробочной, что особенно важно для компаний с повышенными требованиями к безопасности. Платформа активно используется в странах СНГ и подходит как для малого бизнеса, так и для крупных организаций, благодаря гибкой настройке и широкому функционалу [3]. На рисунке 1.3 изображён интерфейс платформы.

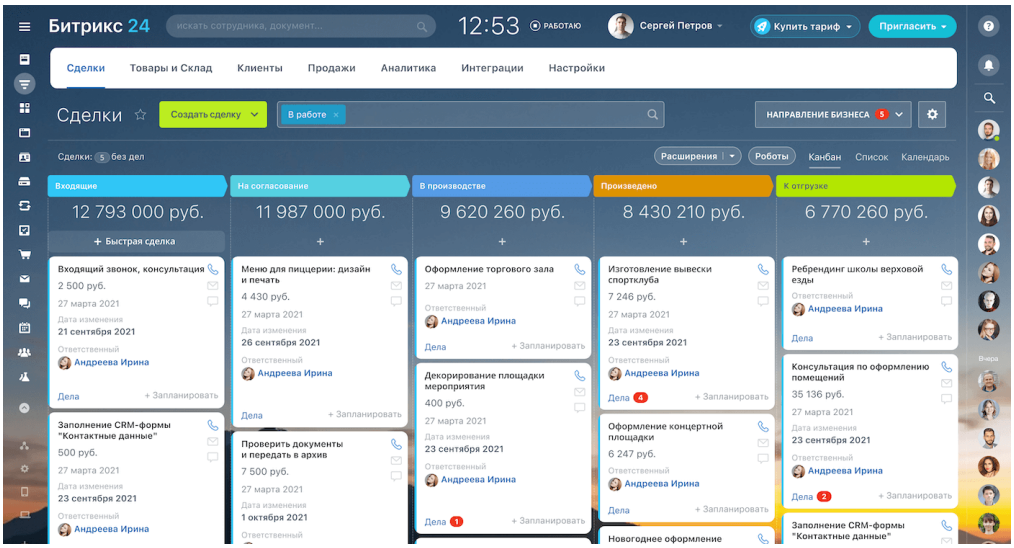

Рисунок 1.3 – Интерфейс платформы «Bitrix24»

Таблица 1.1 – Сводная таблица по характеристикам существующих аналогов

Характеристика	Microsoft SharePoint	Confluence	Bitrix24
Тип платформы	Корпоративный портал	Вики/документация	Универсальный бизнес-инструмент
Интеграции	Microsoft 365, Teams	Jira, Trello	1С, почта
Поддержка видеозвонков	Через Teams	–	Встроенные звонки и чаты
Управление задачами	+	Ограничено	+
Хранилище документов	+	+	+
Контроль версий	+	+	+
Локализация	Многоязычная	Многоязычная	Русскоязычная
Модель распространения	Коммерческая	Коммерческая	Freemium

Внутренний портал компании является ключевым элементом цифровой инфраструктуры, решающим задачи повышения эффективности, коммуникации и автоматизации работы команды. Анализ готовых решений выявил три основных типа платформ: мощные корпоративные системы («SharePoint»), ориентированные на коллаборацию («Confluence») и универсальные бизнес-инструменты («Bitrix24»). Все они предлагают схожий базовый функционал: управление документами, задачами и коммуникацией. Исходя из этого можно сформировать требования к ключевым функциональным модулям:

- система управления документами: хранение, версионность, поиск и использование шаблонов документов;
- портал новостей и коммуникаций: лента корпоративных новостей, анонсы и опросы;
- инструменты коллаборации: профили сотрудников, командные чаты;
- база знаний: внутренние регламенты, инструкции и база часто задаваемых вопросов.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Описание функциональности программного средства

2.1.1 Варианты использования

Разрабатываемая система представляет собой корпоративный портал, предназначенный для организации совместной работы в рамках компании. Функциональность системы структурирована по ролевому принципу, где каждый следующий уровень наследует права и возможности предыдущего, что обеспечивает гибкое и безопасное распределение обязанностей. Графическое отображение вариантов использования программного средства представлено на Use Case диаграмме (см. Рисунок 2.1).

Базовые возможности, такие как регистрация и авторизация в системе, доступны любому интернет-пользователю. После успешной регистрации он получает статус пользователя, что предоставляет ему право на создание новой организации, становясь ее первым сотрудником и супер-администратором.

Основной рабочей единицей системы является сотрудник. В его зону ответственности входит управление личным профилем, просмотр профилей коллег, команд и организации в целом. Сотрудник имеет полноценный доступ к новостной ленте, где может просматривать новости как своей команды, так и всей организации, а также осуществлять их поиск. Работа с документами реализована через централизованную базу данных: сотрудник может просматривать ее содержимое и находить нужные документы с помощью поиска.

Роль руководителя расширяет функционал сотрудника, добавляя инструменты для управления структурой определённой команды. Руководитель наделен правом редактировать профиль команды, публиковать новости для ее сотрудников, а также добавлять новые документы в общую базу.

Следующий уровень – уровень администратора – обладает полным контролем над организацией. Помимо функций руководителя, администратор может создавать новые команды, редактировать профиль организации и публиковать новости общеорганизационного масштаба. В его обязанности входит управление кадрами: он может отправлять приглашения новым сотрудникам через специальные ссылки и удалять учетные записи действующих сотрудников.

Наибольшими правами в системе обладает супер-администратор, который может назначать администраторов из числа сотрудников, делегируя им часть полномочий для эффективного управления крупной организационной структурой.

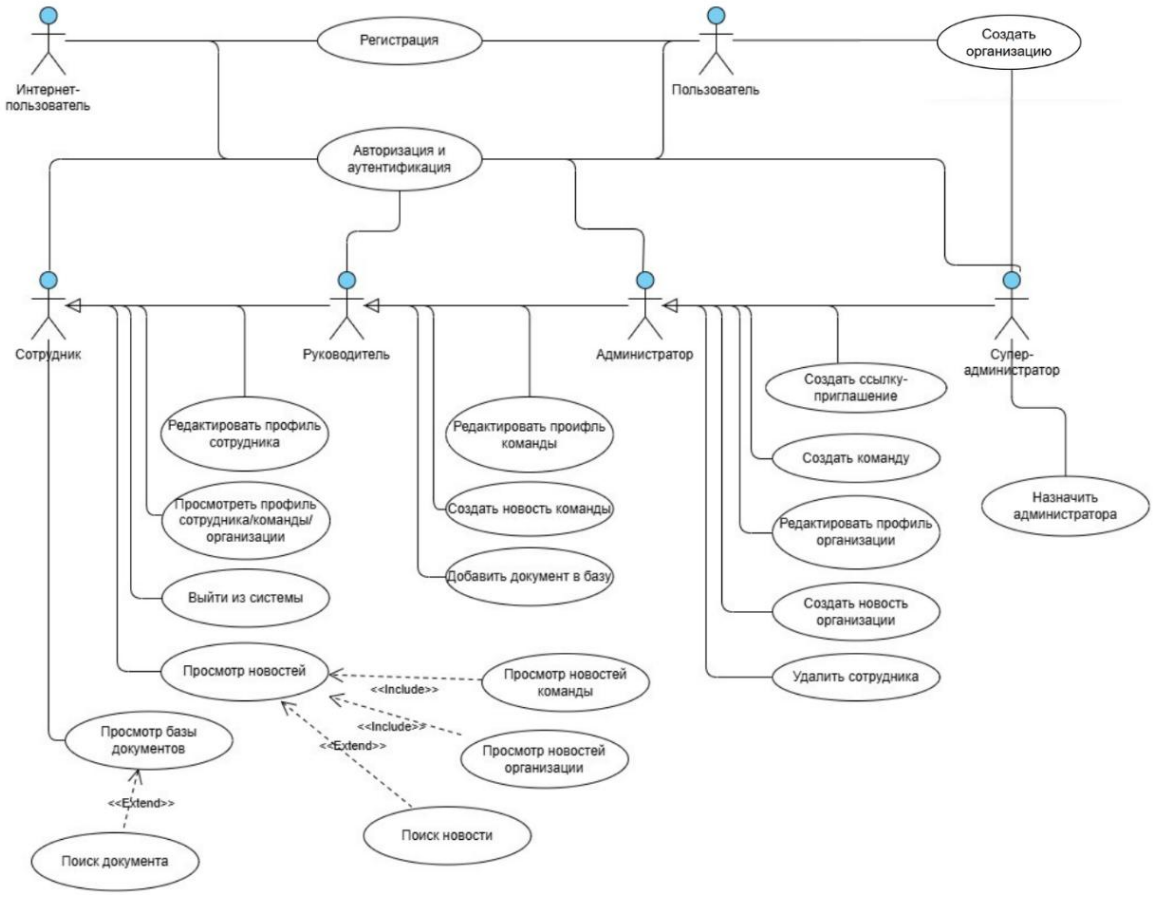

Рисунок 2.1 – Use Case диаграмма

2.1.2 Информационная модель предметной области

В основе проектируемой системы лежит реляционная модель данных, отражающая ключевые бизнес-сущности корпоративного портала и связи между ними (см. Рисунок 2.2). Модель спроектирована для поддержки ролевого доступа и иерархической структуры «Организация → Команда → Сотрудник».

Ключевые сущности:

1) Учётная запись: хранит данные пользователей системы (учётные данные, контактную информацию, фото). Связь с сущностью «Роль» определяет права пользователя в рамках организации.
2) Организация и Команда: описывают структуру компании. Команда всегда принадлежит организации и имеет своего руководителя и создателя (администратор).

3) Новость и Документ: основные типы контента. Важной особенностью модели является возможность привязки этих сущностей как к конкретной команде, так и непосредственно к организации, что позволяет реализовать новости и документы как общеорганизационного, так и командного уровня.
4) Ключевое слово: реализует механизм тегов для категоризации и поиска публикаций.

Модель определяет, что пользователи создают организации и входят в их состав, организации владеют командами, а пользователи публикуют контент (новости и документы) в рамках своих организаций и команд. Для разграничения прав доступа используется связь учётных записей с ролями.

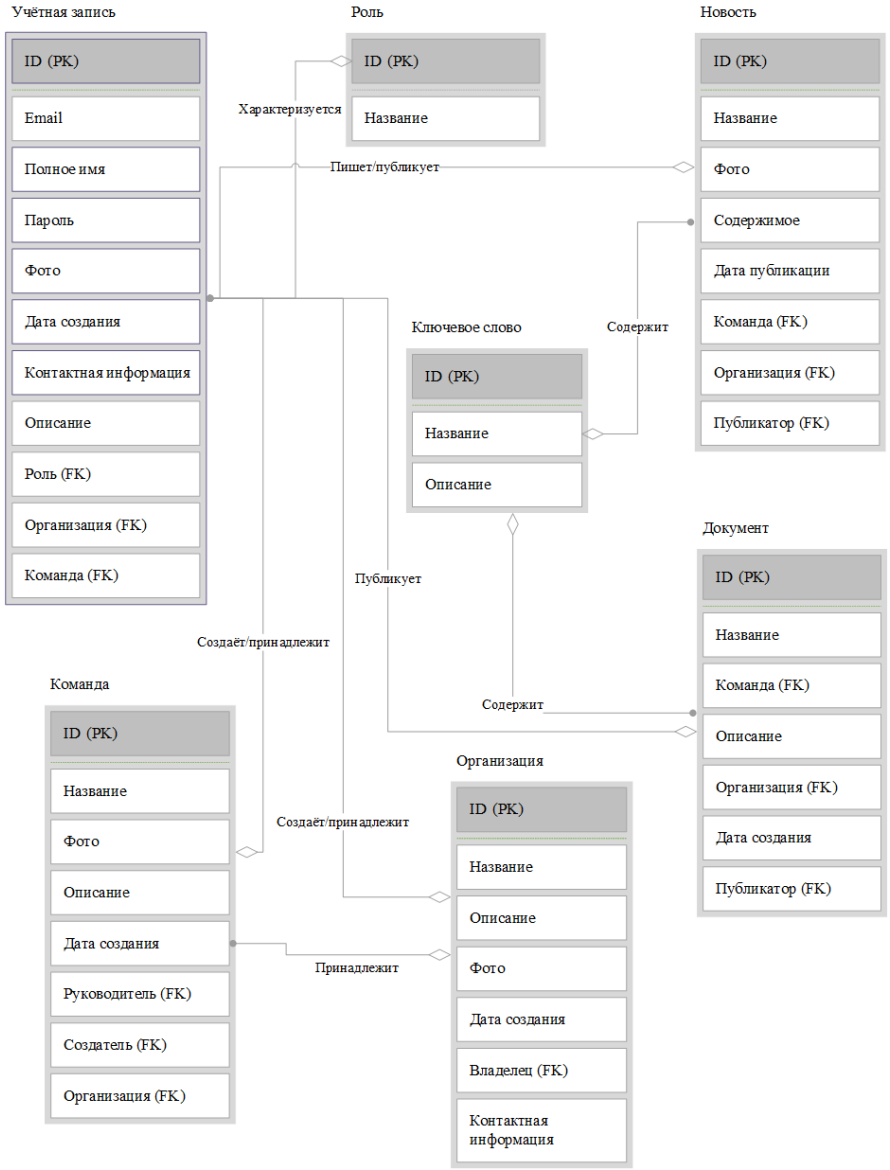

Рисунок 2.2 – Информационная модель предметной области

2.2 Спецификация функциональных требований

Ниже приведена детальная спецификация функций, которые должна реализовывать система. Требования сгруппированы по логическим модулям, охватывающим все основные бизнес-процессы корпоративного портала.

