МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

МАИ

Институт №3

Системы управления, информатика и электроэнергетика

Кафедра 304

Вычислительные машины, системы, сети

Отчет курсовой работе

По учебной дисциплине: «web-программирование»

по теме «Создание веб-платформы для проведения онлайн-опросов»

Подготовил студент группы М3О-119БВ-24:

Тихонов П.С.

Приняла:

Иванова П.М.

Москва 2025

# Задание

Используя концепцию CRUD (Create, Read, Update, Delete), разработать single-page application, представляющее собой список / витрину уникальных сущностей.

В приложении должен быть реализован функционал:

* добавления и удаления сущностей,
* просмотра сущностей,
* редактирования сущностей.

Под «сущностями» могут пониматься, например, видеоигровые персонажи, преподаватели, музеи, книги, памятные фотографии и так далее.

По желанию:

реализовать систему «ролей». Основные роли:

* пользователь (доступно только чтение)
* администратор (создание, редактирование, удаление).

Для ограничения доступа к функционалу администратора необходимо реализовать систему авторизации и аутентификации.

предусмотреть механизм валидации входящих данных на сторонах сервера и клиента.

При реализации валидации заполнения форм на клиенте предусмотреть выделение полей ввода с ошибкой, а также автоматическую прокрутку страницы к первому полю с ошибкой. (для реализации валидации можно использовать Redux-toolkit, Final-form и т.д.)

Поиск сущностей, а также их сортировка по разным критериям (поиск и сортировка должны быть оптимальными по времени)

В процессе разработки необходимо использовать стек технологий PERN:

* PostgreSQL,
* Express,
* React,
* Node.js.

**Оглавление**

[ЗАДАНИЕ 2](#_Toc198147857)

[ВВЕДЕНИЕ 5](#_Toc198147858)

[ГЛАВА 1. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ 7](#_Toc198147859)

[ГЛАВА 2. РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ 11](#_Toc198147860)

[2.1 АРХИТЕКТУРА КЛИЕНТА 11](#_Toc198147861)

[2.1.1 ОПИСАНИЕ ФАЙЛОВОЙ СТРУКТУРЫ 11](#_Toc198147862)

[2.1.2 ОПИСАНИЕ ФУНКЦИОНАЛА СТРАНИЦ ПРИЛОЖЕНИЯ 13](#_Toc198147863)

[2.1.3 ОПИСАНИЕ СВЯЗЕЙ МЕЖДУ СТРАНИЦАМИ 20](#_Toc198147864)

[2.2 АРХИТЕКТУРА СЕРВЕР 23](#_Toc198147865)

[2.2.1 ОПИСАНИЕ ФАЙЛОВОЙ СТРУКТУРЫ 23](#_Toc198147866)

[2.2.2 ОПИСАНИЕ API 26](#_Toc198147867)

[2.3 ОПИСАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ 30](#_Toc198147868)

[ГЛАВА 3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 38](#_Toc198147869)

[3.1 ОПИСАНИЕ ПРОЦЕССА РЕАЛИЗАЦИИ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ 38](#_Toc198147870)

[3.1.1 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ И МИГРАЦИИ 38](#_Toc198147871)

[3.1.2 РЕАЛИЗАЦИЯ GRAPHQL API 39](#_Toc198147872)

[3.1.3 РЕАЛИЗАЦИЯ СИСТЕМЫ АУТЕНТИФИКАЦИИ И АВТОРИЗАЦИИ 41](#_Toc198147873)

[3.1.4 ИНВЕРСИЯ ЗАВИСИМОСТЕЙ 43](#_Toc198147874)

[3.1.5 ОРГАНИЗАЦИЯ HTTP-СЕРВЕРА 44](#_Toc198147875)

[3.2 ОПИСАНИЕ ПРОЦЕССА РЕАЛИЗАЦИИ КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ 45](#_Toc198147876)

[3.2.1 РЕАЛИЗАЦИЯ СИСТЕМЫ АУТЕНТИФИКАЦИИ И АВТОРИЗАЦИИ 45](#_Toc198147877)

[3.2.2 НАСТРОЙКА APOLLO CLIENT ДЛЯ ВЗАИМОДЕЙСТВИЯ С GRAPHQL API 46](#_Toc198147878)

[3.2.3 СИСТЕМА АУТЕНТИФИКАЦИИ И АВТОРИЗАЦИИ 47](#_Toc198147879)

[3.2.4 РАЗРАБОТКА СИСТЕМЫ СОЗДАНИЯ И УПРАВЛЕНИЯ ФОРМАМИ 50](#_Toc198147880)

[3.2.5 СИСТЕМА ПРОСМОТРА И ЗАПОЛНЕНИЯ ФОРМ 51](#_Toc198147881)

[ГЛАВА 4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ 56](#_Toc198147882)

[4.1. ПЛАН ТЕСТИРОВАНИЯ 56](#_Toc198147883)

[4.2. ПРОЦЕСС И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 58](#_Toc198147884)

[ЗАКЛЮЧЕНИЕ 65](#_Toc198147885)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 68](#_Toc198147886)

[ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ 70](#_Toc198147887)

# Введение

Актуальность темы исследования. В современных условиях информационного общества процессы сбора и анализа данных приобретают первостепенное значение для различных организаций и частных лиц. Методы анкетирования, опросов и форм обратной связи стали неотъемлемым инструментом исследований, маркетинга, образовательной деятельности и многих других сфер. Разработка специализированного веб-приложения, реализующего концепцию CRUD (Create, Read, Update, Delete), позволяет существенно оптимизировать и автоматизировать данные процессы, обеспечивая их доступность пользователям с различным уровнем технической подготовки.

Особую научно-практическую значимость имеет создание single-page application, обеспечивающего высокую производительность, отзывчивость пользовательского интерфейса и снижение нагрузки на серверную часть. Внедрение системы с дифференциацией прав доступа (роли пользователя и администратора) способствует обеспечению необходимого уровня информационной безопасности и контроля над данными, что приобретает критическое значение в условиях увеличения количества киберугроз и ужесточения нормативных требований к защите персональных данных.

**Целью курсовой работы** заключается в разработке функционального и эргономичного веб-приложения для создания, управления и анализа форм с различными типами вопросов, реализующего полный цикл операций CRUD.

Для достижения поставленной цели определены **следующие задачи**:

1. Провести анализ существующих аналогов и выявить их функциональные возможности, преимущества и недостатки.
2. Спроектировать архитектуру веб-приложения, обеспечивающую гибкость, масштабируемость и высокую производительность.
3. Разработать клиентскую часть приложения с использованием современных технологий для создания интуитивно понятного и отзывчивого пользовательского интерфейса.
4. Реализовать серверную часть приложения, обеспечивающую эффективное взаимодействие с базой данных и обработку запросов клиента.
5. Внедрить систему авторизации и аутентификации с разграничением прав доступа между пользователями и администраторами.
6. Разработать и интегрировать механизмы валидации данных как на стороне клиента, так и на стороне сервера.
7. Реализовать функционал поиска и сортировки данных с оптимизацией по времени выполнения.
8. Провести тестирование разработанного приложения на соответствие функциональным требованиям и удобство использования.

**Объектом исследования** являются системы управления формами и опросами. В работе анализируются:

— принципы построения веб-приложений с использованием современных технологий;

— архитектурные решения для single-page applications (SPA), направленные на оптимизацию пользовательского опыта;

— критерии обеспечения производительности, безопасности и удобства использования.

**Предметом исследования** выступают методы и технологии разработки веб-приложений с реализацией функционала CRUD (создание, чтение, обновление, удаление) для форм, включая:

— проектирование интерактивного интерфейса на базе React;

— организацию серверной логики и взаимодействия с базами данных;

— механизмы валидации данных;

# Глава 1. Описание используемых технологий

Разработка современного веб-приложения для управления формами и опросами требует комплексного подхода к выбору технологий, обеспечивающих оптимальную производительность, масштабируемость и удобство использования. В рамках данного проекта был сформирован технологический стек, отвечающий современным тенденциям в области веб-разработки и позволяющий эффективно реализовать требуемый функционал.

Архитектура разрабатываемого приложения построена на основе клиент-серверного взаимодействия с чётким разделением функциональности между Frontend и Backend компонентами. Такой подход обеспечивает модульность, гибкость и масштабируемость системы, а также упрощает процесс разработки и дальнейшей поддержки приложения.

Клиентская часть (Frontend) разработана на базе фреймворка Next.js, представляющего собой полноценную экосистему для создания современных React-приложений. В отличие от использования "чистого" React, Next.js предоставляет готовые решения для серверного рендеринга (SSR), статической генерации страниц (SSG), оптимизации изображений, автоматической маршрутизации на основе файловой системы и встроенной поддержки API-маршрутов. Выбор Next.js в качестве основного фреймворка обусловлен его преимуществами в области производительности, SEO-оптимизации и удобства разработки по сравнению с традиционными SPA на React.

React, являясь основой Next.js, обеспечивает компонентный подход к разработке, эффективную работу с виртуальным DOM и однонаправленный поток данных. Использование React в составе Next.js позволяет сохранить все преимущества этой библиотеки, дополнив их возможностями фреймворка для создания более производительных и оптимизированных приложений.

Типобезопасность кода и улучшенная поддержка в интегрированных средах разработки достигаются за счёт использования TypeScript. Данный язык программирования, являясь надмножеством JavaScript, обеспечивает раннее обнаружение ошибок на этапе компиляции и позволяет создавать более надёжные и поддерживаемые программные решения.

Для управления формами в клиентской части применяется библиотека React Hook Form, отличающаяся высокой производительностью и простотой интеграции. Валидация пользовательского ввода реализована с помощью библиотеки Zod, которая обеспечивает типобезопасную валидацию данных и автоматическое выведение типов TypeScript из схем валидации.

