**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ ……………………………………………………………………… 2  
1. ИЗУЧЕНИЕ ОРГАНИЗАЦИОННОЙ И ФУНКЦИОНАЛЬНОЙ …………….. 3  
2. ОБЗОР И АНАЛИЗ СРЕДСТВ IT ИНФРАСТРУКТУР ……………………. 5  
3. ТЕХНОЛОГИЧЕСКИЕ ПРОЦЕССЫ И ОПЕРАЦИИ ……………………... 8  
4. ПОСТАНОВКА ЗАДАЧИ ………………………………………………….. 10  
5. АНАЛИЗ НЕОБХОДИМОЙ АВТОМАТИЗИЦИИ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ ……………………………………………………………………... 12  
6.ВВЕДЕНИЕ ………………………………………………………………….. 15  
7. СТРУКТУРА ПРОГРАММЫ. ПРОЕКТИРОВАНИЕ …………………….. 16  
8. ТЕСТИРОВАНИЕ …………………………………………………...……… 25  
ЗАКЛЮЧЕНИЕ ……………………………………………….…….…….…… 29  
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ………………………….. 30

**ВВЕДЕНИЕ**

**SenamaSoft** — ведущая IT-компания, специализирующаяся на разработке и поддержке программного обеспечения. Компания зарекомендовала себя как надежный партнер для клиентов, предлагая широкий спектр услуг, от разработки веб и мобильных приложений до серверных решений и облачных технологий. Основной офис компании расположен в Варшаве, Польша, и она активно применяет современные технологии и инструменты, такие как NextJS, React, AWS и Docker, чтобы предоставлять высококачественные и инновационные решения.

Цель прохождения практики в **SenamaSoft** заключалась в получении практического опыта в разработке веб-приложений и углубленном изучении технических процессов компании. Практика позволила мне ознакомиться с полным циклом разработки программного обеспечения, от анализа требований и проектирования до внедрения и поддержки решений.

Практика в **SenamaSoft** предоставила мне возможность работать над реальными проектами, участвовать в команде профессионалов и знакомиться с внутренними процессами компании. Основное внимание было уделено разработке веб-приложений с использованием современных технологий и инструментов, что способствовало приобретению ценного опыта и навыков в области веб-разработки и проектирования программного обеспечения.

**1. ИЗУЧЕНИЕ ОРГАНИЗАЦИОННОЙ И ФУНКЦИОНАЛЬНОЙ СТРУКТУРЫ**

**SenamaSoft** представляет собой динамично развивающуюся компанию в сфере разработки программного обеспечения, ориентированную на высокое качество и инновационные решения. Основанная с целью предоставления широкого спектра IT-услуг, компания активно развивает свою деятельность на международном рынке, предоставляя услуги по разработке мобильных и веб-приложений, серверных решений, а также облачных технологий.

#### **Организационная структура**

**SenamaSoft** использует функциональную организационную структуру, которая позволяет эффективно управлять проектами и ресурсами. Основные элементы этой структуры включают:

1. **Руководство**:
   * **Генеральный директор**: Ответственный за стратегическое руководство и принятие ключевых бизнес-решений. Генеральный директор координирует все основные процессы и обеспечивает выполнение корпоративной стратегии.
2. **Бизнес-аналитики**:
   * Команда бизнес-аналитиков занимается взаимодействием с клиентами для сбора и анализа требований. Они создают технические задания и проектные планы, что обеспечивает точное соответствие требованиям заказчика.
3. **Разработческие команды**:
   * **Фронтенд-разработчики**: Специализируются на создании пользовательских интерфейсов с использованием современных технологий, таких как React и NextJS. Они отвечают за взаимодействие с пользователем и обеспечение удобства работы с приложением.
   * **Бэкенд-разработчики**: Работают над серверной частью приложений, включая работу с базами данных и интеграцию с внешними системами. Используют технологии, такие как Node.js и .NET.
   * **Мобильные разработчики**: Занимаются разработкой мобильных приложений с использованием React Native и Xamarin, что позволяет создавать нативные приложения для платформ Android и iOS.
4. **QA и тестирование**:
   * **QA специалисты**: Обеспечивают тестирование программного обеспечения, выявляют и устраняют ошибки, проводя функциональное тестирование, тестирование производительности и безопасности.
5. **Дизайнеры UX/UI**:
   * Разрабатывают дизайн пользовательских интерфейсов, обеспечивая удобство и привлекательность взаимодействия пользователей с приложением. Они работают над созданием макетов и прототипов.

**Функциональная структура**

Функциональная структура **SenamaSoft** позволяет эффективно управлять процессами разработки и обеспечения качества на всех этапах. Основные функциональные области включают:

**Анализ и планирование**: Сбор информации, создание технического задания и планирование проекта. Бизнес-аналитики и технические писатели играют ключевую роль в этом процессе.