2.2.1 Модуль аутентификации и авторизации

1) Система должна предоставлять возможность регистрации нового пользователя:

– при регистрации пользователь должен предоставить уникальный адрес электронной почты, пароль и полное имя;
– система должна проверять уникальность адреса электронной почты.

2) Система должна предоставлять возможность авторизации и аутентификации зарегистрированного пользователя:

– аутентификация осуществляется по паре логин (электронная почта) и пароль;
– авторизация осуществляется на основе информации в базе данных.

2.2.2 Модуль управления организациями, командами и сотрудниками

1) Аутентифицированный пользователь должен иметь возможность создать организацию:

– при создании организации пользователь становится её супер-администратором.

2) Администратор организации должен иметь возможность создавать команды внутри своей организации:

– при создании команды обязательно указывается её название и руководитель.

3) Добавление сотрудника в организацию осуществляется через ссылку-приглашение, которая генерируется администратором или супер-администратором:

– ссылка-приглашение высылается на почту, которая впоследствии должна быть указана в профиле сотрудника;
– ссылка-приглашение должна иметь ограниченный срок действия и количество применений;
– система должна предоставлять возможность просматривать список активных ссылок и менять их статус (активная/не активная).

2.2.3 Модуль управления профилями

1) Сотрудник должен иметь возможность просматривать и редактировать собственный профиль.

2) Руководитель команды должен иметь возможность редактировать профиль своей команды.

3) Администратор и супер-администратор организации должны иметь возможность редактировать все профили внутри организации и профиль самой организации.

4) До тех пор, пока пользователь не является сотрудником, т.е. не принадлежит какой-либо организации, он может редактировать все поля своего профиля и удалить его.

Редактируемыми полями для профиля являются: фотография, описание, контактная информация. Администратор и супер-администратор должны иметь возможность редактировать все поля любом профиле в организации.

2.2.4 Модуль управления новостями

1) Система должна предоставлять возможность просмотра новостной ленты:

- новостная лента должна содержать новости организации и команды, к которой принадлежит сотрудник;
- в новостной ленте отображаются название новости, картинка (опционально), ключевые слова, дата публикации и источник (организация/команда);
- должна быть реализована возможность просмотра всего содержимого новости после нажатия на неё в новостной ленте;
- должна быть обеспечена возможность фильтрации новостей по источнику (организация/команда) и по дате публикации (от и до конкретной даты);
- должна быть предоставлена возможность поиска новости по ключевым словам.

2) Руководитель команды должен иметь возможность создавать и публиковать новости для своей команды.

3) Администратор должен иметь возможность создавать новость для всей организации.

4) После создания новость нельзя редактировать, только просматривать (определённые сотрудники) и удалять (публикатор или администратор).

2.2.5 Модуль управления документами

1) Система должна предоставлять единую документную базу организации:

- должен быть реализован просмотр списка документов;
- в списке документов отображается название документа, дата публикации, автор и ключевые слова;
- должна быть предоставлена возможность поиска документа по названию и ключевым словам.

2) Руководители и администраторы должны иметь возможность добавлять новые документы в общую базу документов:

- при загрузке документа указываются его название, описание, ключевые слова, прикрепляется файл документа или ссылка на него;
- система должна фиксировать автора документа и дату публикации.

2.2.6 Модуль администрирования и управления правами

1) Супер-администратор должен иметь возможность назначать администраторов из числа сотрудников и снимать их с должности администратора.

2) Администратор должен иметь возможность назначать руководителей команд из числа сотрудников организации и снимать их с должности руководителя.

3) Администратор должен иметь возможность удалять сотрудника из организации:

- при удалении сотрудника из организации его учётная запись деактивируется в рамках организации, но не удаляется из системы, т.е. сотрудник переходит в статус пользователя.

4) Система должна обеспечивать наследование прав по иерархии: Сотрудник $\Leftarrow$ Руководитель $\Leftarrow$ Администратор $\Leftarrow$ Супер-администратор. Каждая роль включает возможности предыдущей.

5) Супер-администратор должен иметь возможность удаления организации:

- после удаления организации все её сотрудники переходят в статус пользователей системы.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного средства

Разрабатываемый корпоративный портал построен по трехуровневой клиент-серверной архитектуре с четким разделением ответственности между компонентами.

Диаграмма компонентов (см. Рисунок 3.1) отображает структурное разбиение системы на основные логические компоненты и определяет интерфейсы взаимодействия.

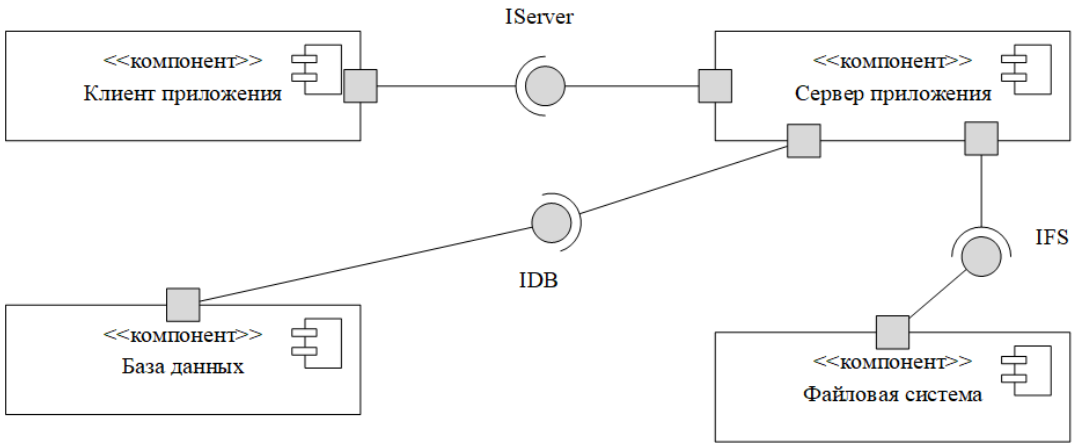

Рисунок 3.1 – Диаграмма компонентов

Основу архитектуры составляют четыре ключевых компонента: «Клиент приложения», «Сервер приложения», «База данных» и «Файловая система».

3.1.1 Компонент «Клиент приложения»

Компонент «Клиент приложения» представляет собой веб-приложение, которое работает в браузере пользователя и обеспечивает его связь с системой. Компонент отвечает за отображение данных и получение пользовательского ввода, при этом вся бизнес-логика делегирована серверной части.

Клиент требует интерфейс «IServer» для выполнения операций, связанных с управлением профилями, работой с новостной лентой, поиском документов и администрированием организационной структуры в зависимости от роли пользователя.

3.1.2 Компонент «Сервер приложения»

Сервер приложения является ядром системы, реализующим всю бизнес-логику портала. Именно здесь происходит проверка прав доступа в

соответствии с ролевой моделью и координация работы подсистем. Сервер предоставляет интерфейс «IServer» для клиентского приложения и сам требует два интерфейса для работы с данными: «IDB» для доступа к структурированной информации и «IFS» для работы с документами.

3.1.3 Компонент «База данных»

База данных обеспечивает надежное хранение структурированных данных через интерфейс «IDB». Здесь хранится информация о пользователях, организационной структуре, правах доступа, метаданных документов и новостях.

3.1.4 Компонент «Файловая система»

Файловая система необходима для хранения документов и медиафайлов через интерфейс «IFS». Этот компонент отвечает за хранение различных типов файлов, включая фотографии профилей, изображения для новостей, документы корпоративной базы знаний и другие файловые ресурсы.

Взаимодействие компонентов происходит по следующему принципу: запрос от клиентского приложения поступает на сервер через интерфейс «IServer», где проходит проверку авторизации и прав доступа. Для выполнения операции сервер обращается к базе данных через «IDB» для работы со структурированными данными и/или к файловой системе через «IFS» для работы с документами.

3.2 Модель базы данных

База данных была спроектирована для системы коллективной работы и управления контентом с поддержкой мультитенантной архитектуры. Она представляет собой реляционную модель, построенную на системе MySQL, и включает в себя десять взаимосвязанных таблиц, которые обеспечивают хранение и управление данными в рамках изолированных организационных пространств. Ключевой особенностью архитектуры является четкое разделение данных между различными организациями, что позволяет обеспечить безопасность и конфиденциальность информации каждой компании-клиента.

Основу базы данных составляют сущности, описывающие пользователей, организационную структуру и контент системы.

Сущность «organizations». Является центральной сущностью для реализации мультитенантной архитектуры. Каждая организация представляет собой изолированное рабочее пространство с собственной структурой пользователей и контента. Основные атрибуты: id (уникальный идентификатор), name (название), owner_id (идентификатор владельца организации, т.е. супер-администратора), contact_info (контактная информация).

Сущность «accounts». Хранит учетные данные и профильную информацию пользователей системы. Каждый пользователь обязательно принадлежит к одной организации и имеет определенную роль в системе. Основные атрибуты: email (электронная почта, на которую была выслана ссылка-приглашение на добавление в организацию), password_hash (хэш паролья), role_id (связь с ролью пользователя в системе).

Подробное описание всех сущностей представлено в таблице 3.1.

Таблица 3.1 – Сущности, атрибуты, связи

Название таблицы	Краткое описание	Атрибуты	Связи
accounts	Хранение данных пользователей системы для аутентификации и авторизации	id, email, full_name, password_hash, photo_path, created_at, contact_info, description, role_id, organization_id, team_id	organizations (organization_id), roles (role_id), teams (team_id)
organizations	Хранение информации о компаниях (мультитенантность)	id, name, description, photo_path, created_at, owner_id, contact_info	accounts (owner_id)
teams	Группа пользователей внутри организации для совместной работы	id, name, photo_path, description, created_at, leader_id, creator_id, organization_id	accounts (manager_id, creator_id), organizations (organization_id)
publications	Основной контент системы – статьи, новости, блог-посты	id, title, photo_path, content, published_at, team_id, organization_id, publisher_id	teams (team_id), organizations (organization_id), accounts (publisher_id)
documents	Документы в общей базе документов	id, title, team_id, description, organization_id, created_at, publisher_id	teams (team_id), organizations (organization_id), accounts (publisher_id)
keywords	Система тегирования для	id, name, description	–

	категоризации контента		

Продолжение Таблицы 3.1

publication_keywords	Связь многие-ко-многим между публикациями и ключевыми словами	id, publication_id, keyword_id	publications (publication_id), keywords (keyword_id)
publication_authors	Связь многие-ко-многим между публикацией и авторами	id, publication_id, author_id	publications (publication_id), accounts (author_id)
document_keywords	Связь многие-ко-многим между документами и ключевыми словами	id, document_id, keyword_id	documents (document_id), keywords (keyword_id)

Схема базы данных с указанием всех сущностей, их связей и типов атрибутов представлена на рисунке 3.2.