Визуальное оформление приложения построено на основе утилитарного CSS-фреймворка Tailwind CSS, позволяющего быстро создавать адаптивные и консистентные интерфейсы. Доступность и интерактивность элементов управления обеспечивается компонентами Radix UI, предоставляющими семантическую разметку и поддержку клавиатурной навигации.

Серверная часть (Backend) разработана с использованием языка программирования Golang, который отличается высокой производительностью, эффективным управлением памятью и встроенной поддержкой конкурентности. Основой серверной архитектуры служит фреймворк Gin, обеспечивающий быструю обработку HTTP-запросов, маршрутизацию и промежуточное программное обеспечение (middleware) для аутентификации и авторизации пользователей.

Для организации модульной архитектуры и управления зависимостями на серверной стороне применяется UberFX – фреймворк для построения масштабируемых сервисов, разработанный компанией Uber. Он предоставляет инструменты для внедрения зависимостей, конфигурирования и мониторинга приложения, что значительно упрощает разработку и тестирование отдельных компонентов системы.

Взаимодействие между клиентской и серверной частями реализовано через GraphQL API, что позволяет оптимизировать сетевые запросы и получать только необходимые данные. GraphQL обеспечивает гибкость при запросе данных, позволяя клиенту точно указать, какие поля требуется получить, что особенно важно при работе с комплексными структурами данных, такими как формы с различными типами вопросов и ответов.

Для хранения данных используется реляционная база данных PostgreSQL, обладающая высокой надёжностью, соответствием стандартам SQL и поддержкой сложных типов данных, включая JSON. Взаимодействие с базой данных осуществляется через ORM-библиотеки, обеспечивающие абстракцию над SQL-запросами и упрощающие операции с данными.

Аутентификация и авторизация пользователей реализованы с использованием сессионного подхода на основе HTTP-куков. В отличие от JWT-токенов, сессионная аутентификация обеспечивает более высокий уровень безопасности за счёт хранения чувствительной информации на сервере, а не в браузере пользователя. Для удобства пользователей внедрена возможность авторизации через Google OAuth 2.0, что позволяет использовать существующие учётные записи Google для входа в систему без необходимости создания и запоминания дополнительных учётных данных. Контроль доступа основан на ролевой модели, разделяющей пользователей на обычных пользователей с ограниченными правами и администраторов с полным доступом к функциональности системы.

В процессе разработки применяются современные инструменты контроля качества кода: статический анализатор golint для серверного кода на Golang, ESLint для клиентского JavaScript/TypeScript кода.

Выбранный технологический стек обеспечивает оптимальное сочетание производительности, масштабируемости и удобства разработки, позволяя создать современное веб-приложение для управления формами и опросами, полностью соответствующее концепции CRUD и удовлетворяющее требованиям к функциональности и пользовательскому опыту.

# Глава 2. Разработка архитектуры приложения

## 2.1 Архитектура клиента

### 2.1.1 Описание файловой структуры

В основу организации кодовой базы проекта положена методология Feature-Sliced Design (FSD), адаптированная под специфику фреймворка Next.js. FSD представляет собой подход к архитектуре frontend-приложений, основанный на принципах разделения ответственности и изоляции функциональных возможностей. Данная методология предполагает организацию кода в виде слоёв (layers) и срезов (slices), что обеспечивает высокий уровень модульности и упрощает навигацию по проекту.

Структура проекта учитывает особенности файловой системы Next.js, в частности, использование каталога /app для организации маршрутизации на основе App Router, введённого в Next.js 13. Благодаря такой интеграции удалось сохранить преимущества обоих подходов: декларативную маршрутизацию Next.js и модульность FSD.

Базовая структура проекта имеет следующий вид:

Изображение выглядит как текст, диаграмма, Шрифт, План

Контент, сгенерированный ИИ, может содержать ошибки.

Адаптация FSD под особенности Next.js потребовала компромиссных решений. В частности, вместо стандартного слоя pages в FSD используется директория /app, соответствующая концепции App Router в Next.js. Это позволяет сохранить преимущества файловой маршрутизации Next.js, такие как вложенные макеты, серверные компоненты и параллельные маршруты.

Слой **features** объединяет функциональные возможности приложения, такие как аутентификация, конструктор форм, отправка заполненных форм и аналитика. Каждая функция представляет собой законченный пользовательский сценарий и может включать в себя компоненты, хуки, модели и API-клиенты.

Слой **processes** представляет собой комплексные бизнес-процессы, объединяющие несколько функциональных возможностей в рамках сложных пользовательских сценариев. Например, процесс создания формы включает в себя выбор шаблона, настройку вопросов, определение логики ветвления и публикацию формы. Слой процессов служит для оркестрации взаимодействия между различными функциями и обеспечивает целостность пользовательского опыта. Данный слой является промежуточным между функциональными возможностями (features) и пользовательским интерфейсом (widgets/app), что соответствует вертикальной архитектуре методологии FSD.

В слое **shared** размещаются инфраструктурные компоненты и утилиты, не привязанные к конкретным бизнес-сущностям или функциям. Это позволяет избежать дублирования кода и обеспечивает единый стиль разработки.

Слой **widgets** содержит композиционные блоки интерфейса, которые объединяют несколько компонентов для решения конкретных задач пользовательского интерфейса, например, шапку приложения, боковую панель или редактор вопросов.

Интеграция методологии FSD с файловой структурой Next.js позволила создать масштабируемую и поддерживаемую архитектуру приложения, сохраняя при этом все преимущества выбранного фреймворка и обеспечивая эффективную организацию кодовой базы клиента.

### 2.1.2 Описание функционала страниц приложения

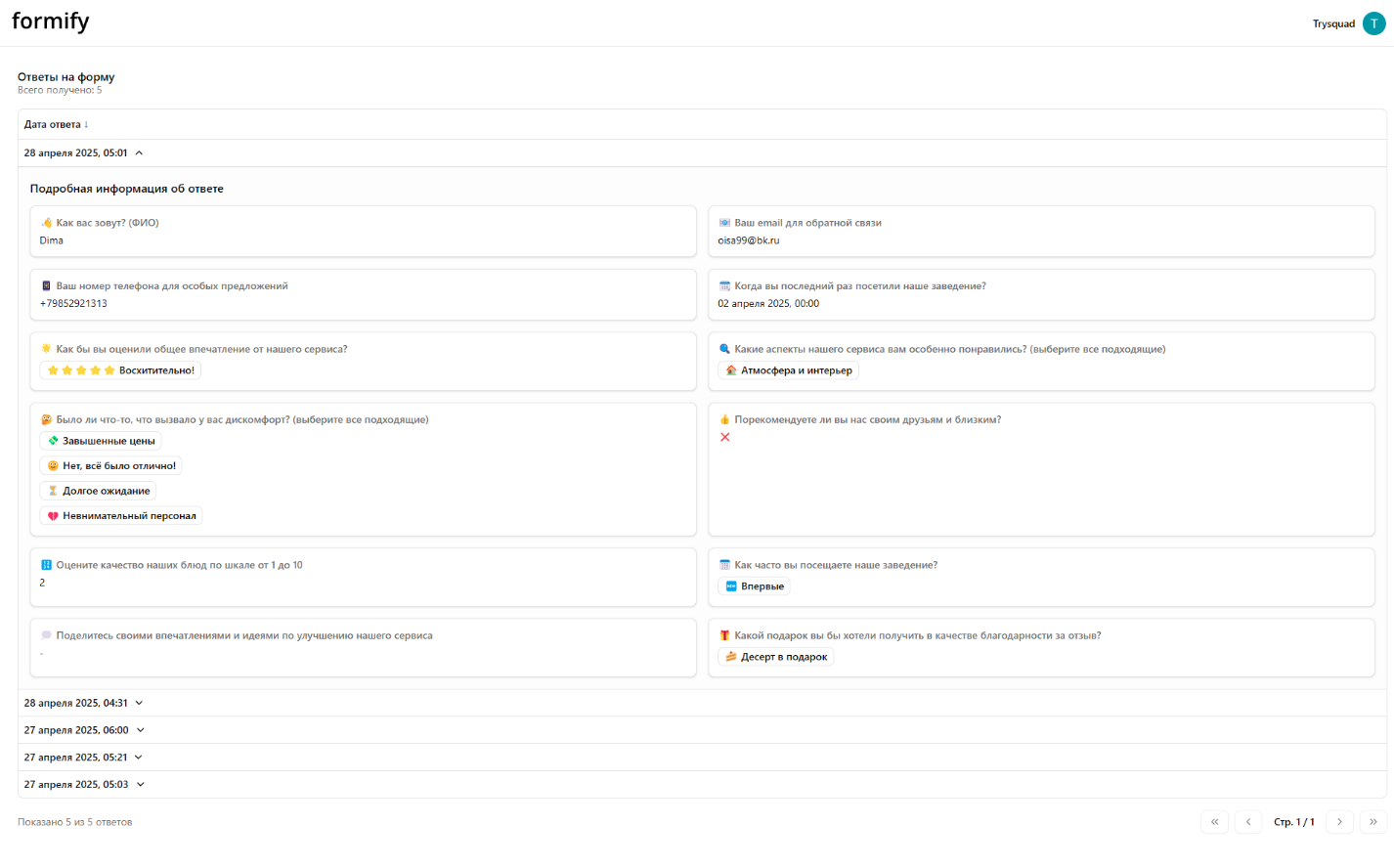
В рамках разработанного веб-приложения реализован набор страниц, обеспечивающих полный цикл работы с формами и ответами. Каждая страница выполняет специфические функции и вместе они образуют целостную систему для создания, редактирования, заполнения и анализа форм. Рассмотрим ключевые страницы приложения и их функциональные возможности.