**Разработка и внедрение**: Включает проектирование, кодирование, тестирование и внедрение ПО. Команды работают по методологиям Agile и Scrum, что обеспечивает гибкость и оперативность в выполнении проектов.

**Качество и тестирование**: QA специалисты проводят систематическое тестирование и контроль качества, чтобы гарантировать надежность и стабильность решений.

**Поддержка и обслуживание**: Обеспечение технической поддержки и выполнение обновлений после внедрения решений, а также предоставление консультационных услуг.

**Исследования и разработки**: Изучение новых технологий и внедрение инновационных решений. Команда R&D фокусируется на повышении качества и эффективности разработки.

**SenamaSoft** активно применяет коллективные методы принятия решений. Это позволяет принимать взвешенные решения, которые соответствуют ожиданиям клиентов и требованиям рынка. Такая структура и методы работы способствуют высокой эффективности и качеству предоставляемых услуг, поддерживая конкурентоспособность компании на рынке IT.

**2. ОБЗОР И АНАЛИЗ СРЕДСТВ IT ИНФРАСТРУКТУРЫ**

**SenamaSoft** активно использует современную IT инфраструктуру для обеспечения высококачественной разработки программного обеспечения. В компании применяются передовые технологии и инструменты, что позволяет эффективно управлять проектами и предоставлять надежные решения своим клиентам.

#### **Обзор IT инфраструктуры**

Офис **SenamaSoft** в Бресте, в котором я и проходил практику находится в небольшом здании на Пионерской. Преимущество его небольших размеров в том, что он занимает всего 2 первых этажа, во время работы в любой момент можно выйти на свежий воздух и освободить разум, посмотреть на текущую задачу с другой стороны. Рабочая обстановка очень милая и уютная, коллеги максимально дружелюбные, так как вы находитесь в группах примерно по 10 человек, все сближаются и ладят, что способствует продуктивной работе. На том же этаже находится фудкорт, где можно попить чая или кофе с коллегами и обсудить работу или другие интересующие темы. Само рабочее место представляет из себя большой стол, мониторы большого разрешения и хорошей цветопередачи и невероятного уровня удобства кресла. Работа в офисе как ни крути подразумевает долгое времяпровождение сидя за столом, что для большинства людей сложная задача. **SenamaSoft** позаботилась о том чтобы вам было приятно проводить свое время в стенах их офиса.

SenamaSoft занимается разработкой мобильноых и Web-приложений преймущественно на языках Javascrypt и Typescrypt. Из этого вытекает и следующий стек технологий.

* Next.js: Фреймворк для React, который упрощает разработку серверных приложений на React. Он предлагает такие возможности, как рендеринг на стороне сервера (SSR), генерация статических страниц (SSG), автоматическая маршрутизация и поддержка API.
* React: Библиотека для создания пользовательских интерфейсов. Она использует компонентный подход, позволяющий разработчикам создавать повторно используемые UI-компоненты. React также обеспечивает высокую производительность за счет использования виртуального DOM.
* Redux: Сторонний инструмент для управления состоянием приложений на JavaScript, особенно часто используется вместе с React. Он позволяет хранить все состояние приложения в одном месте и делает его предсказуемым и легко тестируемым.
* MongoDB: Документо-ориентированная NoSQL база данных, которая хранит данные в формате JSON-подобных документов. MongoDB популярна благодаря своей гибкости и масштабируемости.
* Redis: Высокопроизводительное хранилище данных в памяти, которое используется как кэш, база данных или брокер сообщений. Redis часто применяется для ускорения работы приложений за счет хранения часто используемых данных в памяти.
* MobX: Библиотека для управления состоянием, которая делает акцент на автоматической синхронизации состояния и интерфейса пользователя. В отличие от Redux, MobX более гибок и менее ограничивает разработчика, но при этом предоставляет мощные возможности для управления состоянием.
* React Testing Library: Набор инструментов для тестирования компонентов React. Основная цель этой библиотеки — обеспечивать простое и эффективное тестирование взаимодействия пользователя с интерфейсом.
* Jest: Фреймворк для тестирования JavaScript и TypeScript-кода. Jest имеет встроенные функции для мокинга, асинхронного тестирования и тестирования снимков, что делает его мощным инструментом для обеспечения качества кода.
* Node.js: Среда выполнения JavaScript на сервере. Node.js позволяет создавать масштабируемые сетевые приложения с использованием событийно-ориентированной модели и неблокирующего ввода-вывода.
* TypeScript: Язык программирования, который является строгой надстройкой над JavaScript, добавляющей статическую типизацию. TypeScript помогает обнаруживать ошибки на этапе компиляции, что делает код более надежным и поддерживаемым.
* JavaScript: Основной язык программирования для веб-разработки. JavaScript используется для создания интерактивных веб-страниц и клиентских приложений, а также может применяться на сервере с использованием Node.js.
* React Native: Фреймворк для создания мобильных приложений с использованием React. React Native позволяет писать код на JavaScript/TypeScript и использовать его для создания кроссплатформенных мобильных приложений для iOS и Android.
* Expo: Платформа и набор инструментов для разработки React Native приложений. Expo упрощает процесс разработки мобильных приложений, предлагая ряд готовых решений для работы с камерой, геолокацией, push-уведомлениями и другими возможностями устройств.
* Ant Design: Набор компонентов для React, предназначенный для создания приложений с хорошим пользовательским интерфейсом. Ant Design предоставляет готовые к использованию UI-компоненты, соответствующие современным стандартам дизайна.
* HTML: Язык разметки, используемый для создания структуры веб-страниц. HTML определяет элементы страницы, такие как заголовки, абзацы, ссылки, изображения и формы.
* CSS: Язык стилей, используемый для описания внешнего вида элементов, написанных на HTML. CSS определяет такие аспекты, как цвета, шрифты, отступы, расположение элементов и адаптивность веб-страницы.