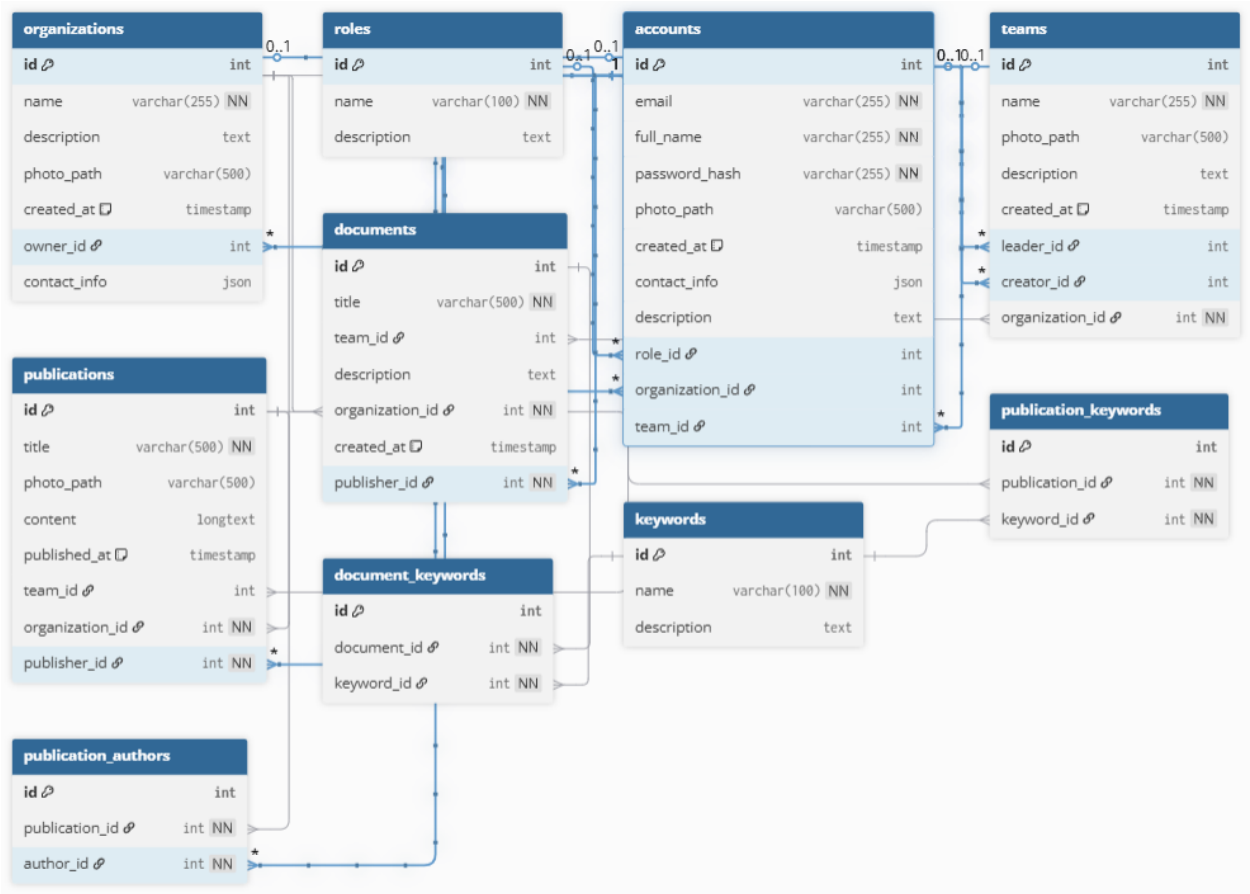

Рисунок 3.2 – Схема базы данных

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

В разрабатываемой системе «Корпоративный портал» логика работы с данными распределена между системой управления базами данных (СУБД) и веб-приложением. Такой архитектурный подход позволяет равномерно распределять нагрузку на составляющие системы и обеспечивать валидацию данных на разных уровнях.

Разделение ответственности между клиентской частью, сервером приложения и базой данных обеспечивает надежность, согласованность и производительность системы. Проверки на уровне клиентского приложения (валидация форм в React) позволяют быстро отсеивать некорректные данные и предоставлять пользователю мгновенную обратную связь, улучшая пользовательский опыт.

На уровне сервера реализуется бизнес-логика, проверка прав доступа (RBAC) и сложная валидация данных перед записью. В то же время ограничения, внешние ключи и уникальные индексы в СУБД гарантируют целостность данных на физическом уровне, исключая появление дубликатов или битых ссылок даже в случае ошибок в программном коде.

4.1 Уточнение выбора средств разработки

Для реализации программного комплекса был выбран современный стек технологий, обеспечивающий высокую скорость разработки, надежность и масштабируемость.

В качестве основного языка программирования серверной части выбран Go (Golang). Встроенные механизмы (горутины) позволяют эффективно обрабатывать множество одновременных HTTP-запросов, что критично для многопользовательского корпоративного портала. Для разработки веб-сервера используется фреймворк Gin Web Framework. Он предоставляет быстрый HTTP-роутер, удобный механизм middleware (используется для авторизации и CORS) и встроенные средства для валидации JSON-запросов. Взаимодействие с базой данных реализовано через ORM-библиотеку GORM. Использование ORM позволило оперировать данными как объектами, автоматизировать создание и миграцию таблиц базы данных и упростить написание сложных запросов.

Клиентское приложение реализовано с использованием библиотеки React.js. React позволяет создавать интерактивный пользовательский интерфейс на основе переиспользуемых компонентов (карточки новостей, модальные окна, формы), что значительно упрощает поддержку и расширение кода.

Для создания визуального интерфейса выбрана библиотека компонентов Material UI. Она предоставляет готовый набор адаптированных под мобильные и десктопные устройства компонентов (Grid, Card, Dialog, TextField), выполненных в едином дизайн-коде Google Material Design. Это

позволило сократить время на верстку и сосредоточиться на реализации бизнес-логики.

4.2 Разработка программных интерфейсов

Реализация программных интерфейсов системы выполнена в соответствии с компонентной диаграммой, разработанной на этапе проектирования. Взаимодействие компонентов обеспечивается за счет использования стандартных протоколов обмена данными и специализированных библиотек.

4.2.1 Реализация интерфейса IServer («Клиент – Сервер»)

Интерфейс взаимодействия между клиентским приложением (React) и сервером приложений (Go) реализован на основе архитектурного стиля REST. Обмен данными происходит по протоколу HTTP/1.1. В качестве формата сериализации данных выбран JSON.

Маршрутизация запросов осуществляется с помощью библиотеки Gin Web Framework. В файле сервера main.go настроен роутер, который распределяет входящие HTTP-запросы по соответствующим методам-обработчикам (handlers). Для строгого соблюдения контракта интерфейса используются DTO. Часть схемы взаимодействия архитектурных слоёв, реализующих интерфейс представлена на рисунке 4.1. Для защиты интерфейса реализован механизм промежуточного ПО (middleware). Все запросы к защищенным маршрутам (группа /api/protected) проходят через метод, который проверяет наличие и валидность JWT-токена в заголовке запроса.

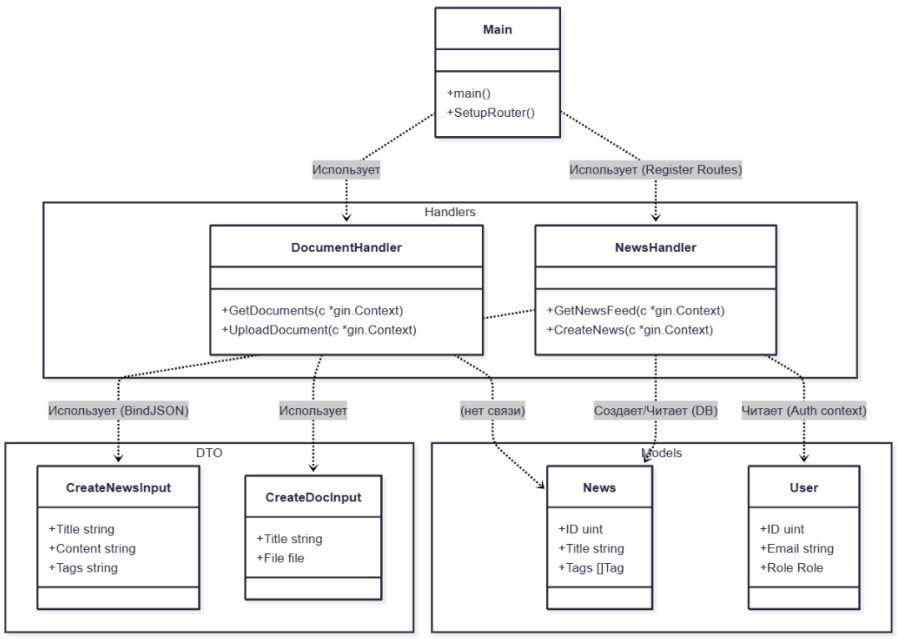

Рисунок 4.1 – Часть схемы взаимодействия архитектурных слоёв

С стороны клиента взаимодействие осуществляется асинхронно с использованием API fetch. Для управления состоянием запросов и глобальной обработки ошибок сети используется компонент с контекстом авторизации, в который обёрнуто приложение.

4.2.2 Реализация интерфейса IDB («Сервер – База данных»)

Интерфейс взаимодействия с базой данных реализован через уровень абстракции (ORM) на базе библиотеки GORM, использующей в качестве системы управления данными встраиваемую реляционную СУБД. Реализован подход «Code First». При запуске приложения вызывается метод, который синхронизирует структуру Go-моделей со схемой базы данных.

4.2.3 Реализация интерфейса IFS («Сервер – Файловая система»)

Передача файлов от клиента к серверу осуществляется с использованием кодировки multipart/form-data. В коде обработчиков (например, UploadDocument) используется метод, который извлекает файловый поток из тела HTTP-запроса. Для предотвращения коллизий имен файлов реализован алгоритм генерации уникального имени: к оригинальному названию файла добавляется уникальный префикс (UUID) и производится очистка имени. Сохранение на диск выполняется методом, который использует системные вызовы ОС для записи байтового потока в директорию ./uploads.

4.3 Реализация функциональных требований

4.3.1 Модуль управления регистрацией и авторизацией

Модуль обеспечивает безопасность системы, реализую аутентификацию пользователей и разграничение прав доступа. При регистрации данные пользователя валидируются, а пароль хэшируется алгоритмом bcrypt перед сохранением в БД. При входе в систему сервер проверяет хэш пароля и, в случае успеха, генерирует подписанный JWT-токен, содержащий идентификатор и пароль пользователя.

```
func AuthMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        authHeader := c.GetHeader("Authorization")
        claims, err := utils.ParseToken(tokenString)
        if err != nil {
            c.AbortWithStatusJSON(http.StatusUnauthorized,
gin.H{"error": "Invalid token"})
            return
        }
        c.Set("userID", claims.UserID)
        c.Set("role", claims.Role)
        c.Next()
    }
}
```

Промежуточное ПО перехватывает запросы к защищённым ресурсам, извлекая JWT-токен из заголовка и проверяя его цифровую подпись и срок действия. Если токен валиден, данные пользователя добавляются в контекст запроса. На клиенте также работают перехватчики, добавляющие или извлекающие токен из всех запросов.

```
api.interceptors.request.use((config) => {
    const token = localStorage.getItem('token');
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
});
```

При истечении сессии, клиент автоматически выполняет выход из системы и перенаправляет пользователя на страницу авторизации.

```
export default function ProtectedRoute({ children }) {
  const { user } = useAuth();
  if (!user) return <Navigate to="/login" />;
  return children;
}
```

4.3.2 Модуль управления организациями, командами и сотрудниками

В основе модуля лежит ролевая модель доступа (Role-Based Access Control), реализованная на уровне базы данных и Middleware. В системе определены следующие роли:

- SuperAdmin (Владелец, уровень 4): Создатель организации. Имеет полные права, включая управление другими администраторами и удаление организации.
- Admin (Администратор, уровень 3): Управляет структурой организации (создание команд, генерация приглашений, редактирование данных организации).
- Manager (Руководитель, уровень 2): Лидер конкретной команды. Может управлять составом своей команды (добавлять/удалять сотрудников).
- User/Employee (Сотрудник, уровень 1): Базовая роль. Имеет доступ к просмотру профилей и новостей своей команды/организации.

При создании организации или команды, при редактировании профилей и при смене ролей используются транзакции, чтобы предотвратить несогласованность данных в разных частях системы. Например, при смене руководителя команды происходит обновление статусов сразу двух пользователей:

1. Система проверяет старого руководителя, если он не руководит другими командами, его роль понижается до базовой (1).

2. Новый руководитель назначается на должность, его роль повышается (2).
3. Все изменения выполняются в рамках одной транзакции для предотвращения ситуаций, когда у команды нет руководителя или у одного из пользователей остались дополнительные права.

```
if team.LeaderID != nil {
    if err := tx.Model(&models.Team{}).
        Where("leader_id = ? AND id != ?", oldLeader.ID,
team.ID).
        Count(&otherTeamsCount).Error; err == nil &&
otherTeamsCount == 0 {
        tx.Model(&oldLeader).Update("role",
models.RoleUser)
    }
}

if newLeader.Role < models.RoleManager {
    tx.Model(&newLeader).Update("role",
models.RoleManager)
}
team.LeaderID = input.LeaderID
```

Для добавления сотрудников в организацию используется система токенизированных ссылок-приглашений. При переходе по ссылке-приглашению или при вводе таковой с аккаунта пользователя, который не состоит в других организациях, система должна гарантировать, что лимит использования ссылки не будет превышен. Для этого, когда транзакция считывает данные приглашения, она блокирует эту строку в базе данных до завершения операции.

```
tx := database.DB.Begin()
var invite models.Invite
if err := tx.Set("gorm:query_option", "FOR UPDATE").
    Where("token = ?", token).First(&invite).Error; err !=
nil {
    tx.Rollback()
    return
}
if invite.Uses >= invite.MaxUses {
    tx.Rollback()
    c.JSON(http.StatusBadRequest, gin.H{"error": "Invite
limit reached"})
    return
}
tx.Model(&invite).Update("uses", gorm.Expr("uses + 1"))
//обновление пользователя
tx.Commit()
```

Контент, отображаемый на страницах, адаптируется под роль пользователя. Например, панель генерации ссылок-приглашений отображается только пользователям, чей статус не ниже статуса Администратора.

```
    const isOwner = organization?.owner_id === user?.id; // 
Владелец
    const isAdmin = user?.role >= 3; // Администратор
    const canEdit = organization && user && (isOwner || isAdmin);

    {canEdit && (
      <Box mt={4}>
        <Typography                              variant="h6">Панель 
администратора</Typography>
        {/* карточки действий*/}
      </Box>
    )}
```

После внесения изменений в состав организации или команды для обновления состояния структуры и отображения её актуального состояния. выполняется запрос на сервер.