**Страница просмотра ответов (answers/[slug])**

Страница просмотра ответов предназначена для анализа и управления полученными ответами на конкретную форму. Идентификатор формы передается в URL через параметр [slug], что обеспечивает доступ к данным соответствующей формы. Основные функциональные возможности данной страницы:

* Табличное представление всех ответов с указанием даты и времени их получения
* Сортировка ответов по различным параметрам, в первую очередь по дате создания
* Постраничная навигация при большом количестве ответов
* Расширяемые строки таблицы с детальной информацией по каждому ответу
* Визуализация различных типов ответов (текст, число, булево значение, дата, выбор из вариантов)
* Оптимизированное отображение длинных текстовых ответов с возможностью просмотра полного текста
* Индикация загрузки данных и обработка возможных ошибок
* Информационное сообщение при отсутствии ответов

Данная страница доступна только авторизованным пользователям, являющимся владельцами соответствующей формы, что обеспечивает конфиденциальность собранных данных.



Изображение выглядит как текст, снимок экрана, программное обеспечение, линия

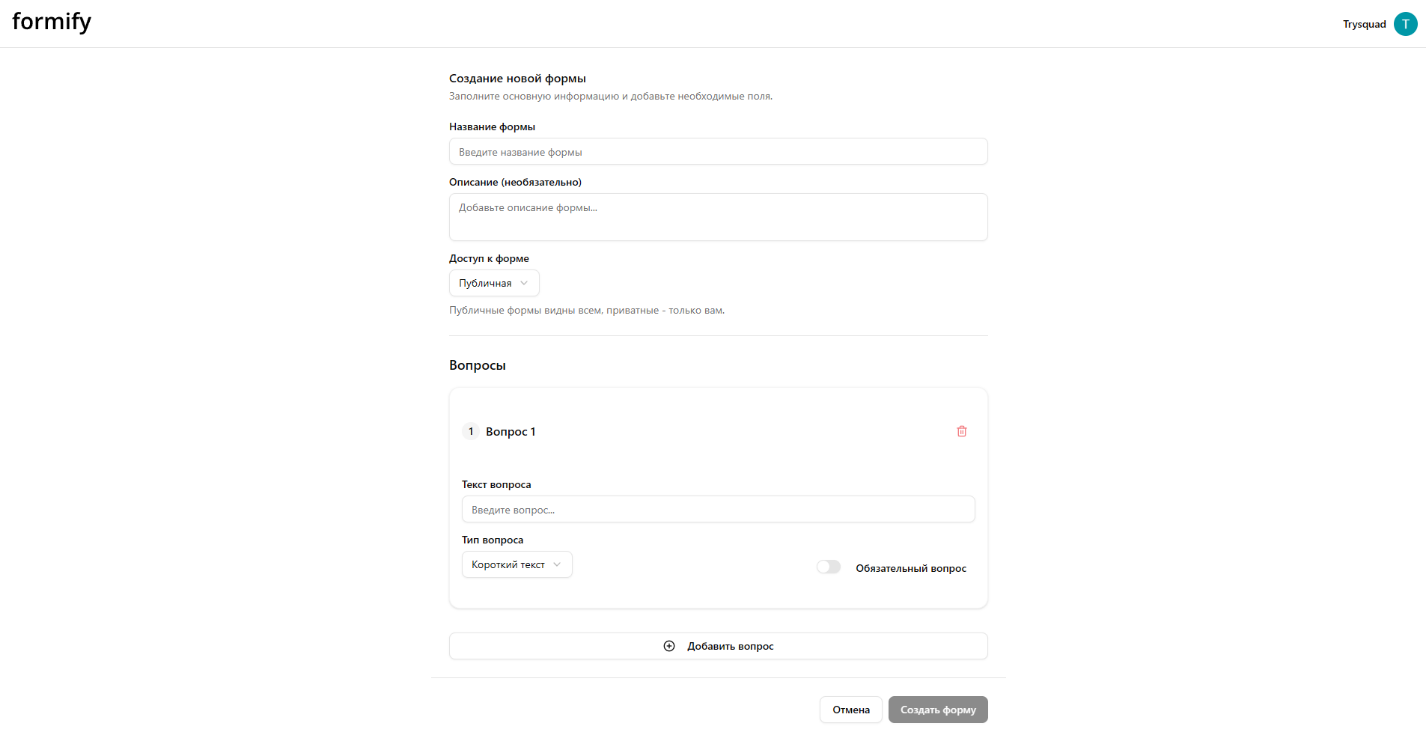
Контент, сгенерированный ИИ, может содержать ошибки.

**Страница создания новой формы (form/new)**

Страница создания новой формы представляет собой интерактивный конструктор, позволяющий пользователю сформировать структуру опроса без необходимости программирования. Функциональные возможности:

* Интуитивно понятный интерфейс с пошаговым заполнением необходимых полей
* Ввод основной информации о форме (название, описание);
* Настройка параметров доступа к форме (публичная или приватная);
* Добавление неограниченного количества вопросов различных типов:
  + Короткий текст;
  + Абзац (длинный текст);
  + Число;
  + Булево значение (да/нет);
  + Дата;
  + Email;
  + Телефон;
  + Выбор одного варианта из нескольких;
  + Выбор нескольких вариантов;
* Для вопросов с вариантами ответов: возможность добавления, удаления и редактирования вариантов;
* Настройка обязательности ответа на каждый вопрос;
* Перемещение вопросов для изменения их порядка в форме;
* Предварительная валидация всех полей с информативными сообщениями об ошибках;
* Индикация процесса сохранения формы;
* Автоматический переход на страницу созданной формы после успешного сохранения.

Данная страница реализует принцип "Create" из концепции CRUD и обеспечивает основу для дальнейшей работы с формами.

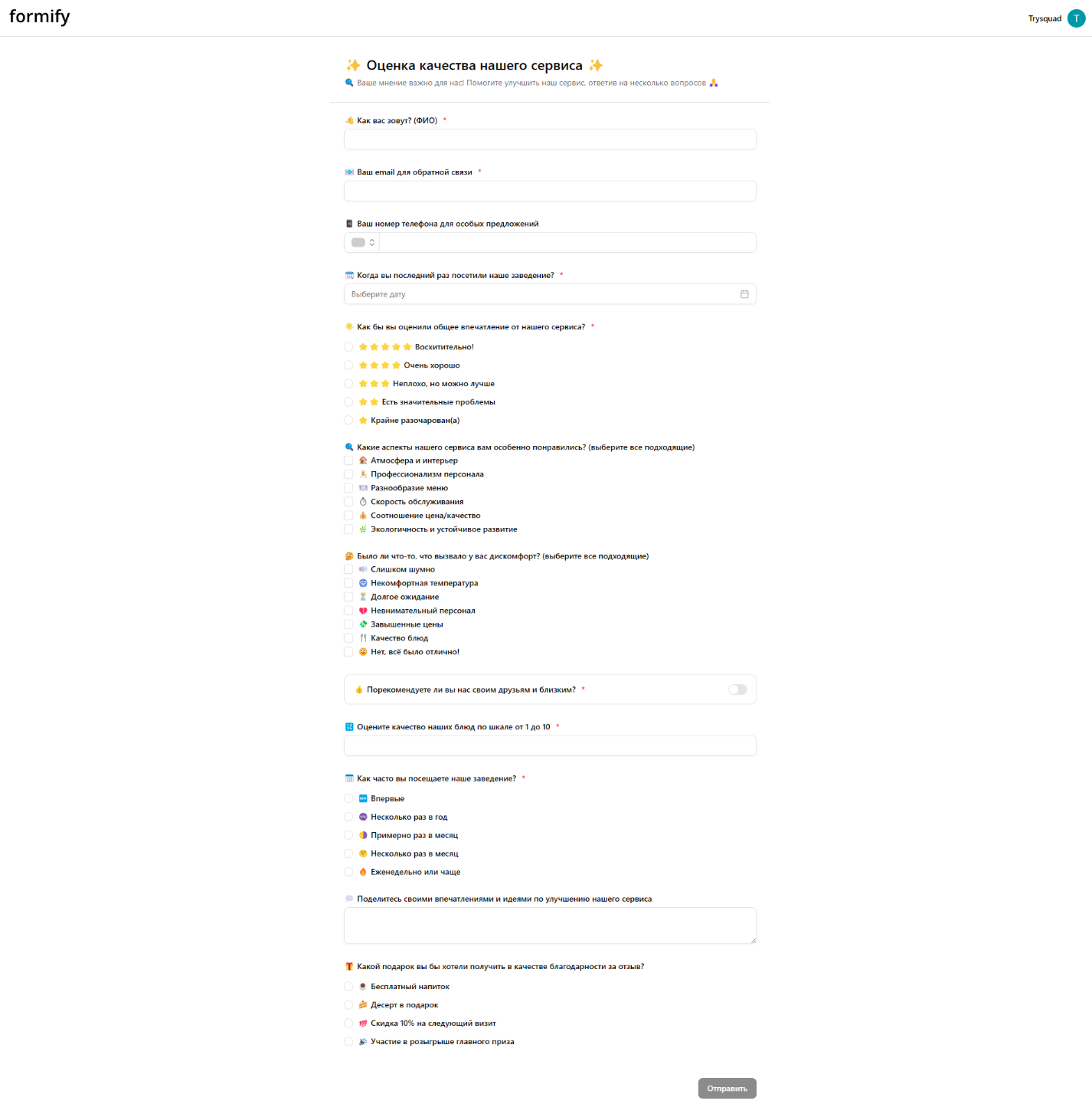


**Страница просмотра и заполнения формы (form/[slug])**

Страница отображения формы предназначена для конечных пользователей, которые будут заполнять форму. Идентификатор формы передается через параметр [slug] в URL. Основной функционал:

* Отображение заголовка и описания формы
* Рендеринг вопросов в соответствии с установленным порядком
* Специализированные компоненты ввода для каждого типа вопроса (текстовые поля, переключатели, календари и т.д.)
* Автофокус на первом поле с ошибкой при попытке отправки с невалидными данными
* Валидация данных в режиме реального времени с наглядной индикацией ошибок:
  + Обязательность заполнения
  + Формат email
  + Формат телефона
  + Числовые значения
  + Минимальное количество выбранных вариантов
* Отображение прогресса отправки формы
* Страница благодарности после успешной отправки ответа
* Обработка ошибок при загрузке формы или отправке ответов
* Адаптивный дизайн для корректного отображения на мобильных устройствах

Данная страница может быть доступна как авторизованным, так и неавторизованным пользователям, в зависимости от настроек доступа формы.