#### **Обоснование выбора технологий**

Выбор технологий, таких как Next.js, MongoDB, Redux, TypeScript обусловлен их возможностями по обеспечению производительности, безопасности и масштабируемости решений. Эти инструменты позволяют эффективно разрабатывать высококачественные приложения, удовлетворяющие потребности клиентов и соответствующие современным требованиям рынка.

# 3. ТЕХНОЛОГИЧЕСКИЕ ПРОЦЕССЫ И ОПЕРАЦИИ

В SenamaSoft процесс разработки программного обеспечения имеет свои особенности и подходит с акцентом на эффективное использование современных технологий и методов. Основные технологические процессы и операции в компании следующие:

1. Анализ требований В SenamaSoft анализ требований включает в себя несколько ключевых этапов, направленных на тщательное понимание и документирование потребностей заказчика. Процесс начинается с сбора требований, который осуществляется через интервью, рабочие сессии и обсуждения с клиентами и пользователями. В компании также активно применяются методы прототипирования и создания пользовательских историй, что позволяет визуализировать требования и лучше их понять.

Далее происходит анализ требований, на котором выявляются противоречия и недочеты, а также разрабатываются предложения по их устранению. Важной частью является документирование требований, которое осуществляется в формате, удобном для всех участников процесса, будь то текстовые описания, сценарии использования или спецификации.

1. Проектирование программного обеспечения В SenamaSoft проектирование ПО осуществляется с акцентом на создание высококачественной архитектуры, которая обеспечит надежность и масштабируемость системы. Процесс начинается с определения архитектуры системы, в которой учитываются требования к производительности, безопасности и масштабируемости.

Основное внимание уделяется модульному проектированию, что позволяет разделить систему на независимые компоненты, упрощая их разработку и тестирование. В зависимости от сложности проекта используются как традиционные подходы, так и современные средства автоматизации проектирования, такие как UML-диаграммы и специализированные инструменты для проектирования архитектуры.

1. Программирование Процесс программирования в SenamaSoft включает в себя написание кода на высокоуровневых языках программирования, таких как JavaScript, TypeScript.

Важной частью процесса является рефакторинг кода, направленный на поддержание его чистоты и производительности.

1. Документация на программное обеспечение Документация в SenamaSoft разделяется на несколько типов:
   * Архитектурная/проектная документация: описывает высокоуровневую архитектуру системы, обоснование проектных решений и принципы работы системы.
   * Техническая документация: включает описание кода, интерфейсов и API. Используются автоматизированные генераторы документации для обеспечения актуальности и полноты информации.
   * Пользовательская документация: включает руководства и инструкции для конечных пользователей, а также часто задаваемые вопросы и рекомендации по решению проблем.
2. Тестирование программного обеспечения SenamaSoft применяет многоуровневое тестирование, чтобы обеспечить высокое качество ПО:
   * Модульное тестирование (юнит-тестирование): проводится для проверки отдельных компонентов системы. Автоматизированные тесты интегрируются в процесс CI/CD.
   * Интеграционное тестирование: проверяет взаимодействие между модулями и подсистемами.
   * Системное тестирование: осуществляется для проверки соответствия всей системы требованиям.
   * Альфа-тестирование: проводится внутренними тестировщиками и разработчиками на ранних этапах, чтобы выявить основные проблемы.
   * Бета-тестирование: включает выпуск версии продукта для ограниченной группы пользователей для получения обратной связи и исправления оставшихся ошибок.

**4. ПОСТАНОВКА ЗАДАЧИ**

Целью данного проекта является разработка системы регистрации и авторизации пользователей с использованием современных веб-технологий. Основной задачей системы является обеспечение безопасного доступа к ресурсам веб-приложения посредством механизма аутентификации и управления сессиями.

Система должна поддерживать следующие функциональные возможности:

* Регистрация новых пользователей.
* Вход (логин) пользователей в систему.
* Управление сессиями с использованием JWT (JSON Web Token).
* Защита маршрутов и ресурсов, требующих аутентификации.
* Безопасное хранение и обработка пользовательских данных, включая пароли и токены.