4.3.3 Модуль управления профилями

Управление профилями также осуществляется на основе роли пользователя, который просматривает определённый профиль. При отображении информации о профиле осуществляется проверка, является ли пользователь владельцем профиля либо администратором. Если является, то у него появляется возможность редактировать данные профиля. Системные поля (роль и принадлежность к команде) могут редактировать только администраторы.

```
    isSelf := target.ID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isSelf && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission 
denied"})
        return
    }
    if _, hasBio := c.Request.Form["bio"]; hasBio {
        updates["bio"] = bio
    }
    if isAdmin {
        if _, hasRole := c.Request.Form["role"]; hasRole {
            // Валидация и обновление роли
            updates["role"] = roleInt
        }
        //логика перевода в другую команду
    }
```

Такой же принцип работает при редактировании профиля команды. Пользователь должен быть либо руководителем команды, либо администратором.

Для каждого профиля есть возможность установить аватар. Файлы аватаров сохраняются в файловой системе. При обновлении аватара, старый удаляется.

4.3.4 Управление новостями и документами

Модули управления новостями и документами реализуют функционал единого информационного пространства организации. Архитектурно оба модуля построены на схожих принципах: использование реляционной базы данных для хранения метаданных, файловой системы для хранения бинарных данных (изображения, файлы документов) и гибкой системы фильтрации контента на основе роли пользователя и его принадлежности к команде.

На базовом уровне просмотра списка новостей/документов выбираются записи, принадлежащие организации пользователя. Если пользователь сотрудник или руководитель, то ему отображаются также принадлежащие команде. Администраторы видят весь контент.

К выборке динамически добавляются условия поиска по заголовку, содержимому и связанным тэгам.

При создании записи (handler CreateNews или UploadDocument) сервер проверяет флаг for_team. Если он установлен, проверяется членство пользователя в команде. Если флаг не установлен (глобальная публикация), проверяется роль пользователя.

```
    if input.ForTeam {
        if user.TeamID == nil {
            c.JSON(http.StatusBadRequest,  gin.H{"error":  "You
are not in a team"})
            return
        }
        isTeamLeader := user.Team != nil && user.Team.LeaderID !=
nil && *user.Team.LeaderID == user.ID
        if role < models.RoleAdmin && !isTeamLeader {
            c.JSON(http.StatusForbidden,  gin.H{"error":  "Only
team leaders can post team news"})
            return
        }
        news.TeamID = user.TeamID
    } else {
        if role < models.RoleAdmin {
            c.JSON(http.StatusForbidden,  gin.H{"error":  "Only
admins can post global news"})
            return
        }
    }
```

В отличие от новостей, документы требуют физического сохранения файлов на сервере. В модуле handlers/documents.go реализован механизм безопасной загрузки:

- для предотвращения коллизий и сохранения читаемости имени при скачивании, используется гибридная стратегия именования: {ShortUUID}_{OriginalName};
- ссылка на файл снабжается атрибутом download, который подсказывает браузеру сохранить файл под его корректным названием.

4.3.5 Управление правами доступа

Управление правами доступа в системе реализовано динамически. Привилегии пользователя определяются совокупностью двух факторов: его глобальной ролью и принадлежностью к команде. Изменение любого из этих параметров влечет за собой пересчет доступных действий.

Функционал прямого назначения ролей реализован в методе UpdateUserRole. К операции применяются строгие ограничения.

Изменять роли других сотрудников может только Супер-администратор (Владелец организации). Это предотвращает ситуацию, когда назначенный администратор пытается захватить управление, понижая владельца.

```
if requestorRole < models.RoleSuperAdmin {
    c.JSON(http.StatusForbidden, gin.H{"error": "Only Super Admin can manage roles"})
    return
}
```

Администратор не может изменить роль самому себе. Это гарантирует, что в организации всегда останется хотя бы один пользователь с наивысшими правами.

```
if targetUser.ID == requestorID {
    c.JSON(http.StatusBadRequest, gin.H{"error": "Cannot change your own role here"})
    return
}
```

Система проверяет, что инициатор запроса и целевой пользователь находятся в одной организации, предотвращая межкорпоративное вмешательство.

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование является неотъемлемой частью жизненного цикла разработки программного обеспечения и проводится с целью проверки соответствия разработанной системы функциональным требованиям, а также выявления и устранения ошибок. В рамках данного этапа выполнялось функциональное тестирование системы методом «черного ящика», при котором проверялось поведение программы на различных входных данных без углубления во внутреннюю структуру кода.

Основное внимание уделялось проверке корректности работы серверной части (REST API), правильности выполнения транзакций в базе данных и адекватности реакции пользовательского интерфейса на действия пользователя. Результаты проверки основных сценариев использования (Use Cases) оформлены в виде тест-кейсов и представлены в таблице 5.1.

Таблица 5.1 – Основные сценарии тестирования

Тест-кейс	Ожидаемый результат	Результат
Регистрация пользователя с Email, который уже зарегистрирован в системе	Регистрация пользователя с Email, который уже зарегистрирован в системе	Ошибка СУБД (нарушение UNIQUE-ограничения индекса поля email), вставка не произведена. Приложение возвращает статус 400 и сообщение об ошибке
Авторизация пользователя с верным Email, но неверным паролем	Отказ на уровне приложения (несовпадение хеша пароля bcrypt). Токен доступа не выдан. Возвращен статус 401	Отказ на уровне приложения (несовпадение хеша пароля bcrypt). Токен доступа не выдан. Возвращен статус 401
Попытка доступа к защищенному эндпоинту (/api/me) без заголовка Authorization	Отказ на уровне Middleware. Обработка запроса прервана до обращения к БД. Возвращен статус 401	Отказ на уровне Middleware. Обработка запроса прервана до обращения к БД. Возвращен статус 401
Создание организации пользователем	Транзакция выполнена успешно: создана запись в таблице organizations, роль пользователя обновлена до 4 (SuperAdmin), проставлен organization_id	Транзакция выполнена успешно: создана запись в таблице organizations, роль пользователя обновлена до 4 (SuperAdmin), проставлен organization_id

Продолжение таблицы 5.1

Попытка создания команды пользователем с ролью «Сотрудник» (Role = 1)	Отказ на уровне бизнес-логики (проверка прав доступа role < Admin). Запись в БД не создана. Возвращен статус 403	Отказ на уровне бизнес-логики (проверка прав доступа role < Admin). Запись в БД не создана. Возвращен статус 403
Добавление в команду сотрудника, который уже состоит в другой команде	Отказ на уровне бизнес-логики (проверка targetUser.TeamID != nil). Обновление записи пользователя не произведено. Сообщение об ошибке от приложения	Отказ на уровне бизнес-логики (проверка targetUser.TeamID != nil). Обновление записи пользователя не произведено. Сообщение об ошибке от приложения
Назначение нового лидера команды (пользователь был обычным сотрудником)	Транзакция выполнена успешно: роль пользователя повышена до 2 (Manager), поле leader_id у команды обновлено	Транзакция выполнена успешно: роль пользователя повышена до 2 (Manager), поле leader_id у команды обновлено
Попытка изменения роли другого пользователя сотрудником с ролью «Менеджер» (Role = 2)	Отказ на уровне приложения (требуется роль SuperAdmin). Данные в БД не изменены. Возвращен статус 403	Отказ на уровне приложения (требуется роль SuperAdmin). Данные в БД не изменены. Возвращен статус 403
Загрузка аватара пользователя (файл формата .png)	Файл сохранен в файловой системе сервера (/uploads/avatars/...). В БД обновлен путь к файлу. Старый файл аватара удален с диска	Файл сохранен в файловой системе сервера (/uploads/avatars/...). В БД обновлен путь к файлу. Старый файл аватара удален с диска
Попытка выхода из организации владельцем (SuperAdmin)	Отказ на уровне бизнес-логики (владелец не может покинуть организацию без передачи прав). Обновление БД не произведено	Отказ на уровне бизнес-логики (владелец не может покинуть организацию без передачи прав). Обновление БД не произведено

Продолжение таблицы 5.1

Создание новости для команды пользователем, не являющимся лидером этой команды	Отказ на уровне бизнес-логики (проверка userID != Team.LeaderID). Запись не создана. Возвращен статус 403	Отказ на уровне бизнес-логики (проверка userID != Team.LeaderID). Запись не создана. Возвращен статус 403
Запрос ленты новостей сотрудником Команды А (проверка фильтрации)	SQL-запрос вернул новости организации (team_id IS NULL) и новости Команды А. Новости Команды Б в выборку не попали	SQL-запрос вернул новости организации (team_id IS NULL) и новости Команды А. Новости Команды Б в выборку не попали
Загрузка документа с именем файла, содержащим пробелы и кириллицу	Файл сохранен на диске с санитарным именем (транслитерация/замена пробелов + UUID префикс). В БД записан корректный URL	Файл сохранен на диске с санитарным именем (транслитерация/замена пробелов + UUID префикс). В БД записан корректный URL

6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

6.1 Установка программного средства

Для установки программного средства необходимо распаковать архив с исходным кодом. В проекте есть 2 папки: frontend и backend. Для запуска сервера необходимо установить зависимости проекта и запустить go-сервер. База данных будет создана автоматически при первом запуске. Соответствующие команды указаны в таблице 6.1. Для запуска клиента также нужно установить зависимости и запустить react-клиент.

Таблица 6.1 – Команды запуска приложения

Часть приложения	Команда	Ожидаемый результат	Результат
Сервер	go mod download	Установка go-зависимостей.	Обновление файлов go.sum и go.mod
Сервер	go run cmd/main.go	Запуск сервера	Listening and serving HTTP on :8080
Клиент	npm install	Установка js-зависимостей	Обновление node_modules
Клиент	npm run dev	Запуск клиента	Local: http://localhost:5173/

6.2 Руководство по использованию

6.2.1 Создание аккаунта

При входе в систему новый пользователь видит форму авторизации. Чтобы создать аккаунт нужно перейти по ссылке в конце формы. Тогда откроется форма регистрации, в которой нужно будет ввести своё имя, почту и пароль минимум из 6 символов. После успешного создания аккаунта пользователю нужно будет авторизоваться, для чего будет совершен автоматический переброс пользователя снова на форму авторизации. После успешной авторизации пользователю отобразится главная страница портала. На этой странице (рис. 6.1) можно создать новую организацию, войти в уже существующую, перейти к просмотру своего профиля или выйти из системы.

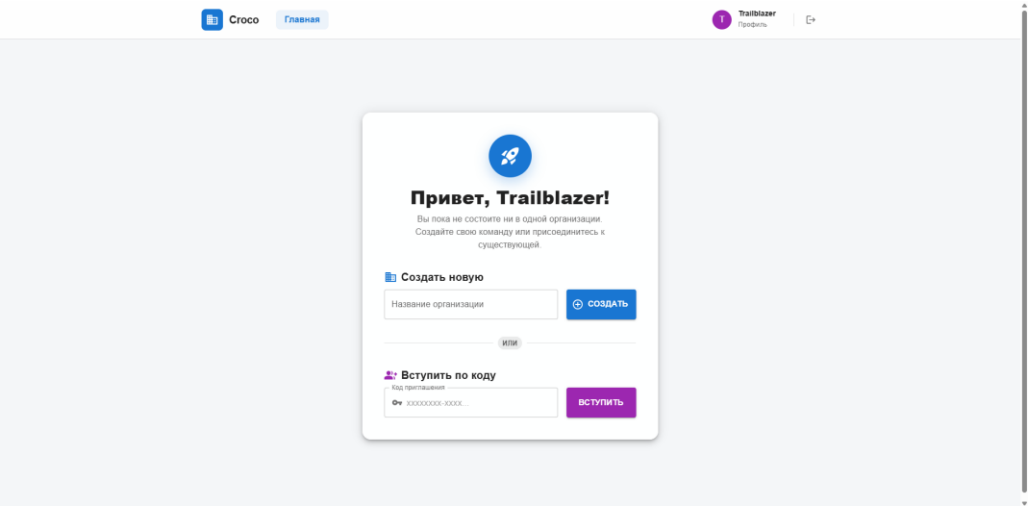

Рисунок 6.1 – Главная страница портала пользователя без организации

В профиле пользователя без организации, пример которого представлен на рисунке 6.2, отображается, что пользователь пока нигде не состоит. Можно изменить часть своей информации.