****

**Страница профиля пользователя (profile)**

Страница профиля служит персональным кабинетом пользователя для управления созданными формами. Функциональные возможности:

* Отображение списка всех форм, созданных пользователем
* Карточки форм с основной информацией (название, описание, дата создания, дата обновления)
* Быстрый доступ к управлению формой через контекстное меню
* Изменение настроек доступа формы (публичная/приватная)
* Возможность удаления форм с подтверждением
* Переход к просмотру ответов на конкретную форму
* Переход к редактированию формы
* Копирование ссылки на форму для распространения
* Создание новой формы через кнопку "Создать форму"
* Информативное сообщение и призыв к действию при отсутствии созданных форм
* Индикация загрузки при обращении к серверу

Данная страница реализует принципы "Read", "Update" и "Delete" из концепции CRUD, позволяя пользователю полностью управлять жизненным циклом созданных форм.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

### 2.1.3 Описание связей между страницами

Разработанное веб-приложение представляет собой целостную систему взаимосвязанных страниц, обеспечивающих комплексную работу с формами на всех этапах их жизненного цикла. Организация навигационных потоков и функциональных связей между страницами построена с учетом логики использования приложения и удобства пользовательского опыта.

Центральным элементом навигационной структуры является главная страница (/), выполняющая роль входной точки приложения. Она содержит основные навигационные элементы, направляющие пользователя к ключевым функциональным блокам системы: созданию новой формы и просмотру существующих форм пользователя. Главная страница намеренно минималистична и ориентирована на быстрый переход к основным функциям приложения.

Страница профиля пользователя (/profile) является агрегатором всех форм, созданных пользователем, и служит центром управления ими. Эта страница имеет двунаправленные связи с другими компонентами системы:

* С главной страницей через навигационные элементы верхнего меню
* С формами создания (/form/new) и редактирования через кнопки действий
* С представлением конкретных форм (/form/[slug]) через ссылки на карточках форм
* С просмотром ответов на формы (/answers/[slug]) через контекстное меню карточек форм

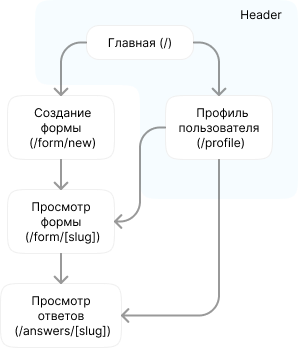
Страница создания новой формы (/form/new) является изолированным функциональным блоком с односторонними связями. После успешного завершения создания происходит автоматическое перенаправление на страницу просмотра созданной формы (/form/[slug]) с новым идентификатором. В случае отмены создания пользователь возвращается на страницу профиля.

Страница просмотра и заполнения формы (/form/[slug]) является динамической и зависит от идентификатора формы, переданного в URL-параметре. Она может отображаться в двух режимах:

1. Для владельца формы: с дополнительными элементами управления, включая кнопки перехода к редактированию и просмотру ответов
2. Для респондента: с функционалом заполнения и отправки ответов

В первом режиме страница имеет связи со страницей ответов (/answers/[slug]) и страницей профиля. Во втором режиме после успешной отправки формы пользователь видит страницу благодарности, являющуюся терминальным состоянием данного пользовательского сценария.

Страница просмотра ответов (/answers/[slug]) доступна только владельцу формы и имеет обратные связи со страницей просмотра формы и страницей профиля. Она является завершающим элементом в цикле работы с формой, предоставляя аналитические возможности по собранным данным.



Важно отметить наличие постоянных глобальных навигационных элементов (шапка приложения), обеспечивающих доступ к ключевым страницам (главная, профиль) из любой точки приложения. Это повышает удобство использования и позволяет пользователю быстро перемещаться между различными функциональными блоками.

Система авторизации интегрирована в навигационную структуру приложения и обеспечивает контроль доступа к страницам в зависимости от аутентификационного статуса пользователя. Неавторизованные пользователи имеют ограниченный доступ только к главной странице и к публичным формам для заполнения.

Технически связи между страницами реализованы с использованием компонента Link из Next.js, обеспечивающего клиентскую навигацию без полной перезагрузки страницы, что значительно улучшает пользовательский опыт. Программное перенаправление реализовано через хук useRouter, предоставляющий методы для навигации при обработке действий пользователя.

Такая организация связей между страницами обеспечивает логичное и интуитивно понятное перемещение пользователя в рамках приложения, соответствуя концепции CRUD и принципам современного пользовательского интерфейса.

## 2.2 Архитектура сервер

### 2.2.1 Описание файловой структуры

Серверная часть приложения реализована с использованием языка Go и следует принципам модульной архитектуры, обеспечивающей четкое разделение ответственности между компонентами. В основе организации кодовой базы лежит структурный подход, позволяющий эффективно управлять сложностью системы и обеспечивать её масштабируемость.

Корневая структура серверной части разделена на два основных каталога: **apps** — содержащий исполняемые приложения, и **lib** — включающий общие библиотеки и утилиты, используемые в различных компонентах системы.

В директории api-gql реализован основной сервер приложения, предоставляющий GraphQL API для взаимодействия с клиентской частью. Структура сервера следует принципам чистой архитектуры с четким разделением на слои:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Транспортный слой** (директория internal/delivery) отвечает за преобразование внешних запросов во внутренние структуры данных и наоборот. Данный слой включает два основных компонента:
   * **GraphQL** (internal/delivery/gql) — определяет схему GraphQL, резолверы и директивы для обработки запросов.
   * **HTTP** (internal/delivery/http) — содержит HTTP-обработчики для дополнительных эндпоинтов, не охваченных GraphQL.
2. **Сервисный слой** (internal/services) реализует бизнес-логику приложения, включая работу с токенами и пользователями. Здесь сосредоточены правила и алгоритмы, не зависящие от способа доставки данных.
3. **Слой аутентификации** (internal/auth) обеспечивает механизмы проверки подлинности и авторизации пользователей с использованием сессий и куков. Особое внимание уделено интеграции с Google OAuth для обеспечения удобного входа пользователей через существующие аккаунты Google.
4. **Серверный слой** (internal/server) отвечает за конфигурацию и запуск HTTP-сервера с использованием фреймворка Gin. Здесь определены middleware-компоненты, настройки CORS и обработка запросов.

Особое внимание заслуживает реализация GraphQL API, которая основана на библиотеке gqlgen. Структура GraphQL-схемы разделена на несколько файлов в директории internal/delivery/gql/schema:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Система аутентификации и авторизации реализована с использованием сессионного подхода, что отражено в структуре директории internal/auth:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Для организации зависимостей и обеспечения инверсии управления используется фреймворк UberFX, что позволяет избежать жесткого связывания компонентов и упрощает тестирование.

В директории lib размещены общие библиотеки и утилиты, используемые на сервере:

1. **Модели данных** (gomodels) содержат определения структур, соответствующих таблицам базы данных PostgreSQL, включая:
   * form.go — модели для форм, вопросов и вариантов ответов
   * form\_answer.go — модели для ответов пользователей
   * users.go — модель пользователя
   * tokens.go — токены для аутентификации
2. **Конфигурация** (config) предоставляет механизмы загрузки и управления конфигурацией приложения из различных источников.
3. **Миграции** (migrations) содержат скрипты для создания и обновления структуры базы данных, включая автоматическое создание таблиц с использованием GORM ORM.
4. **Интеграция с Google** (google) реализует взаимодействие с API Google для аутентификации пользователей через OAuth 2.0.

Взаимодействие с базой данных PostgreSQL осуществляется через ORM-библиотеку GORM, что обеспечивает типобезопасную работу с данными и абстрагирует приложение от деталей конкретной СУБД.

### 2.2.2 Описание API

Взаимодействие между клиентской и серверной частями приложения осуществляется через GraphQL API, что позволяет оптимизировать сетевой трафик и обеспечивает гибкость при запросе данных. GraphQL, в отличие от традиционного REST API, позволяет клиенту точно указывать, какие данные ему необходимы, что особенно важно при работе со сложными структурами данных, такими как формы с различными типами вопросов и вариантами ответов.

**Аутентификация и авторизация.**

В приложении реализован механизм аутентификации на основе сессий с использованием HTTP-куков. Этот подход обеспечивает более высокий уровень безопасности по сравнению с токенами JWT, поскольку чувствительная информация хранится на сервере, а не в браузере пользователя.

Для удобства пользователей реализована возможность входа через Google OAuth 2.0, что позволяет использовать существующие учётные записи Google без необходимости создания дополнительных учётных данных. Процесс аутентификации включает следующие эндпоинты:

* /auth/register - регистрация нового пользователя
* /auth/login - вход по email и паролю
* /google/login - начало процесса аутентификации через Google
* /google/callback - обработчик ответа от Google OAuth
* /auth/session - проверка статуса аутентификации
* /logout - завершение сессии пользователя

Для защиты операций GraphQL, требующих аутентификации, используется директива @isAuthenticated, которая проверяет наличие действительной сессии пользователя перед выполнением запроса.