Для реализации поставленной задачи были выбраны следующие технологии:

* **Next.js** — фреймворк для React, который предоставляет возможности серверного рендеринга и создания API маршрутов, что позволяет объединить клиентскую и серверную части приложения в одном проекте.
* **MongoDB** — документо-ориентированная база данных, используемая для хранения данных пользователей, включая учетные записи и связанные сессии.
* **Redux** — библиотека для управления состоянием приложения, использующая централизованное хранилище, что упрощает работу с данными пользователя на клиентской стороне.
* **TypeScript** — статически типизированный язык программирования, который предоставляет возможность выявления ошибок на этапе компиляции, что повышает надежность и читаемость кода.
* **bcrypt** — библиотека для безопасного хэширования паролей, что обеспечивает защиту паролей от несанкционированного доступа.
* **jose** — библиотека для работы с JWT, включая их создание, верификацию и обновление, что необходимо для управления сессиями пользователей.

Для достижения целей проекта необходимо выполнить следующие задачи:

1. **Разработка клиентской части**:
   * Создание интерфейсов для регистрации и входа пользователей с использованием React и react-hook-form.
   * Валидация вводимых данных с использованием zod, что позволит предотвратить ввод некорректной информации и повысить безопасность.
2. **Разработка серверной части**:
   * Реализация API для регистрации пользователей, включая хэширование паролей с помощью bcrypt и сохранение учетных записей в базе данных MongoDB.
   * Реализация API для входа пользователей, включая генерацию JWT и управление сессиями.
   * Создание middleware для проверки JWT токенов при доступе к защищенным маршрутам, что обеспечит безопасность доступа к ресурсам.
3. **Обеспечение безопасности системы**:
   * Реализация механизма обновления JWT токенов с использованием рефреш-токенов для продления сессий пользователей.
   * Защита хранения токенов с использованием безопасных куки или других методов хранения, что минимизирует риски XSS атак.
4. **Тестирование и отладка системы**:
   * Проведение тестирования всех компонентов системы, включая обработку ошибок и сценарии несанкционированного доступа.
   * Отладка системы для обеспечения корректной работы и повышения пользовательского опыта.

В результате выполнения проекта должна быть разработана и внедрена система аутентификации и управления сессиями, которая будет удовлетворять всем требованиям безопасности и обеспечивать удобство использования для конечных пользователей. Система должна быть легко масштабируемой и готовой к интеграции с другими модулями веб-приложения.

**5. АНАЛИЗ НЕОБХОДИМОЙ АВТОМАТИЗИЦИИ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ**

В современном веб-разработке требования к безопасности и удобству использования становятся все более высокими. Особенно это касается систем, связанных с управлением учетными записями пользователей, аутентификацией и управлением сессиями. Вручную реализовывать и поддерживать такие системы является сложной и подверженной ошибкам задачей, что делает автоматизацию данного процесса необходимой.

Автоматизация процесса регистрации, аутентификации и управления сессиями позволяет:

* **Повысить безопасность**: автоматические проверки ввода данных, хэширование паролей и управление сессиями минимизируют человеческие ошибки и обеспечивают защиту данных пользователей.
* **Улучшить пользовательский опыт**: интуитивно понятные интерфейсы и быстрая обработка запросов снижают нагрузку на пользователей и повышают их удовлетворенность.
* **Обеспечить масштабируемость и интеграцию**: автоматизированные системы легче масштабировать и интегрировать с другими модулями и сервисами, что позволяет развивать проект без серьезных изменений в базовой архитектуре.

Для решения задачи автоматизации регистрации и авторизации пользователей существует множество технологий и инструментов, которые можно использовать в зависимости от требований и условий проекта. Рассмотрим несколько популярных решений, которые могли бы быть применены в рамках данного проекта.

**NextAuth.js** — это библиотека для аутентификации в приложениях Next.js, которая предоставляет готовые решения для интеграции с различными провайдерами OAuth, а также поддержку JWT для работы с сессиями. Основные преимущества NextAuth.js:

* Простая интеграция с Next.js проектами.
* Поддержка множества провайдеров OAuth (например, Google, Facebook, GitHub).
* Автоматическое управление сессиями и безопасное хранение JWT.

**Недостатки**:

* Ограниченная гибкость в настройке процессов регистрации и авторизации.
* Сложности с интеграцией специфических требований безопасности.

**Firebase Authentication** — это сервис от Google, предоставляющий готовое решение для аутентификации пользователей с поддержкой различных методов входа (электронная почта, телефон, OAuth-провайдеры). Основные преимущества:

* Быстрая интеграция и готовая инфраструктура для управления пользователями.
* Поддержка мультифакторной аутентификации.
* Высокая надежность и безопасность благодаря инфраструктуре Google.

**Недостатки**:

* Зависимость от стороннего сервиса, что может вызывать сложности с контролем над данными.
* Ограниченные возможности кастомизации, особенно в части управления сессиями и специфических требований безопасности.