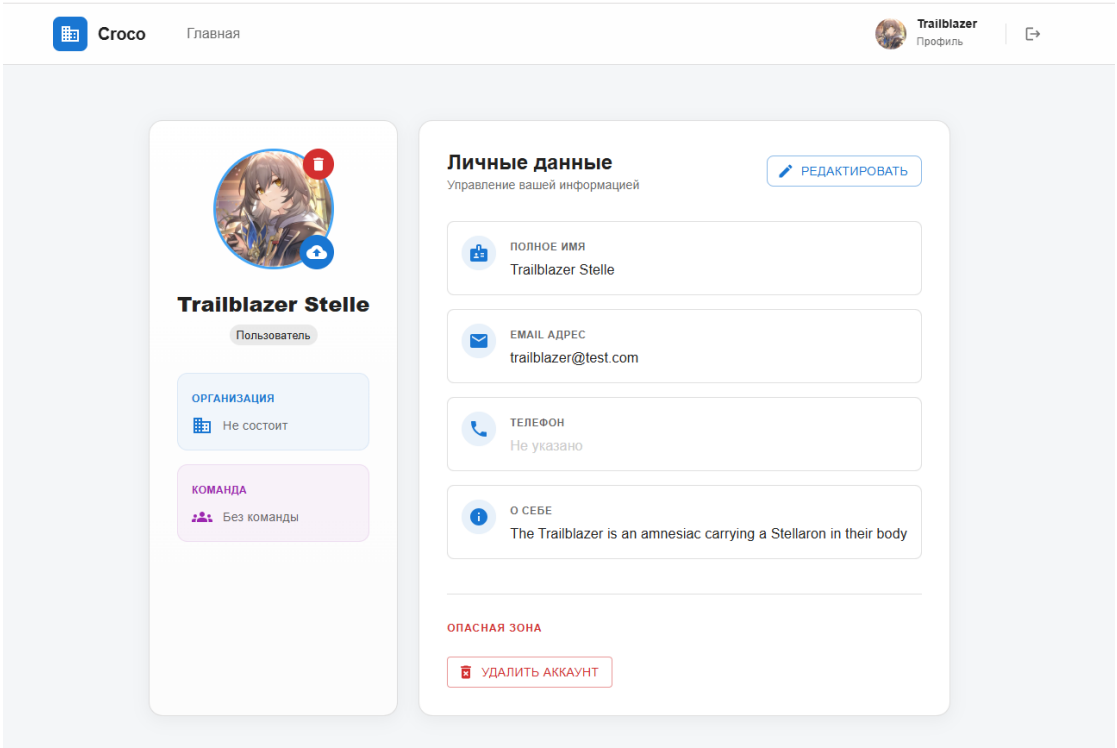

Рисунок 6.2 – Профиль пользователя с изменённой информацией

6.2.2 Создание и управление организацией

Для создания организации необходимо ввести название будущей организации на главной странице и нажать кнопку «Создать».

После создания организации пользователь перейдёт на главную страницу портала организации, пример которой представлен на рисунке 6.3. Так же у пользователя изменится статус на «SuperAdmin».

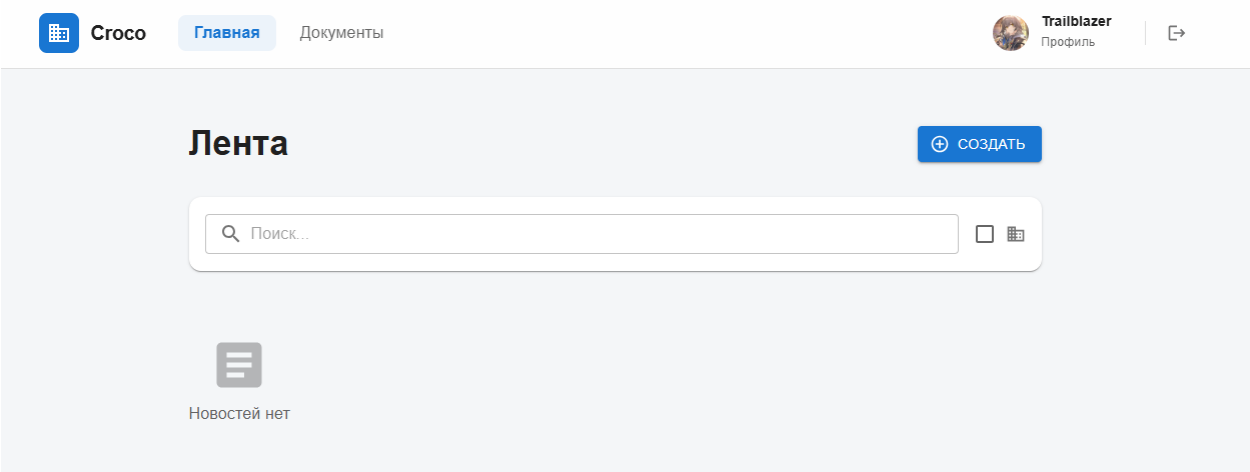

Рисунок 6.3 – Главная страница созданной организации

В профиле организации (рис. 6.4) будет отображаться количество и список активных команд и участников. Так же, поскольку текущая учётная запись является главной в организации, то будет отображаться панель администратора и возможность редактирования профиля.

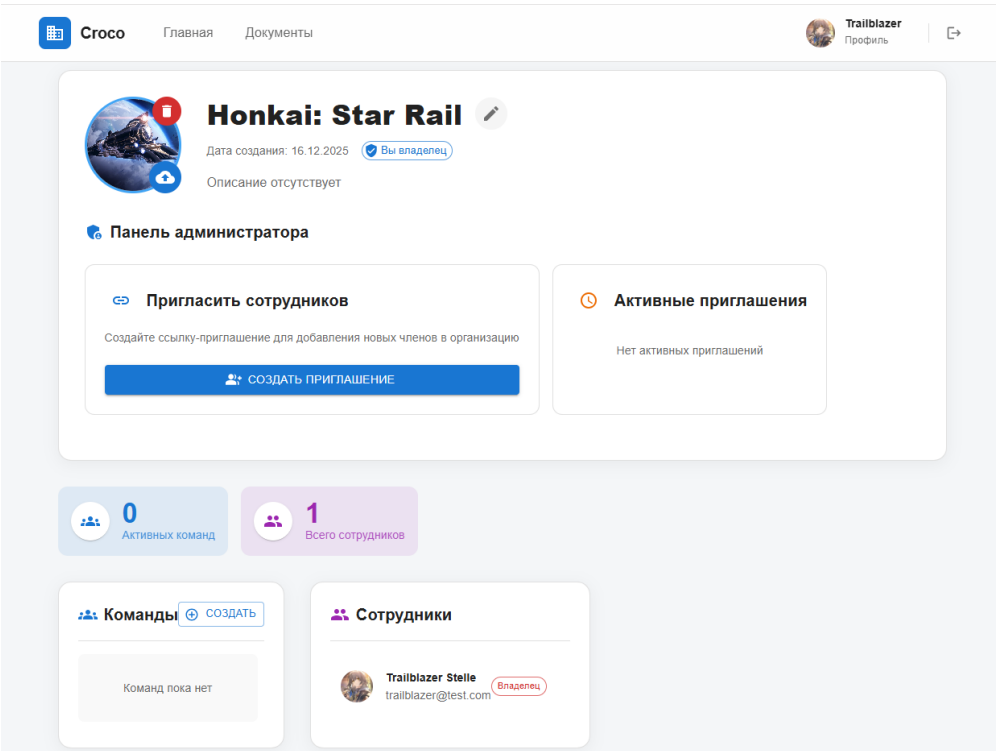

Рисунок 6.4 – Профиль организации

Для добавления нового сотрудника в команду необходимо первым делом сгенерировать ссылку приглашение на панели администратора. Токен из этой ссылки будет ключом, который должен будет ввести новый пользователь в своём аккаунте.

После добавления пользователя нового пользователя количество использований ссылки-приглашения изменится, а новый пользователь добавится в организацию. Обновлённый профиль организации представлен на рисунке 6.4.

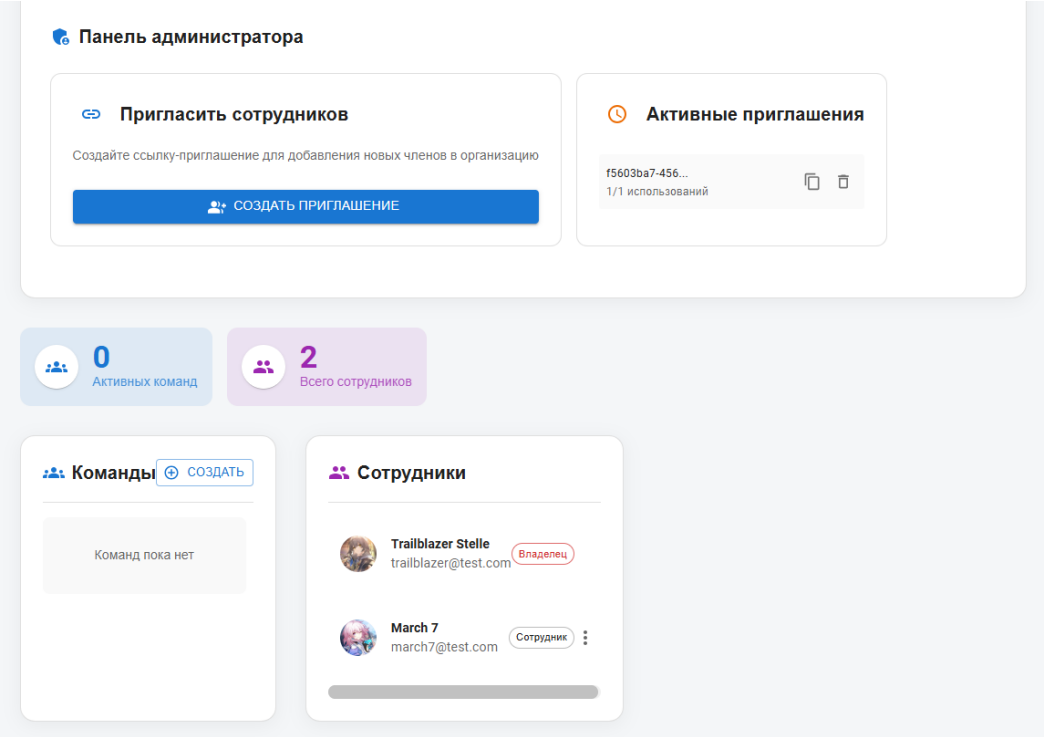

Рисунок 6.4 – Профиль организации после добавления нового сотрудника

6.2.3 Создание и управление командами

Администраторы могут создавать команды в пределах своей организации. Для этого необходимо указать название, описание (опционально) и руководителя (опционально) команды из списка сотрудников организации. Пример формы создания команды представлен на рисунке 6.3.

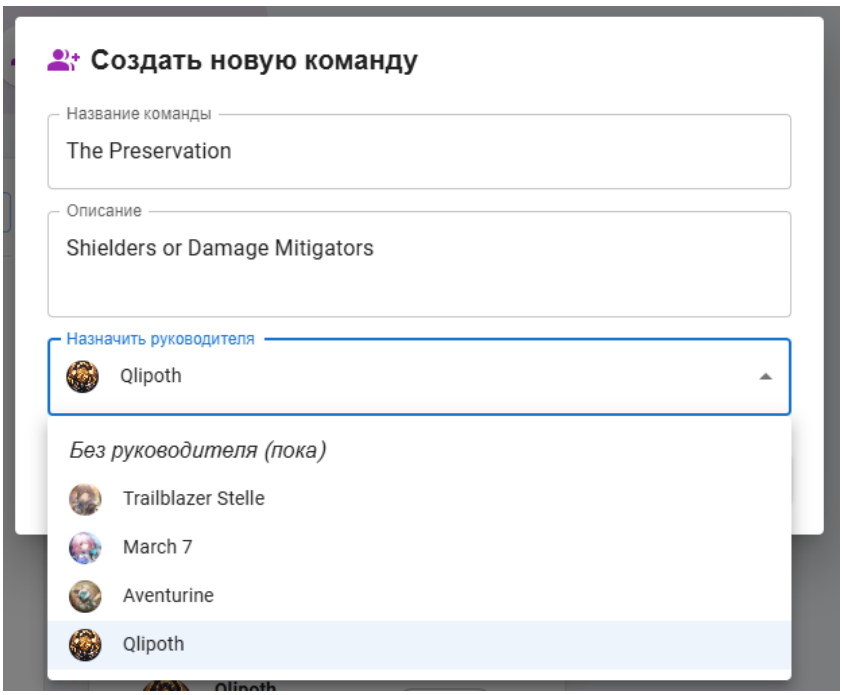

Рисунок 6.5 – Форма создания команды

После создания команды должны обновиться профили организации и роли сотрудников, а также появиться профиль команды. Обновлённый профиль организации представлен на рисунке 6.6.