**Общие принципы работы с API.**

Разработанное приложение предоставляет GraphQL API, предназначенное для создания, редактирования и анализа интерактивных форм. API доступно по эндпоинту /graphql и предоставляет комплексный набор операций для работы с формами, вопросами и ответами. В данном разделе описаны основные возможности API и способы взаимодействия с ним для разработчиков сторонних приложений.

API построено на основе спецификации GraphQL, что предоставляет следующие преимущества:

* Возможность запрашивать только необходимые данные
* Получение нескольких ресурсов в одном запросе
* Строгая типизация данных
* Интроспекция схемы для автодокументации

Все запросы отправляются на единую конечную точку /graphql с использованием HTTP метода POST и содержат JSON-payload со следующей структурой:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Примеры операций GraphQL.**

Получение детальной информации о форме

**Изображение выглядит как текст, снимок экрана, Шрифт, рукописный текст

Контент, сгенерированный ИИ, может содержать ошибки.**

* id (обязательно) - идентификатор формы

**Доступные операции API.**

API предоставляет следующие типы запросов и мутаций для работы с сущностями системы:

Запросы (Queries):

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Мутации (Mutations):

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

API предоставляет полный набор операций для реализации концепции CRUD (Create, Read, Update, Delete) в отношении форм, вопросов и ответов, что позволяет эффективно интегрировать его с различными клиентскими приложениями и сервисами.

## 2.3 Описание структуры базы данных

Для хранения данных о пользователях, формах, вопросах и ответах в приложении используется реляционная база данных PostgreSQL. Структура базы данных спроектирована с учётом требуемой функциональности и масштабируемости системы, обеспечивая эффективное хранение и получение информации. В данном разделе приведено описание схемы базы данных и взаимосвязей между её компонентами.

База данных разработана в соответствии с принципами реляционной модели и состоит из восьми взаимосвязанных таблиц, каждая из которых представляет определённую сущность в системе или связь между сущностями:

1. **users** — информация о пользователях системы
2. **tokens** — токены аутентификации
3. **forms** — формы, созданные пользователями
4. **questions** — вопросы в формах
5. **options** — варианты ответов для вопросов с выбором
6. **form\_responses** — ответы на формы
7. **answers** — ответы на конкретные вопросы
8. **answer\_options** — связь между ответами и выбранными вариантами

Структура базы данных реализует все необходимые связи для поддержки полного цикла работы с формами, включая создание, редактирование, заполнение и анализ.

Изображение выглядит как текст, диаграмма, План, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Таблица users.**

Таблица users является центральной для системы аутентификации и авторизации. Каждый пользователь может создавать формы и быть их владельцем.

| Поле | Тип | Описание |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор пользователя |
| email | VARCHAR | Электронная почта (уникальная) |
| display\_name | VARCHAR | Отображаемое имя пользователя |
| picture | TEXT | URL изображения профиля |
| google\_id | VARCHAR | Идентификатор пользователя в Google (для OAuth) |
| is\_banned | BOOLEAN | Флаг блокировки пользователя |
| created\_at | TIMESTAMP | Дата и время создания записи |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |
| passwordHash | TEXT | Хэшированный пароль для аутентификации |

**Таблица tokens.**

Хранит токены для аутентификации пользователей:

| Поле | Тип | Описание |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор токена |
| user\_id | UUID | Идентификатор пользователя (внешний ключ) |
| token | TEXT | Значение токена |
| type | VARCHAR(50) | Тип токена (доступ, восстановление и т.д.) |
| expires\_at | TIMESTAMP | Срок действия токена |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

Таблица tokens поддерживает механизм аутентификации и хранит токены различных типов, связанные с конкретными пользователями.

**Таблица forms.**

Содержит информацию о созданных формах:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор формы |
| owner\_id | UUID | Идентификатор владельца формы (внешний ключ) |
| title | VARCHAR | Заголовок формы |
| description | TEXT | Описание формы |
| access | VARCHAR(20) | Тип доступа (PRIVATE, BY\_LINK, PUBLIC) |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

Таблица forms является центральной для функциональности приложения и хранит основную информацию о формах. Поле access определяет уровень доступа к форме: приватный (только владелец), по ссылке или публичный.

**Таблица questions.**

Хранит вопросы, включенные в формы:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор вопроса |
| form\_id | UUID | Идентификатор формы (внешний ключ) |
| text | TEXT | Текст вопроса |
| type | VARCHAR(20) | Тип вопроса (см. перечисление типов) |
| required | BOOLEAN | Флаг обязательности ответа |
| order | INTEGER | Порядковый номер в форме |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

Поле **type** определяет тип вопроса и может принимать значения: SHORT\_TEXT, PARAGRAPH, BOOLEAN, NUMBER, PHONE, DATE, EMAIL, SINGLE\_CHOICE, MULTIPLE\_CHOICE. Типы вопросов определяют способ ввода и валидации ответов.

**Таблица options.**

Содержит варианты ответов для вопросов с выбором:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор варианта |
| question\_id | UUID | Идентификатор вопроса (внешний ключ) |
| text | TEXT | Текст варианта ответа |
| order | INTEGER | Порядковый номер варианта |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

**Таблица form\_responses.**

Хранит информацию о заполненных формах:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор ответа на форму |
| form\_id | UUID | Идентификатор формы (внешний ключ) |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

Каждая запись в таблице form\_responses представляет один заполненный экземпляр формы и связана с конкретными ответами на вопросы.

**Таблица answers.**

Содержит ответы на конкретные вопросы:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор ответа |
| form\_response\_id | UUID | Идентификатор ответа на форму (внешний ключ) |
| question\_id | UUID | Идентификатор вопроса (внешний ключ) |
| text\_value | TEXT | Текстовый ответ |
| bool\_value | BOOLEAN | Булевый ответ |
| number\_value | FLOAT | Числовой ответ |
| date\_value | TIMESTAMP | Ответ-дата |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

**Таблица answer\_options.**

Связывает ответы с выбранными вариантами для вопросов с выбором:

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | UUID | Уникальный идентификатор связи |
| answer\_id | UUID | Идентификатор ответа (внешний ключ) |
| option\_id | UUID | Идентификатор варианта (внешний ключ) |
| created\_at | TIMESTAMP | Дата и время создания |
| updated\_at | TIMESTAMP | Дата и время последнего обновления |

Таблица answer\_options реализует связь "многие ко многим" между ответами и вариантами, что необходимо для вопросов с множественным выбором.

**Связи между таблицами.**

Связи между таблицами организованы по принципу внешних ключей, что обеспечивает целостность данных и отражает логические взаимоотношения между сущностями:

1. Пользователи и формы: один пользователь может создать много форм (связь "один ко многим" между users и forms через поле owner\_id).
2. Формы и вопросы: одна форма содержит много вопросов (связь "один ко многим" между forms и questions через поле form\_id).
3. Вопросы и варианты ответов: один вопрос может иметь несколько вариантов ответа (связь "один ко многим" между questions и options через поле question\_id).
4. Формы и ответы на формы: на одну форму может быть дано много ответов (связь "один ко многим" между forms и form\_responses через поле form\_id).
5. Ответы на формы и ответы на вопросы: один ответ на форму содержит много ответов на вопросы (связь "один ко многим" между form\_responses и answers через поле form\_response\_id).
6. Вопросы и ответы: один вопрос может иметь много ответов в разных заполненных формах (связь "один ко многим" между questions и answers через поле question\_id).
7. Ответы и выбранные варианты: для вопросов с выбором ответ может быть связан с несколькими вариантами (связь "многие ко многим" между answers и options через таблицу answer\_options).
8. Пользователи и токены: один пользователь может иметь несколько токенов (связь "один ко многим" между users и tokens через поле user\_id).

**Индексы и оптимизация.**

Для обеспечения высокой производительности запросов в базе данных созданы следующие индексы:

1. Индексы по внешним ключам для оптимизации операций соединения таблиц
2. Уникальный индекс по полю email в таблице users
3. Уникальный индекс по паре полей (form\_response\_id, question\_id) в таблице answers для обеспечения уникальности ответа на вопрос в рамках одного заполнения формы
4. Уникальный индекс по паре полей (answer\_id, option\_id) в таблице answer\_options для предотвращения дублирования выбора варианта
5. Индексы по полям сортировки (order) в таблицах questions и options
6. Индекс по полю access в таблице forms для оптимизации запросов, фильтрующих формы по типу доступа

Такая структура индексов обеспечивает эффективное выполнение наиболее частых запросов, таких как получение всех вопросов формы, получение всех ответов на форму, а также поиск и фильтрация форм.

# Глава 3. Реализация приложения

## 3.1 Описание процесса реализации серверной части приложения

### 3.1.1 Проектирование базы данных и миграции

На основе выявленных требований была спроектирована структура базы данных, включающая таблицы для хранения пользователей, форм, вопросов, вариантов ответов и полученных ответов. Для создания и поддержания схемы базы данных использован механизм автомиграций GORM, позволяющий автоматически генерировать SQL-скрипты на основе определенных моделей данных:



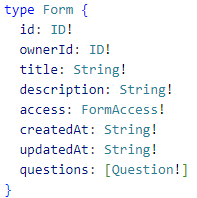
Преимуществом данного подхода является автоматическое обновление схемы при изменении структуры моделей, что значительно упростило процесс разработки и тестирования. Модели данных определены с использованием структур Go и тегов GORM для указания особенностей отображения в базе данных.