**Passport.js** — это модуль для Node.js, обеспечивающий гибкую систему аутентификации с поддержкой более 500 стратегий, включая JWT. Преимущества Passport.js:

* Высокая гибкость и кастомизация под конкретные нужды проекта.
* Поддержка множества стратегий аутентификации, включая локальную аутентификацию, OAuth и другие.
* Возможность интеграции с любыми фреймворками на основе Node.js.

**Недостатки**:

* Сложность интеграции и настройки по сравнению с NextAuth.js и Firebase Authentication.
* Требует больше времени на разработку и поддержку.

Реализация системы регистрации и авторизации с нуля с использованием JWT, bcrypt для хэширования паролей, и собственной системы middleware для защиты маршрутов является еще одним подходом. Преимущества такого решения:

* Полный контроль над процессом аутентификации и управления сессиями.
* Гибкость в реализации любых специфических требований безопасности и логики работы.

**Недостатки**:

* Требует значительных усилий на разработку, тестирование и поддержку.
* Потенциальные риски при неправильной реализации безопасности.

Для данного проекта был выбран подход с использованием Next.js, MongoDB и JWT, который включает создание собственной системы регистрации и авторизации с нуля. Такой выбор обоснован следующими факторами:

* **Контроль над системой**: создание собственной системы позволяет гибко настраивать процессы аутентификации, регистрации и управления сессиями в соответствии с требованиями проекта.
* **Безопасность**: использование библиотеки bcrypt для хэширования паролей и jose для управления JWT токенами позволяет обеспечить высокий уровень защиты данных.
* **Интеграция**: выбранный стек технологий (Next.js, MongoDB) хорошо интегрируется между собой и позволяет создавать легко масштабируемые и поддерживаемые решения.

Таким образом, выбранный подход обеспечивает баланс между гибкостью, безопасностью и возможностью дальнейшего развития проекта.

**6. ВВЕДЕНИЕ**

В современном мире, где цифровизация охватывает все сферы жизни, безопасность и удобство использования веб-приложений становятся приоритетными задачами для разработчиков. Пользовательские данные требуют надежной защиты, а сам процесс взаимодействия с приложением должен быть максимально простым и понятным. Это особенно важно для систем, в которых происходит регистрация и авторизация пользователей, так как они являются основными точками входа и источниками данных в большинстве веб-приложений.

С развитием технологий и ростом числа киберугроз разработчики сталкиваются с необходимостью внедрения современных методов защиты и оптимизации пользовательского опыта. В связи с этим, создание эффективной системы регистрации и авторизации с использованием проверенных технологий и методов, таких как JWT (JSON Web Token), bcrypt, и других, является важной задачей для обеспечения безопасности и масштабируемости приложений.

Целью данного проекта является разработка системы регистрации и авторизации пользователей на основе стек технологий Next.js, MongoDB и JWT, которая обеспечит высокий уровень безопасности, гибкость в настройках и удобство использования. Основные задачи, поставленные в рамках проекта, включают:

* Анализ существующих решений для аутентификации и их сравнение.
* Разработка системы регистрации и авторизации пользователей с использованием JWT для управления сессиями.
* Обеспечение безопасности хранения паролей с использованием bcrypt.
* Интеграция созданной системы с MongoDB для управления пользовательскими данными.
* Создание удобного и интуитивно понятного интерфейса для пользователей с использованием библиотеки Ant Design.
* Оценка и тестирование системы для выявления и устранения потенциальных уязвимостей.

Таким образом, данная работа направлена на разработку надежной, гибкой и удобной в использовании системы регистрации и авторизации пользователей, которая соответствует современным требованиям безопасности и удобства.

**7. СТРУКТУРА ПРОГРАММЫ. ПРОЕКТИРОВАНИЕ**

Проект представляет собой веб-приложение, основанное на стек технологий **Next.js**, **Node.js** и **MongoDB**. Основная задача приложения — реализовать механизм регистрации и авторизации пользователей с использованием JWT-токенов (JSON Web Tokens).

Архитектура проекта построена на принципах модульности и разделения ответственности. Каждый компонент системы отвечает за свою четко определенную задачу. Ниже приведены основные модули и их описание:

* **Frontend**: Реализован на Next.js с использованием App Router. Это одностраничное приложение (SPA), которое взаимодействует с сервером через REST API. В качестве UI-библиотеки используется Ant Design.
* **Backend**: Реализован на Node.js. Основные задачи backend — обработка запросов, управление базой данных MongoDB и генерация JWT-токенов.
* **База данных**: Используется MongoDB для хранения данных пользователей. Каждый пользователь имеет уникальный идентификатор, а также информацию о токенах.

Frontend-часть проекта реализована с использованием **Next.js**. Данный фреймворк был выбран благодаря его гибкости и поддержке server-side rendering (SSR), что положительно сказывается на SEO и производительности.