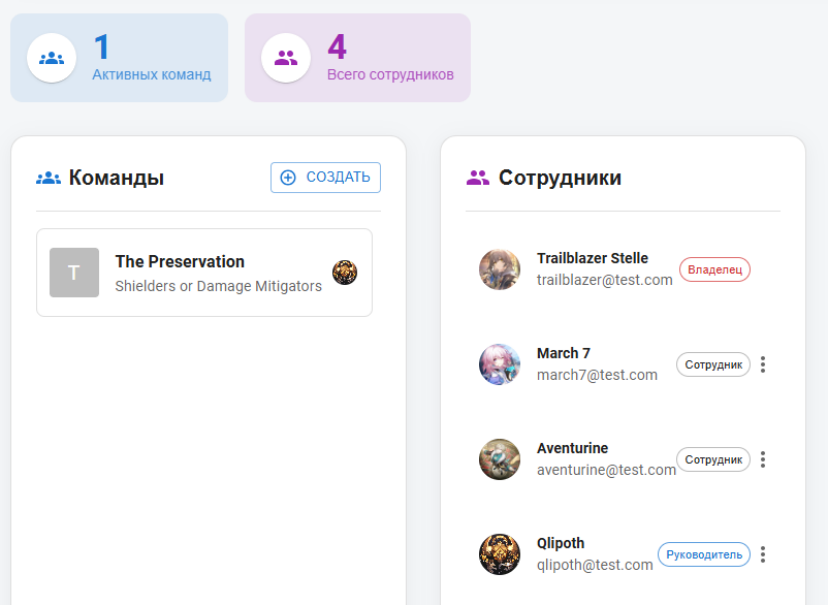

Рисунок 6.6 – Профиль организации после добавления команды

Для управления командой необходимо перейти на профиль этой команды. Далее можно изменить саму информацию о команде, сменить руководителя и добавить или удалить сотрудников из команды. Пример профиля команды представлен на рисунке 6.7.

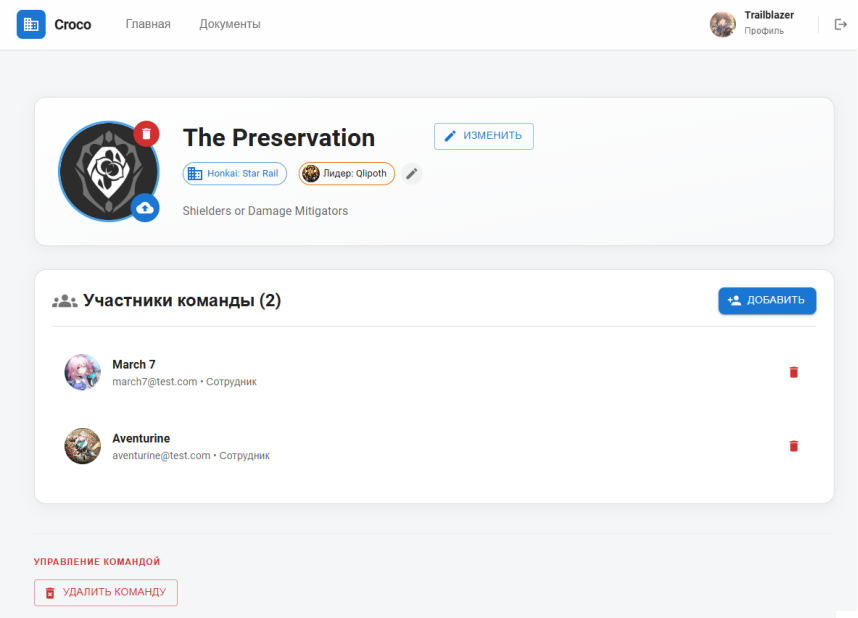

Рисунок 6.7 – Профиль одной из команд организации

6.2.4 Обновление общего хранилища организации

Для добавления документов нужно перейти на соответствующую страницу через меню. На этой странице будет отображаться поле для поиска документов, а также кнопка для добавления документа. В зависимости от роли, пользователь может создавать документы для организации или для команды. Если для команды, то в общем списке документов будут отображаться только документы организации и определённой команды. Фильтрацию можно настроить рядом с кнопкой поиска.

По такому же принципу работает поиск, создание и просмотр новостей. Формы для добавления документа и создания новости представлены на рисунках 6.8 и 6.9 соответственно.

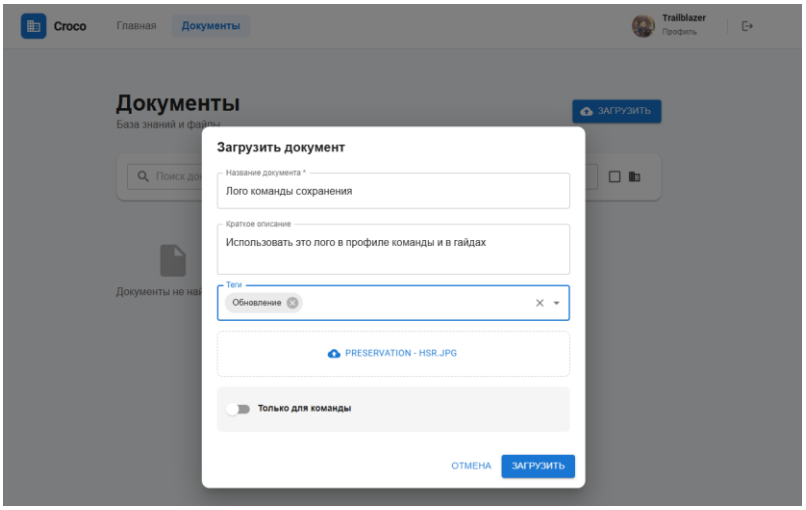

Рисунок 6.8 – Пример добавления документа для организации

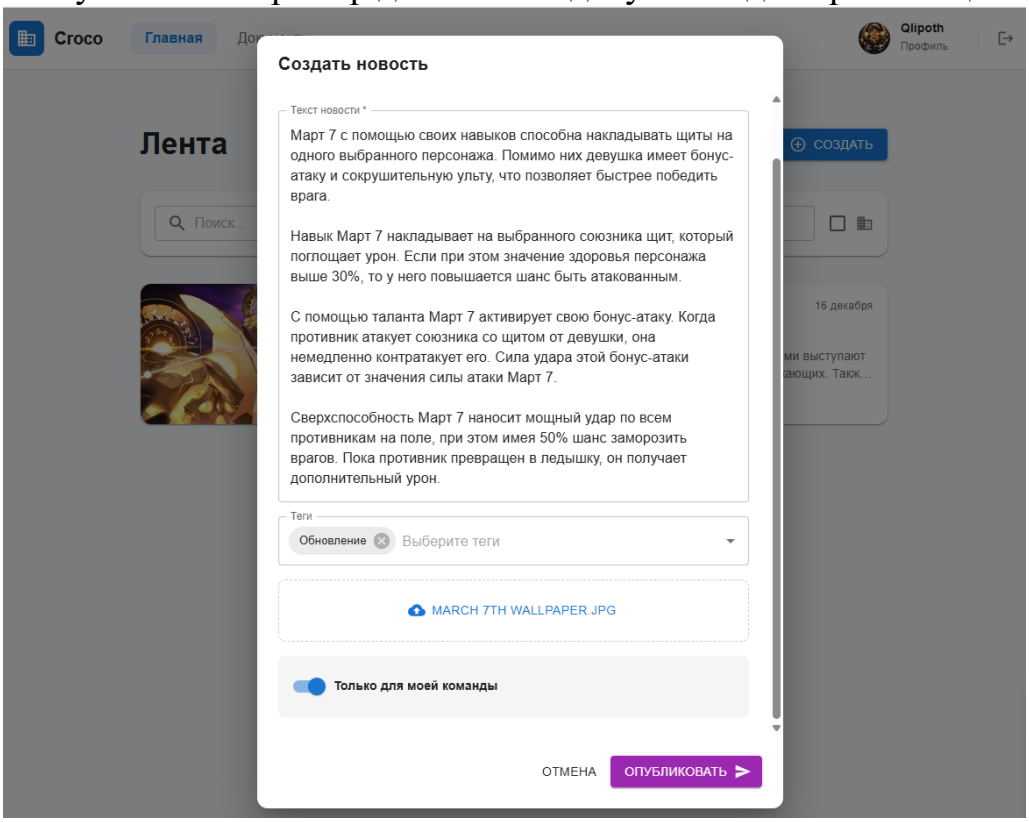

Рисунок 6.9 – Создание новости для организации

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была спроектирована и реализована информационная система «Корпоративный портал», предназначенная для централизованного управления сотрудниками, командами и внутренним контентом организации. Главная цель работы достигнута – создано единое информационное пространство, повышающее эффективность внутренних коммуникаций.

Программный комплекс построен на основе современной клиент-серверной архитектуры. Серверная часть, разработанная на языке Go с использованием фреймворка Gin, обеспечивает высокую производительность, транзакционную целостность данных и безопасность благодаря внедрению гибкой ролевой модели доступа (RBAC). Клиентское приложение на React предоставляет интуитивно понятный интерфейс для работы с новостной лентой, базой знаний и профилями сотрудников.

Разработанный продукт полностью соответствует требованиям технического задания и готов к эксплуатации. Выбранный технологический стек и модульная структура приложения обеспечивают возможности для дальнейшего масштабирования системы, например, внедрения мессенджера или интеграции с внешними сервисами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Microsoft SharePoint: Решения для управления контентом и совместной работы [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/microsoft-365/sharepoint/collaboration. Дата доступа: 30.09.2025.

[2] Atlassian Confluence: Программное обеспечение для совместной работы [Электронный ресурс]. Режим доступа: https://www.atlassian.com/software/confluence. Дата доступа: 30.09.2025.

[3] Битрикс24: Онлайн-сервис для управления бизнесом и CRM [Электронный ресурс]. Режим доступа: https://www.bitrix24.by. Дата доступа: 30.09.2025.

[4] The Go Programming Language [Электронный ресурс]. Режим доступа: https://go.dev/. Дата доступа: 10.10.2025.

[5] Gin Web Framework Documentation [Электронный ресурс]. Режим доступа: https://gin-gonic.com/docs/. Дата доступа: 10.10.2025.

[6] GORM: The fantastic ORM library for Golang [Электронный ресурс]. Режим доступа: https://gorm.io/docs/. Дата доступа: 10.10.2025.

[7] React: The library for web and native user interfaces [Электронный ресурс]. Режим доступа: https://react.dev/. Дата доступа: 29.10.2025.

[8] MUI: The React component library you always wanted [Электронный ресурс]. Режим доступа: https://mui.com/core/. Дата доступа: 29.10.2025.

[9] Vite: Next Generation Frontend Tooling [Электронный ресурс]. Режим доступа: https://vitejs.dev/. Дата доступа: 30.10.2025.

[10] Introduction to JSON Web Tokens [Электронный ресурс]. Режим доступа: https://jwt.io/introduction. Дата доступа: 2.12.2025.

[11] SQLite Home Page [Электронный ресурс]. Режим доступа: https://www.sqlite.org/index.html. Дата доступа: 2.12.2025.

[12] Axios HTTP Client Documentation [Электронный ресурс]. Режим доступа: https://axios-http.com/docs/intro. Дата доступа: 2.12.2025.

[13] MDN Web Docs: HTTP response status codes [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Дата доступа: 2.12.2025.