### 3.1.2 Реализация GraphQL API

Центральным компонентом серверной части стал GraphQL API, обеспечивающий гибкий интерфейс для взаимодействия с клиентским приложением. Процесс его реализации включал следующие этапы:

Центральным компонентом серверной части стал GraphQL API, обеспечивающий гибкий интерфейс для взаимодействия с клиентским приложением. Процесс его реализации включал следующие этапы:

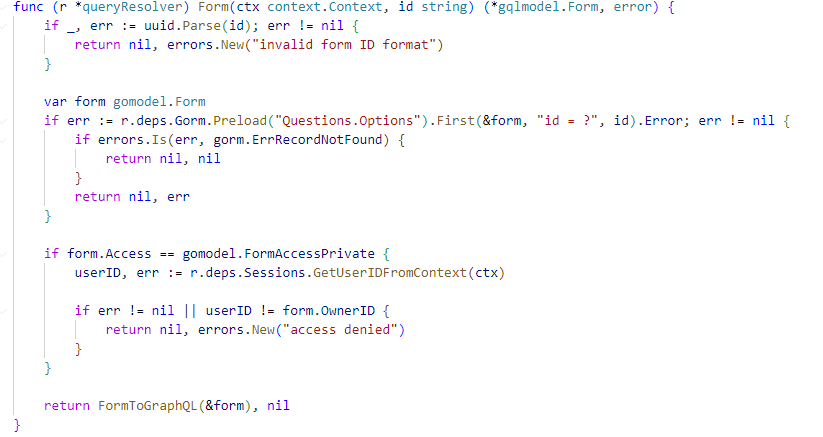
1. **Определение схемы GraphQL**:



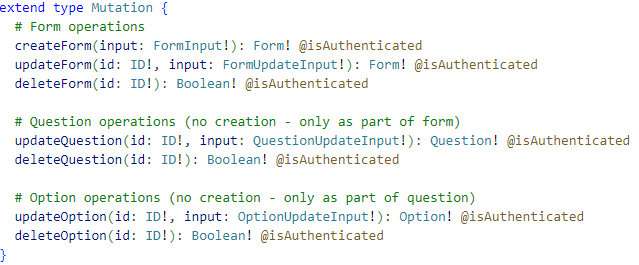
1. **Генерация кода** с использованием gqlgen для создания резолверов и типов:



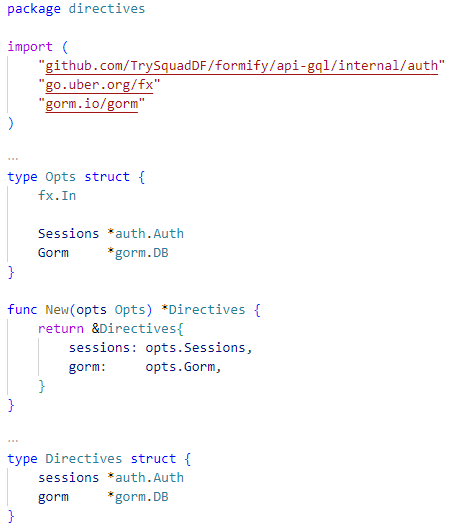
1. **Реализация резолверов** для обработки различных типов запросов, например:



1. **Настройка директив безопасности** для ограничения доступа к определенным операциям:









Особое внимание уделено эффективности запросов к базе данных для предотвращения проблемы N+1 запросов при получении связанных данных, таких как вопросы в форме или варианты ответов.

### 3.1.3 Реализация системы аутентификации и авторизации

Для обеспечения безопасности приложения разработана система аутентификации на основе сессионного подхода с использованием HTTP-куков. Процесс включал:

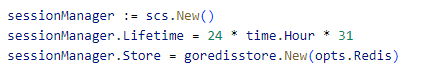
1. **Настройку Google OAuth 2.0** для аутентификации пользователей:



1. **Реализацию классической аутентификации** по email и паролю:

* Создание эндпоинтов для регистрации (/auth/register) и входа (/auth/login)
* Безопасное хранение паролей с использованием алгоритма bcrypt
* Валидация учетных данных при входе в систему
* Механизм обработки ошибок аутентификации с информативными сообщениями

1. **Реализацию сессионного механизма** с использованием библиотеки SCS и хранилища Redis



:

1. **Создание middleware для проверки аутентификации**:



1. **Реализацию директивы GraphQL** для защиты определенных операций.

Такой подход обеспечивает более высокий уровень безопасности по сравнению с JWT-токенами, поскольку чувствительная информация хранится на сервере, а не в браузере пользователя.

### 3.1.4 Инверсия зависимостей

Для эффективного управления зависимостями и обеспечения тестируемости кода использован фреймворк Uber FX, позволяющий реализовать принцип инверсии зависимостей. Основные компоненты внедряются через конструкторы:

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Контент, сгенерированный ИИ, может содержать ошибки.

Данный подход позволил:

* Централизовать создание и настройку компонентов
* Упростить тестирование отдельных модулей
* Обеспечить четкий порядок инициализации и завершения работы компонентов

### 3.1.5 Организация HTTP-сервера

Фреймворк Gin использован для создания HTTP-сервера и обработки запросов. Настройка сервера включала:

1. **Конфигурацию CORS** для обеспечения безопасного взаимодействия с клиентской частью:

****

1. **Настройку middlewar**  для аутентификации и обработки контекста:

****

1. **Настройку маршрутов** для GraphQL API и OAuth2:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

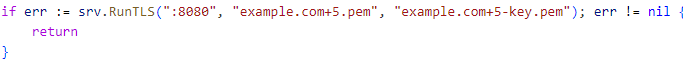
Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. **Конфигурацию TLS** для защищенного соединения:



## 3.2 Описание процесса реализации клиентской части приложения

### 3.2.1 Реализация системы аутентификации и авторизации

В основу клиентской части был положен фреймворк Next.js, обеспечивающий широкие возможности для создания современных React-приложений с серверным рендерингом, статической генерацией страниц и автоматической маршрутизацией. В проекте использована новейшая версия Next.js с App Router, что позволило организовать маршрутизацию на основе файловой системы и эффективно использовать серверные и клиентские компоненты.

Структура проекта организована по методологии Feature-Sliced Design (FSD), адаптированной под особенности Next.js. Основные слои проекта включают:

* app — маршруты приложения согласно архитектуре Next.js App Router;
* components — переиспользуемые UI-компоненты;
* src/api — функции для взаимодействия с API;
* src/gql — автогенерированные типы и запросы GraphQL;
* src/processes — сложные бизнес-процессы, такие как аутентификация;
* src/shared — общие компоненты и утилиты;
* src/widgets — композиционные блоки интерфейса;

### 3.2.2 Настройка Apollo Client для взаимодействия с GraphQL API

Для организации взаимодействия с серверным GraphQL API был настроен Apollo Client. Реализация включала создание провайдера, обеспечивающего доступ к клиенту Apollo во всем приложении:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Конфигурация клиента включала настройку кеширования, обработки ошибок и обеспечения передачи куков с сессионной информацией для авторизованных запросов. Такой подход предоставил унифицированный способ работы с данными, включая управление кешем и оптимистичные обновления.

### 3.2.3 Система аутентификации и авторизации

Система аутентификации в данном веб-приложении реализована с использованием современных подходов и технологий для обеспечения безопасности и удобства пользователей. Основная архитектура построена на основе клиент-серверного взаимодействия, где клиентская часть использует Redux для управления состоянием аутентификации, а серверная часть обрабатывает запросы и управляет сессиями пользователей.

**Компоненты системы аутентификации**

Система аутентификации включает следующие ключевые компоненты:

1. **Хук управления сессией** (useUserSession) - центральный элемент, предоставляющий доступ к функциям аутентификации.
2. **Хранилище Redux** с отдельным slice для состояния аутентификации.
3. **API интеграция** через GraphQL запросы и REST-эндпоинты.
4. **UI компоненты** для процесса аутентификации (модальные окна входа и регистрации).
5. **Защита маршрутов** для авторизации доступа к определенным страницам.

Приложение поддерживает несколько методов аутентификации:

**Аутентификация по email и паролю**

Классический метод аутентификации реализован с использованием формы входа, где пользователи указывают свой email и пароль:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Функция emailLogin выполняет запрос к серверу с учетными данными пользователя, а затем обновляет состояние Redux в случае успеха.

**Интеграция с Google OAuth**

Система также поддерживает аутентификацию через Google OAuth, что позволяет пользователям входить с использованием своих аккаунтов Google:



**Регистрация новых пользователей**

Процесс регистрации реализован с использованием отдельной формы, где пользователи указывают имя, email и пароль:

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Механизм выхода из системы**

Выход из системы реализован через функцию handleLogout, которая выполняет соответствующий запрос к серверу и обновляет состояние приложения:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Управление состоянием аутентификации с использованием Redux**

Состояние аутентификации централизовано хранится в Redux store и управляется с помощью slice authSlice:

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

**Управление сессиями пользователей**

Сессия пользователя управляется через hook useUserSession, который обеспечивает доступ к текущему состоянию аутентификации и методам для работы с ним:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

### 3.2.4 Разработка системы создания и управления формами

Ключевым функциональным блоком приложения является система создания и редактирования форм. Для страницы создания новой формы реализован динамический конструктор с возможностью добавления различных типов вопросов.

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Для каждого типа вопроса были разработаны специализированные компоненты ввода, учитывающие особенности представления и взаимодействия с пользователем. Система поддерживает 8 типов вопросов:

* Короткий текст
* Абзац (длинный текст)
* Число
* Булево значение (да/нет)
* Дата
* Email
* Телефон
* Выбор одного или нескольких вариантов

Реализована возможность изменения порядка вопросов через перетаскивание, а также добавления и удаления вариантов ответов для вопросов с выбором.