В проекте используется App Router для организации маршрутов. Это позволяет эффективно разделять код и упрощать навигацию по приложению. Кроме того, для визуальной составляющей используется **Ant Design**, что обеспечивает единообразие и профессиональный вид интерфейса.

В данном проекте основное внимание было уделено построению надежной и масштабируемой системы регистрации и авторизации пользователей с использованием JSON Web Tokens (JWT). Проект реализован на платформе Next.js с использованием MongoDB в качестве базы данных и Node.js для серверной логики. В этой главе будет рассмотрена структура программы и основные проектные решения, которые были приняты при разработке.

**Проект имеет следующую структуру**

├─ src  
│ ├─ app  
│ │ ├─ (protected)  
│ │ │ ├─ home  
│ │ │ │ ├─ page.module.css  
│ │ │ │ └─ page.tsx  
│ │ │ ├─ layout.module.css  
│ │ │ └─layout.tsx  
│ │ ├─ api  
│ │ │ ├─ auth  
│ │ │ │ ├─ login  
│ │ │ │ │ └─ route.ts  
│ │ │ │ ├─ logout  
│ │ │ │ │ └─ route.ts  
│ │ │ │ ├─ refresh  
│ │ │ │ │ └─ route.ts  
│ │ │ │ └─ register  
│ │ │ │ └─ route.ts  
│ │ │ └─ users  
│ │ │ ├─ get  
│ │ │ │ └─ [id]  
│ │ │ │ └─ route.ts  
│ │ │ └─ route.ts  
│ │ ├─ auth  
│ │ │ ├─ activate  
│ │ │ │ ├─ page.module.css  
│ │ │ │ └─ page.tsx  
│ │ │ ├─ page.module.css  
│ │ │ └─ page.tsx  
│ │ ├─ global.css  
│ │ ├─ layout.tsx  
│ │ ├─ \_app.tsx  
│ │ └─ \_document.tsx  
│ ├─ lib  
│ │ ├─ config.ts  
│ │ ├─ middlewares  
│ │ │ └─ check-login-middleware.ts  
│ │ ├─ mongo  
│ │ │ ├─ connect.ts  
│ │ │ └─ models  
│ │ │ ├─ chat-model.ts  
│ │ │ ├─ token-model.ts  
│ │ │ └─ user-model.ts  
│ │ ├─ notifications  
│ │ │ ├─ show-error-notification.ts  
│ │ │ ├─ show-success-notification.ts  
│ │ │ └─ toastify-provider.tsx  
│ │ └─ services  
│ │ ├─ auth-service.ts  
│ │ └─ user-service.ts  
│ ├─ middleware.ts  
│ ├─ modules  
│ │ ├─ auth  
│ │ │ ├─ auth.module.css  
│ │ │ ├─ auth.tsx  
│ │ │ ├─ sing-in-form  
│ │ │ │ ├─ sing-in-form.module.css  
│ │ │ │ └─ sing-in-form.tsx  
│ │ │ ├─ sing-up-form  
│ │ │ │ ├─ sing-up-form.module.css  
│ │ │ │ └─ sing-up-form.tsx  
│ │ │ └─ switch-auth  
│ │ │ ├─ switch-auth.module.css  
│ │ │ └─ switch-auth.tsx  
│ │ ├─ global  
│ │ │ ├─ auth-provider  
│ │ │ │ └─ auth-provider.tsx  
│ │ │ ├─ header  
│ │ │ │ ├─ header.module.css  
│ │ │ │ └─ header.tsx  
│ │ │ └─ loading-provider.tsx  
│ │ │ ├─ loading-provider.module.css  
│ │ │ └─ loading-provider.tsx  
│ │ └─ shared  
│ │ ├─ logo  
│ │ │ ├─ logo.module.css  
│ │ │ └─ logo.tsx  
│ │ └─ navigation  
│ │ ├─ navigation.module.css  
│ │ └─ navigation.tsx  
│ ├─ store  
│ │ ├─ reducers  
│ │ │ ├─ auth-reducer.ts  
│ │ │ └─ loading-reducer.ts  
│ │ ├─ store-provider.tsx  
│ │ └─ store.ts  
│ ├─ types  
│ │ └─ global.d.ts  
│ ├─ UI  
│ │ ├─ button  
│ │ │ ├─ button.module.css  
│ │ │ └─ button.tsx  
│ │ ├─ content  
│ │ │ ├─ content.module.css  
│ │ │ └─ content.ts│ │ ├─ icon-button  
│ │ │ ├─ icon-button.module.css  
│ │ │ └─ icon-button.tsx  
│ │ ├─ image-input  
│ │ │ ├─ image-input.module.css  
│ │ │ └─ image-input.tsx  
│ │ ├─ input  
│ │ │ ├─ input.module.css  
│ │ │ └─ input.tsx  
│ │ ├─ modal  
│ │ │ ├─ modal.module.css  
│ │ │ └─ modal.tsx  
│ │ ├─ text  
│ │ │ ├─ text.module.css  
│ │ │ └─ text.tsx  
│ │ ├─ title  
│ │ │ ├─ title.module.css  
│ │ │ └─ title.tsx  
│ │ └─ types.ts  
│ └─ utils  
│ └─ cn.ts  
└─ tsconfig.json

Модуль аутентификации и авторизации включает в себя обработку следующих операций:

* **Регистрация** — создание нового пользователя в системе.
* **Авторизация** — проверка учетных данных пользователя и выдача JWT-токенов.
* **Обновление токенов** — предоставление нового access-токена при помощи refresh-токена.
* **Выход из системы** — аннулирование токенов.