ПРИЛОЖЕНИЕ А

```
package main
import (
    "log"
    "os"
    "time"

    "corp-portal/internal/database"
    "corp-portal/internal/handlers"
    "corp-portal/internal/middleware"

    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
    "github.com/joho/godotenv"
)

func main() {
    if err := godotenv.Load(); err != nil {
        log.Println("No .env file found, using defaults")
    }

    database.Connect()

    r := gin.Default()
    r.Use(cors.New(cors.Config{
                    AllowOrigins:             []string{"http://localhost:3000",
"http://localhost:5173", "http://localhost:8080"},
         AllowMethods:      []string{"GET", "POST", "PUT", "DELETE", "PATCH",
"OPTIONS"},
        AllowHeaders:     []string{"Origin", "Content-Type", "Content-Length",
"Authorization", "Accept"},
        AllowCredentials: true,
        ExposeHeaders:    []string{"Content-Length"},
        MaxAge:           12 * time.Hour,
    }))

    r.Static("/uploads", "./uploads")

    api := r.Group("/api")
    {
        api.POST("/register", handlers.Register)
        api.POST("/login", handlers.Login)

        protected := api.Group("/")
        protected.Use(middleware.AuthMiddleware())
        {
            protected.GET("/me", handlers.GetProfile)

            protected.DELETE("/profile", handlers.DeleteAccount)
            protected.POST("/profile/leave", handlers.LeaveOrganization)
            protected.POST("/profile/upload-avatar", handlers.UploadAvatar)
                                    protected.DELETE("/profile/remove-avatar",
handlers.RemoveUserAvatar)

            protected.GET("/users/:id", handlers.GetUserByID)
            protected.PUT("/users/:id", handlers.UpdateUserByID)
            protected.DELETE("/users/:id/avatar", handlers.RemoveUserAvatar)
            protected.PUT("/users/:id/avatar", handlers.UploadUserAvatar)

            protected.POST("/organizations", handlers.CreateOrganization)
            protected.GET("/organizations/my", handlers.GetMyOrganization)
            protected.GET("/organizations/:id", handlers.GetOrganizationByID)
```

```
            protected.PUT("/organizations/:id", handlers.UpdateOrganization)
                                protected.POST("/organizations/:id/upload-avatar",
handlers.UploadOrganizationAvatar)
                                    protected.DELETE("/organizations/:id/avatar",
handlers.RemoveOrganizationAvatar)

            protected.POST("/teams", handlers.CreateTeam)
            protected.GET("/teams/:id", handlers.GetTeamByID)
            protected.PUT("/teams/:id", handlers.UpdateTeam)
                                        protected.POST("/teams/:id/upload-avatar",
handlers.UploadTeamAvatar)
            protected.DELETE("/teams/:id/avatar", handlers.RemoveTeamAvatar)
            protected.POST("/teams/:id/members", handlers.AddTeamMember)
                                    protected.DELETE("/teams/:id/members/:userId",
handlers.RemoveTeamMember)
            protected.DELETE("/teams/:id", handlers.DeleteTeam)
            protected.PUT("/teams/:id/leader", handlers.UpdateTeamLeader)

                                    protected.GET("/organizations/:id/free-users",
handlers.GetFreeUsersInOrganization)

            protected.POST("/invites", handlers.CreateInvite)
            protected.GET("/invites", handlers.GetInvitesForOrganization)
            protected.DELETE("/invites/:token", handlers.DeleteInvite)

            protected.GET("/potential-leaders", handlers.GetPotentialLeaders)
            protected.POST("/join/:token", handlers.JoinByInvite)

            protected.GET("/news", handlers.GetNewsFeed)
            protected.POST("/news", handlers.CreateNews)
            protected.GET("/tags", handlers.GetTags)

            protected.GET("/documents", handlers.GetDocuments)
            protected.POST("/documents", handlers.UploadDocument)

            protected.PUT("/users/:id/role", handlers.UpdateUserRole)
        }
    }

    port := os.Getenv("PORT")
    if port == "" {
        port = "8080"
    }
    log.Printf("Server starting on port %s...", port)
    r.Run(":" + port)
}

package database

import (
    "log"
    "os"

    "corp-portal/internal/models"

    "github.com/glebarez/sqlite"
    "gorm.io/gorm"
)

var DB *gorm.DB

func Connect() {
    dbName := os.Getenv("DB_NAME")
```

```
    if dbName == "" {
        dbName = "portal.db"
    }

    db, err := gorm.Open(sqlite.Open(dbName), &gorm.Config{})
    if err != nil {
        log.Fatal("Failed to connect to database: ", err)
    }

    log.Println("Connected to SQLite (Pure Go) successfully")

    log.Println("Running Migrations...")
    err = db.AutoMigrate(
        &models.User{},
        &models.Organization{},
        &models.Team{},
        &models.Invite{},
        &models.News{},
        &models.Document{},
    )
    if err != nil {
        log.Fatal("Migration failed: ", err)
    }

    DB = db
}
package utils

import (
    "corp-portal/internal/models"
    "os"
    "time"

    "github.com/golang-jwt/jwt/v5"
)

type Claims struct {
    UserID uint        `json:"user_id"`
    Role   models.Role `json:"role"`
    jwt.RegisteredClaims
}

func GenerateToken(user models.User) (string, error) {
    jwtSecret := []byte(os.Getenv("JWT_SECRET"))

    claims := Claims{
        UserID: user.ID,
        Role:   user.Role,
        RegisteredClaims: jwt.RegisteredClaims{
            ExpiresAt: jwt.NewNumericDate(time.Now().Add(24 * time.Hour)),
            IssuedAt:  jwt.NewNumericDate(time.Now()),
        },
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString(jwtSecret)
}

func ParseToken(tokenString string) (*Claims, error) {
    jwtSecret := []byte(os.Getenv("JWT_SECRET"))

      token, err := jwt.ParseWithClaims(tokenString, &Claims{}, func(token
*jwt.Token) (interface{}, error) {
```

```
        return jwtSecret, nil
    })

    if err != nil {
        return nil, err
    }

    if claims, ok := token.Claims.(*Claims); ok && token.Valid {
        return claims, nil
    }

    return nil, err
}
package handlers

import (
    "corp-portal/internal/database"
    "corp-portal/internal/models"
    "fmt"
    "net/http"
    "path/filepath"
    "strings"
    "time"

    "github.com/gin-gonic/gin"
    "gorm.io/gorm"
)

type CreateOrgInput struct {
    Name        string `json:"name" binding:"required"`
    Description string `json:"description"`
}

type CreateTeamInput struct {
    Name        string `json:"name" binding:"required"`
    Description string `json:"description"`
    LeaderID    uint   `json:"leader_id"`
}

func CreateOrganization(c *gin.Context) {
    userID := c.MustGet("userID").(uint)
    var input CreateOrgInput
    if err := c.ShouldBindJSON(&input); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    tx := database.DB.Begin()

    org := models.Organization{
        Name:        input.Name,
        Description: input.Description,
        OwnerID:     userID,
    }
    if err := tx.Create(&org).Error; err != nil {
        tx.Rollback()
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Could not create
organization"})
        return
    }
                        result          :=          tx.Model(&models.User{ID:
userID}).Updates(map[string]interface{}{
        "organization_id": org.ID,
```

```
        "role":            int(models.RoleSuperAdmin),
    })

    if result.Error != nil {
        tx.Rollback()
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Could not update
user role"})
        return
    }
    if result.RowsAffected == 0 {
        tx.Rollback()
        c.JSON(http.StatusNotFound, gin.H{"error": "User not found"})
        return
    }

    tx.Commit()
    c.JSON(http.StatusCreated, org)
}

func CreateTeam(c *gin.Context) {
    userID := c.MustGet("userID").(uint)
    userRole := c.MustGet("role").(models.Role)

    if userRole < models.RoleAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Not enough permissions"})
        return
    }

    var input CreateTeamInput
    if err := c.ShouldBindJSON(&input); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    var user models.User
    database.DB.Select("organization_id").First(&user, userID)

    if user.OrganizationID == nil {
          c.JSON(http.StatusBadRequest, gin.H{"error": "You are not in an
organization"})
        return
    }

    team := models.Team{
        Name:           input.Name,
        Description:    input.Description,
        OrganizationID: *user.OrganizationID,
    }

    if input.LeaderID != 0 {
        var leader models.User
        if err := database.DB.First(&leader, input.LeaderID).Error; err != nil
{
            c.JSON(http.StatusBadRequest, gin.H{"error": "Leader not found"})
            return
        }
            if leader.OrganizationID == nil || *leader.OrganizationID !=
*user.OrganizationID {
             c.JSON(http.StatusBadRequest, gin.H{"error": "Leader must be from
the same organization"})
            return
        }
```

```
        team.LeaderID = &input.LeaderID

        if leader.Role < models.RoleManager {
            database.DB.Model(&leader).Update("role", models.RoleManager)
        }
    }

    if err := database.DB.Create(&team).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to create
team"})
        return
    }

    c.JSON(http.StatusCreated, team)
}

func GetMyOrganization(c *gin.Context) {
    userID := c.MustGet("userID").(uint)

    var user models.User
    database.DB.First(&user, userID)

    if user.OrganizationID == nil {
        c.JSON(http.StatusNotFound, gin.H{"message": "No organization"})
        return
    }

    var org models.Organization
       if  err  :=  database.DB.Preload("Teams").Preload("Users").First(&org,
*user.OrganizationID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Organization not found"})
        return
    }

    c.JSON(http.StatusOK, org)
}

func GetTeamByID(c *gin.Context) {
    requestorID := c.MustGet("userID").(uint)
    teamID := c.Param("id")

    var user models.User
    database.DB.First(&user, requestorID)

    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }
   if user.OrganizationID == nil || team.OrganizationID != *user.OrganizationID
{
        c.JSON(http.StatusForbidden, gin.H{"error": "Access denied"})
        return
    }
    response := buildTeamProfileResponse(&team)
    c.JSON(http.StatusOK, response)
}

func buildTeamProfileResponse(team *models.Team) models.TeamProfileResponse {
    response := models.TeamProfileResponse{
        ID:             team.ID,
        Name:           team.Name,
        Description:    team.Description,
```

```
        AvatarURL:      team.AvatarURL,
        OrganizationID: team.OrganizationID,
        LeaderID:       team.LeaderID,
        CreatedAt:      team.CreatedAt,
    }
    var org models.Organization
    if err := database.DB.
            Select("id",  "name",  "description",  "avatar_url",  "owner_id",
"created_at").
        First(&org, team.OrganizationID).Error; err == nil {
        response.Organization = &models.OrganizationResponse{
            ID:          org.ID,
            Name:        org.Name,
            Description: org.Description,
            AvatarURL:   org.AvatarURL,
            OwnerID:     org.OwnerID,
            CreatedAt:   org.CreatedAt,
        }
    }
    if team.LeaderID != nil {
        var leader models.User
        if err := database.DB.
            Select("id", "full_name", "email", "avatar_url", "role").
            First(&leader, *team.LeaderID).Error; err == nil {
            response.Leader = &models.UserSimpleResponse{
                ID:        leader.ID,
                FullName:  leader.FullName,
                Email:     leader.Email,
                AvatarURL: leader.AvatarURL,
                Role:      leader.Role,
            }
        }
    }
    var members []models.User
    if err := database.DB.
        Select("id", "full_name", "email", "avatar_url", "role", "team_id").
        Where("team_id = ?", team.ID).
        Find(&members).Error; err == nil {
        response.Members = make([]models.UserSimpleResponse, len(members))
        for i, member := range members {
            response.Members[i] = models.UserSimpleResponse{
                ID:        member.ID,
                FullName:  member.FullName,
                Email:     member.Email,
                AvatarURL: member.AvatarURL,
                Role:      member.Role,
            }
        }
    }

    return response
}

func UpdateTeam(c *gin.Context) {
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)
    teamID := c.Param("id")

    var input struct {
        Name        string `json:"name"`
        Description string `json:"description"`
    }
    if err := c.ShouldBindJSON(&input); err != nil {
```

```
        return
    }

    var team models.Team
    database.DB.First(&team, teamID)
    isLeader := team.LeaderID != nil && *team.LeaderID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isLeader && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Only Team Leader or Admin
can edit team"})
        return
    }

    database.DB.Model(&team).Updates(models.Team{
        Name:        input.Name,
        Description: input.Description,
    })
    c.JSON(http.StatusOK, team)
}

func UpdateOrganization(c *gin.Context) {
    requestorRole := c.MustGet("role").(models.Role)
    orgID := c.Param("id")

    if requestorRole < models.RoleAdmin {
          c.JSON(http.StatusForbidden, gin.H{"error": "Only Admins can edit
organization"})
        return
    }

    var input struct {
        Name        string `json:"name"`
        Description string `json:"description"`
    }
    c.ShouldBindJSON(&input)

    var org models.Organization
    database.DB.First(&org, orgID)

    database.DB.Model(&org).Updates(models.Organization{
        Name:        input.Name,
        Description: input.Description,
    })
    c.JSON(http.StatusOK, org)
}
func GetOrganizationByID(c *gin.Context) {
    requestorID := c.MustGet("userID").(uint)
    targetOrgID := c.Param("id")

    var user models.User
    database.DB.First(&user, requestorID)

    var org models.Organization
    if err := database.DB.First(&org, targetOrgID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Org not found"})
        return
    }
    if user.OrganizationID == nil || *user.OrganizationID != org.ID {
        c.JSON(http.StatusForbidden, gin.H{"error": "Access denied"})
        return
    }
    response := buildOrganizationProfileResponse(&org)
```

```
    c.JSON(http.StatusOK, response)
}

func buildOrganizationProfileResponse(org *models.Organization)
models.OrganizationProfileResponse {
    response := models.OrganizationProfileResponse{
        ID:          org.ID,
        Name:        org.Name,