### 3.2.5 Система просмотра и заполнения форм

Для просмотра и заполнения форм был разработан комплексный компонент FormView, который динамически формирует интерфейс в зависимости от структуры вопросов. Этот компонент включает:

1. Загрузку данных формы через GraphQL запрос



1. Динамическое построение схемы валидации Zod на основе полученных вопросов

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. Рендеринг соответствующих полей ввода для каждого типа вопроса

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

1. Обработку отправки формы с трансформацией данных для GraphQL API

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Разработка компонента карточки формы**

Для отображения информации о форме был создан универсальный компонент FormCard, используемый на странице профиля и других разделах приложения. Этот компонент включает следующие функции:

1. Отображение основной информации о форме (название, описание, дата создания)
2. Управление настройками доступа через выпадающее меню
3. Кнопки для быстрого доступа к просмотру ответов и самой форме
4. Функциональность удаления формы с диалогом подтверждения

Изображение выглядит как текст, снимок экрана, документ

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, документ

Контент, сгенерированный ИИ, может содержать ошибки.

**Реализация отправки ответов на формы**

Для обработки отправки заполненных форм был разработан специализированный хук useFormSubmission, который обеспечивает:

1. Трансформацию данных формы в формат, ожидаемый API

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

1. Отправку данных через GraphQL мутацию

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

1. Управление состоянием загрузки и результатом отправки
2. Обработку ошибок и уведомление пользователя

# Глава 4. Тестирование приложения

В данной главе описывается процесс тестирования разработанного веб-приложения, включающий план тестирования и его практическую реализацию. Тестирование является неотъемлемой частью разработки программного обеспечения, позволяющей выявить потенциальные проблемы и убедиться в корректной работе всех функций приложения перед его внедрением.

### 4.1. План тестирования

**Цели и задачи тестирования**

Основной целью тестирования является проверка соответствия разработанного веб-приложения заявленным требованиям и подтверждение корректного функционирования всех его компонентов. В рамках тестирования были поставлены следующие задачи:

1. Проверка работоспособности функции регистрации пользователей
2. Тестирование процесса аутентификации пользователей
3. Проверка функциональности авторизации и разграничения доступа
4. Тестирование основных пользовательских сценариев работы с приложением
5. Проверка корректности работы API приложения

**Методы тестирования**

Для достижения поставленных целей были выбраны следующие методы тестирования:

**Ручное тестирование** - последовательное выполнение базовых пользовательских сценариев с фиксацией результатов

**Объекты тестирования**

В рамках тестирования были выделены следующие ключевые объекты:

1. **Система аутентификации и авторизации**:
   * Регистрация новых пользователей
   * Вход существующих пользователей
   * Выход из системы
2. **Основной функционал приложения**:
   * Просмотр доступных курсов
   * Создание и редактирование учебных материалов
   * Взаимодействие с образовательным контентом
3. **API приложения**:
   * Аутентификация через API
   * Получение данных пользователя
   * Получение и изменение образовательного контента

**Тестовые сценарии**

Для ручного тестирования были разработаны следующие сценарии:

1. **Регистрация нового пользователя**:
   * Открытие формы регистрации
   * Заполнение полей валидными данными
   * Проверка успешной регистрации и последующего входа
2. **Аутентификация пользователя**:
   * Вход с использованием email и пароля
   * Проверка доступности функций авторизованного пользователя
3. **Выход из системы**:
   * Нажатие на кнопку выхода
   * Проверка корректного завершения сессии
4. **Работа с формами**:
   * Просмотр списка созданных форм
   * Создание новой формы
   * Просмотр ответов на формы

### 4.2. Процесс и результаты тестирования

Ручное тестирование системы аутентификации

**Тестирование процесса регистрации**

Первым шагом было тестирование функции регистрации нового пользователя. Для этого использовался сценарий с заполнением формы регистрации валидными данными:

1. Открытие главной страницы приложения
2. Нажатие на кнопку "Войти"
3. Переход на вкладку "Регистрация"
4. Заполнение полей формы:
   * Имя: "MAI"
   * Email: "mai@mai.education"
   * Пароль: "Test123@3!"
5. Нажатие на кнопку "Зарегистрироваться"

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Результат: Система успешно создала нового пользователя и выполнила автоматический вход. В правом верхнем углу отобразилось имя пользователя, что подтверждает успешное завершение процесса регистрации.



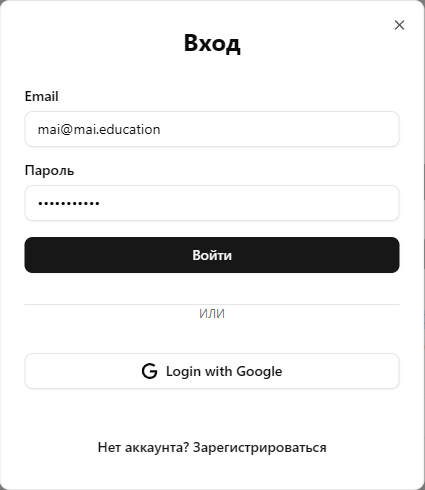
**Тестирование процесса входа**

Следующим этапом было тестирование функции входа существующего пользователя:

1. Открытие главной страницы приложения
2. Нажатие на кнопку "Войти"
3. Ввод учетных данных:
   * Email: " mai@mai.education"
   * Пароль: "Test123@3!"
4. Нажатие на кнопку "Войти"

Изображение выглядит как текст, снимок экрана, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.



Результат: Система успешно аутентифицировала пользователя. В интерфейсе отобразился профиль пользователя, что подтверждает успешное завершение процесса входа.



**Тестирование процесса выхода**

Для проверки корректности завершения сессии был протестирован процесс выхода:

1. Нажатие на иконку профиля в верхнем меню
2. Выбор пункта "Выйти" в выпадающем меню

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

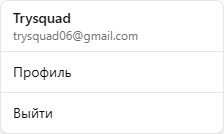
Результат: Система корректно завершила сессию пользователя. Интерфейс обновился, показывая опцию входа вместо профиля пользователя.

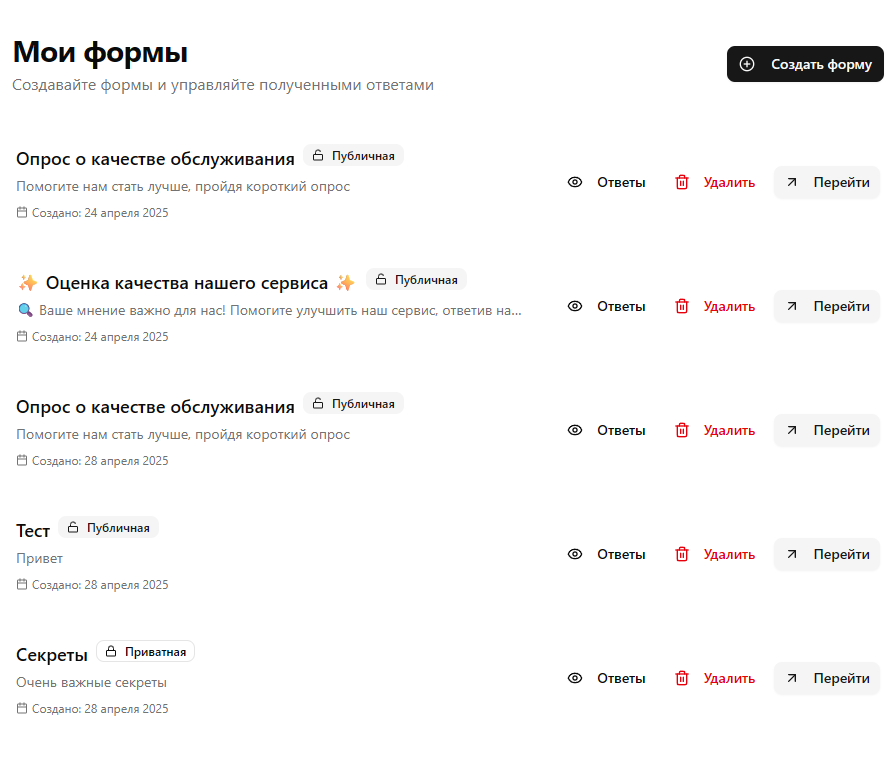


**Просмотр доступных форм**

Для проверки основной функциональности был протестирован сценарий просмотра доступных форм:

1. Вход в систему с учетными данными пользователя
2. Переход на страницу "Профиль"





Результат: Система успешно отобразила список доступных курсов с информацией о каждом из них.