**Пример кода для регистрации пользователя:**

"use server"

import bcrypt from "bcrypt"

import { SignJWT } from "jose"

import { cookies } from "next/headers"

import { NextRequest, NextResponse } from "next/server"

import { v4 } from "uuid"

import connect from "@/lib/mongo/connect"

import { Token } from "@/lib/mongo/models/token-model"

import { User } from "@/lib/mongo/models/user-model"

export async function POST(req: NextRequest) {

const { username, email, password } = await req.json()

const refreshTokenCookie = cookies().get("refreshToken")?.value

if (refreshTokenCookie) {

return NextResponse.json({ message: "User already login" }, { status: 400 })

}

await connect()

const candidate = await User.findOne({ email })

if (candidate) {

return NextResponse.json({ message: "User already exists" }, { status: 400 })

}

const hashPassword = await bcrypt.hash(password, 3)

const activationLink = v4()

const user = await User.create({

username,

email,

password: hashPassword,

activationLink,

})

const userDto = {

id: user.\_id,

username: user.username,

email: user.email,

isActivated: user.isActivated,

}

const encoder = new TextEncoder()

const jwtAccessKey = encoder.encode(process.env.JWT\_ACCESS)

const jwtRefreshKey = encoder.encode(process.env.JWT\_REFRESH)

if (!jwtAccessKey || !jwtRefreshKey) {

return NextResponse.json({ status: 500 })

}

const accessToken = await new SignJWT(userDto)

.setProtectedHeader({ alg: "HS256" })

.setExpirationTime("15m")

.sign(jwtAccessKey)

const refreshToken = await new SignJWT(userDto)

.setProtectedHeader({ alg: "HS256" })

.setExpirationTime("60d")

.sign(jwtRefreshKey)

let refreshTokenData = await Token.findOne({ id: user.\_id })

if (refreshTokenData) {

refreshTokenData.refreshToken = refreshToken

await refreshTokenData.save()

} else {

await Token.create({ user, refreshToken })

}

cookies().set("refreshToken", refreshToken, { httpOnly: true, maxAge: 30 \* 24 \* 60 \* 60 \* 1000 })

return NextResponse.json({ accessToken, refreshToken, user: userDto })

}

В этом коде происходит подключение к базе данных, хеширование пароля пользователя и создание нового документа в коллекции User. После успешного создания пользователя генерируется JWT-токен, который возвращается клиенту.

Для защиты маршрутов, требующих авторизации, используются мидлвары. Они проверяют наличие и валидность access-токена и, в случае успеха, разрешают доступ к защищенным ресурсам.

**Пример мидлвары для защиты маршрутов**

"use server"

import { jwtVerify } from "jose"

import { headers } from "next/headers"

import { NextRequest, NextResponse } from "next/server"

import { HOST, PORT } from "../config"

import { AuthService } from "../services/auth-service"

const protectedRoutes = ["/api/users", "/api/audios"]

const protectedPages = ["/home", "/audio", "/eva"]

export const checkLoginMiddleware = async (req: NextRequest) => {

if (protectedRoutes.some((route) => req.nextUrl.pathname.startsWith(route))) {

const accessToken = headers().get("Authentication")?.split(" ")[1]

if (!accessToken) {

return NextResponse.redirect(new URL(`http://${HOST}:${PORT}/auth`))

}

const isValidAccessToken = await jwtVerify(

accessToken,

new TextEncoder().encode(process.env.JWT\_ACCESS!),

)

if (!isValidAccessToken) {

const res = await AuthService.refresh()

if (!res.data || !res.data.accessToken) {

return NextResponse.redirect(new URL(`http://${HOST}:${PORT}/auth`))

}

headers().set("Authentication", `Bearer ${accessToken}`)

}

}

}

Этот код представляет собой middleware-функцию checkLoginMiddleware, которая используется для проверки наличия и валидности JWT-токена при доступе к защищённым маршрутам и страницам.

Для хранения информации о пользователях и их учетных данных используется MongoDB. Модель пользователя определена в файле /src/models/User.js и включает поля для имени пользователя и хешированного пароля.

**Пример модели пользователя**

import { model, models, Schema } from "mongoose"

const UserSchema = new Schema({

username: { type: String, require: true },

email: { type: String, unique: true, require: true },

password: { type: String, require: true },

isActivated: { type: Boolean, default: false },

activationLink: { type: String },

})

export const User = models.User || model("User", UserSchema)

Эта модель определяет структуру документов в коллекции User и обеспечивает уникальность пользователей.