        Description: org.Description,
        AvatarURL:   org.AvatarURL,
        OwnerID:     org.OwnerID,
        CreatedAt:   org.CreatedAt,
    }
    var teams []models.Team
    if err := database.DB.
         Select("id", "name", "description", "avatar_url", "organization_id",
"leader_id", "created_at").
        Where("organization_id = ?", org.ID).
        Find(&teams).Error; err == nil {

        response.Teams = make([]models.TeamResponse, len(teams))
        for i, team := range teams {
            teamResponse := models.TeamResponse{
                ID:             team.ID,
                Name:           team.Name,
                Description:    team.Description,
                AvatarURL:      team.AvatarURL,
                OrganizationID: team.OrganizationID,
                LeaderID:       team.LeaderID,
                CreatedAt:      team.CreatedAt,
            }
            if team.LeaderID != nil {
                var leader models.User
                if err := database.DB.
                    Select("id", "full_name", "email", "avatar_url", "role").
                    First(&leader, *team.LeaderID).Error; err == nil {
                    teamResponse.Leader = &models.UserSimpleResponse{
                        ID:        leader.ID,
                        FullName:  leader.FullName,
                        Email:     leader.Email,
                        AvatarURL: leader.AvatarURL,
                        Role:      leader.Role,
                    }
                }
            }
            response.Teams[i] = teamResponse
        }
    }
    var users []models.User
    if err := database.DB.
         Select("id", "full_name", "email", "avatar_url", "role", "team_id",
"created_at").
        Where("organization_id = ?", org.ID).
        Find(&users).Error; err == nil {

        response.Users = make([]models.UserSimpleResponse, len(users))
        for i, user := range users {
            response.Users[i] = models.UserSimpleResponse{
                ID:        user.ID,
                FullName:  user.FullName,
                Email:     user.Email,
                AvatarURL: user.AvatarURL,
```

```
                Role:      user.Role,
            }
        }
    }

    return response
}

func UploadTeamAvatar(c *gin.Context) {
    teamID := c.Param("id")
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }

    isLeader := team.LeaderID != nil && *team.LeaderID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isLeader && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }
    file, err := c.FormFile("avatar")
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "No file uploaded"})
        return
    }
    ext := strings.ToLower(filepath.Ext(file.Filename))
    allowedExts := map[string]bool{".jpg": true, ".jpeg": true, ".png": true,
".gif": true, ".webp": true}
    if !allowedExts[ext] {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid file type"})
        return
    }
    filename := fmt.Sprintf("team_%s_%d%s", teamID, time.Now().Unix(), ext)
    uploadPath := filepath.Join("uploads", "team_avatars", filename)

    if err := c.SaveUploadedFile(file, uploadPath); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to save
file"})
        return
    }
    avatarURL := fmt.Sprintf("/uploads/team_avatars/%s", filename)

    if err := database.DB.Model(&team).Update("avatar_url", avatarURL).Error;
err != nil {
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update
database"})
        return
    }

    c.JSON(http.StatusOK, gin.H{
        "message":    "Avatar uploaded successfully",
        "avatar_url": avatarURL,
    })
}

func RemoveTeamAvatar(c *gin.Context) {
    teamID := c.Param("id")
```

```
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }

    isLeader := team.LeaderID != nil && *team.LeaderID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isLeader && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }
    if err := database.DB.Model(&team).Update("avatar_url", "").Error; err !=
nil {
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to remove
avatar"})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "Avatar removed"})
}

func UploadOrganizationAvatar(c *gin.Context) {
    orgID := c.Param("id")
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var org models.Organization
    if err := database.DB.First(&org, orgID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Organization not found"})
        return
    }

    isOwner := org.OwnerID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isOwner && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }

    file, err := c.FormFile("avatar")
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "No file uploaded"})
        return
    }

    ext := strings.ToLower(filepath.Ext(file.Filename))
    allowedExts := map[string]bool{".jpg": true, ".jpeg": true, ".png": true,
".gif": true, ".webp": true}
    if !allowedExts[ext] {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid file type"})
        return
    }

    filename := fmt.Sprintf("org_%s_%d%s", orgID, time.Now().Unix(), ext)
    uploadPath := filepath.Join("uploads", "org_avatars", filename)

    if err := c.SaveUploadedFile(file, uploadPath); err != nil {
```

```
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to save
file"})
        return
    }
    avatarURL := fmt.Sprintf("/uploads/org_avatars/%s", filename)
    if err := database.DB.Model(&org).Update("avatar_url", avatarURL).Error;
err != nil {
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update
database"})
        return
    }

    c.JSON(http.StatusOK, gin.H{
        "message":    "Avatar uploaded successfully",
        "avatar_url": avatarURL,
    })
}

func RemoveOrganizationAvatar(c *gin.Context) {
    orgID := c.Param("id")
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var org models.Organization
    if err := database.DB.First(&org, orgID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Organization not found"})
        return
    }

    isOwner := org.OwnerID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isOwner && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }

    if err := database.DB.Model(&org).Update("avatar_url", "").Error; err !=
nil {
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to remove
avatar"})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "Avatar removed"})
}

func GetFreeUsersInOrganization(c *gin.Context) {
    orgID := c.Param("id")

    var freeUsers []models.User
    if err := database.DB.
        Select("id", "full_name", "email", "avatar_url", "role").
          Where("organization_id = ? AND (team_id IS NULL OR team_id = 0)",
orgID).
        Find(&freeUsers).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to fetch
users"})
        return
    }

    response := make([]models.UserSimpleResponse, len(freeUsers))
    for i, u := range freeUsers {
```

```
        response[i] = models.UserSimpleResponse{
            ID:        u.ID,
            FullName:  u.FullName,
            Email:     u.Email,
            AvatarURL: u.AvatarURL,
            Role:      u.Role,
        }
    }

    c.JSON(http.StatusOK, response)
}

func AddTeamMember(c *gin.Context) {
    teamID := c.Param("id")
    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var input struct {
        UserID uint `json:"user_id" binding:"required"`
    }
    if err := c.ShouldBindJSON(&input); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }

    isLeader := team.LeaderID != nil && *team.LeaderID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isLeader && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Only Leader or Admin can
add members"})
        return
    }

    var targetUser models.User
    if err := database.DB.First(&targetUser, input.UserID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "User not found"})
        return
    }

     if targetUser.OrganizationID == nil || *targetUser.OrganizationID !=
team.OrganizationID {
        c.JSON(http.StatusBadRequest, gin.H{"error": "User belongs to another
organization"})
        return
    }
    if targetUser.TeamID != nil && *targetUser.TeamID != 0 {
         c.JSON(http.StatusBadRequest, gin.H{"error": "User is already in a
team"})
        return
    }

    if err := database.DB.Model(&targetUser).Update("team_id", team.ID).Error;
err != nil {
         c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to add
user"})
        return
```

```
    }

    c.JSON(http.StatusOK, gin.H{"message": "User added to team"})
}

func RemoveTeamMember(c *gin.Context) {
    teamID := c.Param("id")
    targetUserID := c.Param("userId")

    requestorID := c.MustGet("userID").(uint)
    requestorRole := c.MustGet("role").(models.Role)

    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }

    isLeader := team.LeaderID != nil && *team.LeaderID == requestorID
    isAdmin := requestorRole >= models.RoleAdmin

    if !isLeader && !isAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }

    var targetUser models.User
    if err := database.DB.First(&targetUser, targetUserID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "User not found"})
        return
    }

    if targetUser.TeamID == nil || *targetUser.TeamID != team.ID {
          c.JSON(http.StatusBadRequest, gin.H{"error": "User is not in this
team"})
        return
    }

    if team.LeaderID != nil && *team.LeaderID == targetUser.ID {
       c.JSON(http.StatusBadRequest, gin.H{"error": "Cannot remove team leader.
Change leader first."})
        return
    }

         if err := database.DB.Model(&targetUser).Update("team_id",
gorm.Expr("NULL")).Error; err != nil {
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to remove
user"})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "User removed from team"})
}

func DeleteTeam(c *gin.Context) {
    teamID := c.Param("id")
    requestorRole := c.MustGet("role").(models.Role)

    if requestorRole < models.RoleAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }
```

```
    var team models.Team
    if err := database.DB.First(&team, teamID).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
        return
    }

    tx := database.DB.Begin()
        if err := tx.Model(&models.User{}).Where("team_id = ?",
team.ID).Update("team_id", gorm.Expr("NULL")).Error; err != nil {
        tx.Rollback()
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update
members"})
        return
    }
    if team.LeaderID != nil {
        var leader models.User
        if err := tx.First(&leader, *team.LeaderID).Error; err == nil {
            if leader.Role == models.RoleManager {
                tx.Model(&leader).Update("role", models.RoleUser)
            }
        }
    }
    if err := tx.Delete(&team).Error; err != nil {
        tx.Rollback()
       c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete
team"})
        return
    }

    tx.Commit()
    c.JSON(http.StatusOK, gin.H{"message": "Team deleted"})
}

type UpdateLeaderInput struct {
    LeaderID *uint `json:"leader_id"`
}

func UpdateTeamLeader(c *gin.Context) {
    teamID := c.Param("id")
    requestorRole := c.MustGet("role").(models.Role)

    if requestorRole < models.RoleAdmin {
        c.JSON(http.StatusForbidden, gin.H{"error": "Permission denied"})
        return
    }

    var input UpdateLeaderInput
    if err := c.ShouldBindJSON(&input); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    tx := database.DB.Begin()
    defer func() {
        if r := recover(); r != nil {
            tx.Rollback()
        }
    }()

    var team models.Team
    if err := tx.First(&team, teamID).Error; err != nil {
        tx.Rollback()
        c.JSON(http.StatusNotFound, gin.H{"error": "Team not found"})
```

```
        return
    }

    if team.LeaderID != nil {
        var oldLeader models.User
        if err := tx.First(&oldLeader, *team.LeaderID).Error; err == nil {
            if oldLeader.Role == models.RoleManager {
                var otherTeamsCount int64
                if err := tx.Model(&models.Team{}).
                    Where("leader_id = ? AND id != ?", oldLeader.ID, team.ID).
                   Count(&otherTeamsCount).Error; err == nil && otherTeamsCount
== 0 {
                            if  err  :=  tx.Model(&oldLeader).Update("role",
models.RoleUser).Error; err != nil {
                        tx.Rollback()
                         c.JSON(http.StatusInternalServerError, gin.H{"error":
"Failed to demote old leader"})
                        return
                    }
                }
            }
        }
    }
    if input.LeaderID != nil && *input.LeaderID != 0 {
        var newLeader models.User
        if err := tx.First(&newLeader, *input.LeaderID).Error; err != nil {
            tx.Rollback()
           c.JSON(http.StatusNotFound, gin.H{"error": "New leader not found"})
            return
        }

         if newLeader.OrganizationID == nil || *newLeader.OrganizationID !=
team.OrganizationID {
            tx.Rollback()
               c.JSON(http.StatusBadRequest, gin.H{"error": "User belongs to
another organization"})
            return
        }
        if newLeader.TeamID == nil || *newLeader.TeamID != team.ID {
             if err := tx.Model(&newLeader).Update("team_id", team.ID).Error;
err != nil {
                tx.Rollback()
                 c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed
to add leader to team"})
                return
            }
        }
        if newLeader.Role < models.RoleManager {
                        if   err   :=   tx.Model(&newLeader).Update("role",
models.RoleManager).Error; err != nil {
                tx.Rollback()
                 c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed
to promote new leader"})
                return
            }
        }

        team.LeaderID = input.LeaderID
    } else {
        team.LeaderID = nil
    }

    if err := tx.Save(&team).Error; err != nil {
```

```
        tx.Rollback()
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update
team"})
        return
    }

    if err := tx.Commit().Error; err != nil {
        tx.Rollback()
           c.JSON(http.StatusInternalServerError,  gin.H{"error":  "Transaction
failed"})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "Leader updated successfully"})
}
package handlers
```

ВЕДОМОСТЬ ДОКУМЕНТОВ

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1–40 01 01 217 ПЗ	Пояснительная записка	57 с.
	Графические документы	
ГУИР. 251002 017 БД	Схема алгоритма	Формат А1

					БГУИР КП І- 40 01 01 217 ПЗ
И	Ли	№ докум.	Под	Дата	
Разраб.		Маталыга Е.А.			
Провер.		Петровская В.В.			

Литер	Лист		
Т	58	58	
Внутренний портал компании Ведомость курсового проекта	Кафедра ПОИТ гр. 251002		