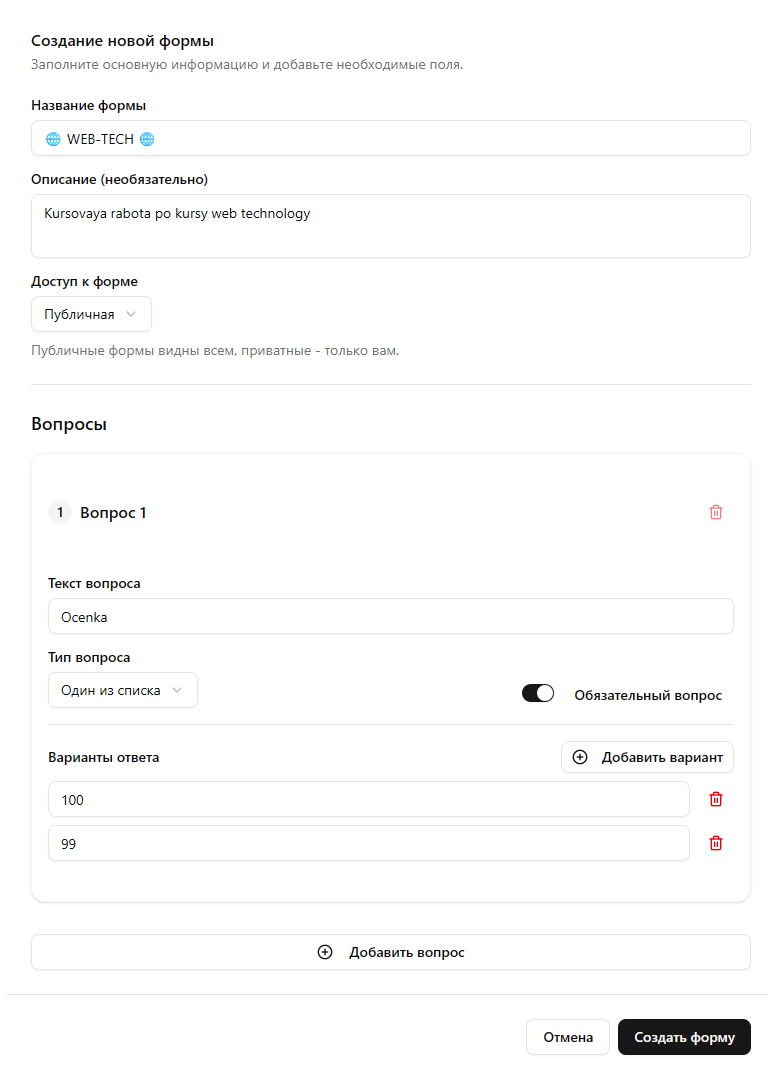
**Создание новой формы**

Для проверки возможности создания новых форм был выполнен следующий сценарий:

1. Нажатие на кнопку "Создать форму"
2. Ввод названия формы
3. Добавление вопроса с одиночным выбором
4. Сохранение формы

Результат: Система успешно создала новую форму с указанными вопросами и сохранила её.

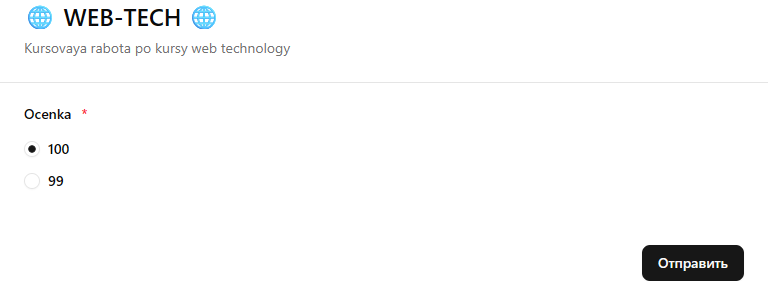




**Заполнение опубликованной формы**

Для проверки возможности заполнения форм был выполнен следующий сценарий:

1. Переход по полученной ссылке опубликованной формы
2. Заполнение всех полей формы
3. Отправка ответов



Результат: Система успешно приняла ответы и отобразила страницу подтверждения.



**Просмотр собранных ответов**

Для проверки функционала просмотра ответов был выполнен следующий сценарий:

1. Вход в систему с учетными данными создателя формы
2. Переход к созданной форме
3. Нажатие на вкладку "Ответы"

Изображение выглядит как текст, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Результат: Система успешно отобразила все полученные ответы с возможностью их анализа.

# Заключение

В результате выполнения курсовой работы было успешно разработано веб-приложение для создания, управления и анализа форм с различными типами вопросов, полностью реализующее весь цикл CRUD-операций. Созданная система представляет собой современное single-page application с дифференциацией прав доступа, обеспечивающее высокую производительность и отзывчивость пользовательского интерфейса.

В ходе выполнения работы были решены все поставленные задачи:

1. Проведен анализ существующих аналогов, выявлены их функциональные возможности, преимущества и недостатки, что позволило определить оптимальный набор функций для разрабатываемого приложения
2. Спроектирована архитектура веб-приложения, основанная на клиент-серверной модели с четким разделением функциональности между Frontend и Backend компонентами, что обеспечило необходимую гибкость, масштабируемость и высокую производительность
3. Разработана клиентская часть приложения с использованием Next.js, React и TypeScript, что позволило создать интуитивно понятный и отзывчивый пользовательский интерфейс
4. Реализована серверная часть на языке Golang с использованием фреймворка Gin и UberFX, обеспечивающая эффективное взаимодействие с базой данных PostgreSQL и обработку запросов клиента через GraphQL API
5. Внедрена система авторизации и аутентификации на основе сессионного подхода с поддержкой OAuth 2.0
6. Разработаны и интегрированы механизмы валидации данных с использованием библиотеки Zod на стороне клиента и соответствующих инструментов на стороне сервера
7. Проведено тестирование приложения, подтвердившее его соответствие функциональным требованиям и удобство использования

Созданное веб-приложение отличается следующими ключевыми преимуществами:

1. **Современная архитектура** - приложение построено с использованием актуальных технологий веб-разработки, обеспечивающих высокую производительность и масштабируемость
2. **Типобезопасность** - применение TypeScript и Zod для валидации данных минимизирует возможность возникновения ошибок в процессе эксплуатации
3. **Удобный пользовательский интерфейс** - благодаря использованию React, Tailwind CSS и Radix UI удалось создать интуитивно понятный и эстетичный интерфейс, соответствующий современным стандартам UX/UI дизайна
4. **Эффективная серверная часть** - высокопроизводительный бэкенд на языке Golang обеспечивает быструю обработку запросов даже при значительной нагрузке
5. **Гибкая система управления данными** - GraphQL API позволяет оптимизировать сетевые запросы и получать только необходимые данные
6. **Надежная аутентификация и авторизация** - применение сессионного подхода и интеграция с Google OAuth 2.0 обеспечивают высокий уровень безопасности

Тестирование разработанного приложения подтвердило его соответствие всем функциональным требованиям. Система позволяет создавать формы с различными типами вопросов, управлять доступом к ним, собирать и анализировать ответы респондентов. Ролевая модель контроля доступа обеспечивает необходимый уровень безопасности и разграничения прав между обычными пользователями и администраторами.

Перспективы дальнейшего развития проекта включают:

1. Расширение аналитического функционала для более глубокого анализа собранных данных
2. Интеграцию с популярными сервисами автоматизации для построения рабочих процессов на основе собранных ответов
3. Разработку мобильного приложения для более удобного использования системы на мобильных устройствах
4. Внедрение технологий машинного обучения для интеллектуального анализа результатов опросов
5. Расширение возможностей коллаборативной работы над формами для команд

Таким образом, поставленная цель курсовой работы была полностью достигнута: разработано функциональное и эргономичное веб-приложение для создания, управления и анализа форм с различными типами вопросов, реализующее полный цикл операций CRUD. Созданное решение может быть использовано широким кругом пользователей для проведения опросов, сбора обратной связи, организации тестирования и других задач, связанных со сбором и анализом структурированной информации.

# Список использованных источников

1. Next.js Documentation [Электронный ресурс] : официальная документация веб-фреймворка / Vercel Inc. — 2025. — URL: [https://nextjs.org/docs](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
2. GraphQL: A query language for your API [Электронный ресурс] : официальная документация языка запросов / GraphQL Foundation. — 2025. — URL: [https://graphql.org/learn/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
3. Apollo Client Documentation [Электронный ресурс] : руководство по использованию библиотеки для GraphQL / Apollo Graph Inc. — 2025. — URL: [https://www.apollographql.com/docs/react/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
4. Gin Web Framework Documentation [Электронный ресурс] : руководство разработчика по HTTP веб-фреймворку для Golang / Gin-Gonic Team. — 2025. — URL: [https://gin-gonic.com/docs/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
5. UberFX: A Go service framework [Электронный ресурс] : документация фреймворка для построения масштабируемых микросервисов / Uber Technologies, Inc. — 2025. — URL: [https://github.com/uber-go/fx](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
6. GORM Documentation [Электронный ресурс] : официальное руководство по ORM-библиотеке для Golang / Jinzhu. — 2025. — URL: [https://gorm.io/docs/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
7. Tailwind CSS Documentation [Электронный ресурс] : официальная документация CSS-фреймворка / Tailwind Labs Inc. — 2025. — URL: [https://tailwindcss.com/docs](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
8. React Hook Form Documentation [Электронный ресурс] : руководство по библиотеке для управления формами в React / React Hook Form. — 2025. — URL: [https://react-hook-form.com/get-started](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
9. TypeScript Documentation [Электронный ресурс] : официальное руководство по языку программирования / Microsoft Corporation. — 2025. — URL: [https://www.typescriptlang.org/docs/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
10. React Documentation [Электронный ресурс] : официальная документация JavaScript-библиотеки для создания пользовательских интерфейсов / Meta Platforms, Inc. — 2025. — URL: [https://reactjs.org/docs/getting-started.html](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
11. PostgreSQL Documentation [Электронный ресурс] : официальное руководство по системе управления базами данных / The PostgreSQL Global Development Group. — 2025. — URL: [https://www.postgresql.org/docs/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
12. Zod Documentation [Электронный ресурс] : руководство по библиотеке валидации схем для TypeScript / Colm Tuite. — 2025. — URL: [https://zod.dev/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
13. Golang Documentation [Электронный ресурс] : официальная документация языка программирования / Google LLC. — 2025. — URL: [https://golang.org/doc/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
14. OAuth 2.0 [Электронный ресурс] : спецификация протокола авторизации / Internet Engineering Task Force (IETF). — 2012. — URL: [https://oauth.net/2/](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).
15. Google Identity: Authentication Documentation [Электронный ресурс] : руководство по интеграции Google аутентификации / Google LLC. — 2025. —URL: [https://developers.google.com/identity/gsi/web/guides/overview](vscode-file://vscode-app/c:/Users/oisa0/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) (дата обращения: 09.05.2025).

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

(обязательное)

URL: [https://github.com/TrySquadDF/fomrify](https://github.com/TrySquadDF/fomrify%20%20)

Репозиторий содержит исходный код.

Дата обращения: 14.05.2025