### **Проектирование интерфейса пользователя**

Для разработки интерфейса пользователя используется библиотека Ant Design, которая предоставляет готовые компоненты для построения форм, кнопок и других элементов управления. Это позволяет быстро создавать удобные и интуитивно понятные интерфейсы.

Пример кода для формы регистрации:

"use client"  
  
import { Controller, SubmitHandler, useForm } from "react-hook-form"  
import { useDispatch, useSelector } from "react-redux"  
import \* as z from "zod"  
  
import { showErrorNotification } from "@/lib/notifications/show-error-notification"  
import { showSuccessNotification } from "@/lib/notifications/show-success-notification"  
import { AuthService } from "@/lib/services/auth-service"  
import { setAuthState } from "@/store/reducers/auth-reducer"  
import { setIsLoading } from "@/store/reducers/loading-reducer"  
import { RootState } from "@/store/store"  
import { Button } from "@/UI/button/button"  
import { Content } from "@/UI/content/content"  
import { Input } from "@/UI/input/input"  
import { Text } from "@/UI/text/text"  
import { Title } from "@/UI/title/title"  
import { zodResolver } from "@hookform/resolvers/zod"  
  
import styles from "./sing-up-form.module.css"  
import { useRouter } from "next/navigation"  
  
interface Inputs {  
 username: string  
 email: string  
 password: string  
 repeatPassword: string  
}  
  
const schema = z  
 .object({  
 username: z.string().nonempty({ message: "Username is required" }),  
 email: z  
 .string()  
 .email({ message: "Email is not valid" })  
 .nonempty({ message: "Email is required" }),  
 password: z  
 .string()  
 .min(6, { message: "Password must be at least 6 characters" })  
 .max(12, { message: "Password must be less than 12 characters" })  
 .nonempty({ message: "Password is required" }),  
 repeatPassword: z.string().nonempty({ message: "Repeat Password is required" }),  
 })  
 .refine((data) => data.password === data.repeatPassword, {  
 message: "Passwords don't match",  
 path: ["repeatPassword"],  
 })  
  
export const SingUpForm = () => {  
 const {  
 handleSubmit,  
 control,  
 formState: { errors },  
 } = useForm<Inputs>({  
 resolver: zodResolver(schema),  
 })  
  
 const dispatch = useDispatch()  
 const router = useRouter()  
  
 const onSingUp: SubmitHandler<Inputs> = async (data) => {  
 dispatch(setIsLoading(true))  
 const res = await AuthService.registerUser(data.username, data.email, data.password)  
 if (res.data) {  
 dispatch(setAuthState(res.data))  
 localStorage.setItem("accessToken", JSON.stringify(res.data.accessToken))  
 router.push("/home")  
 showSuccessNotification(`Welcome, ${res.data.user.username}!`)  
 } else {  
 showErrorNotification(res.message)  
 }  
 dispatch(setIsLoading(false))  
 }  
  
 return (  
 <Content  
 padding="32px 24px"  
 maxWidth="320px"  
 width="100%"  
 >  
 <form  
 className={styles.container}  
 onSubmit={handleSubmit(onSingUp)}  
 >  
 <Title  
 type="h1"  
 textAlign="center"  
 margin="16px 0 20px"  
 text="Sign Up"  
 />  
 <Controller  
 name="username"  
 defaultValue=""  
 control={control}  
 render={({ field }) => (  
 <Input  
 placeholder="Enter your Username"  
 {...field}  
 />  
 )}  
 />  
 {errors.username && (  
 <Text  
 type="error"  
 text={errors.username.message!}  
 />  
 )}  
 <Controller  
 name="email"  
 defaultValue=""  
 control={control}  
 render={({ field }) => (  
 <Input  
 placeholder="Enter your Email"  
 {...field}  
 />  
 )}  
 />  
 {errors.email && (  
 <Text  
 type="error"  
 text={errors.email.message!}  
 />  
 )}  
 <Controller  
 name="password"  
 defaultValue=""  
 control={control}  
 render={({ field }) => (  
 <Input  
 placeholder="Enter your Password"  
 {...field}  
 />  
 )}  
 />  
 {errors.password && (  
 <Text  
 type="error"  
 text={errors.password.message!}  
 />  
 )}  
 <Controller  
 name="repeatPassword"  
 defaultValue=""  
 control={control}  
 render={({ field }) => (  
 <Input  
 placeholder="Repeat your Password"  
 {...field}  
 />  
 )}  
 />  
 {errors.repeatPassword && (  
 <Text  
 type="error"  
 text={errors.repeatPassword.message!}  
 />  
 )}  
 <Button  
 type="white"  
 btnType="submit"  
 text="Sing Up"  
 width="100%"  
 />  
 </form>  
 </Content>  
 )  
}

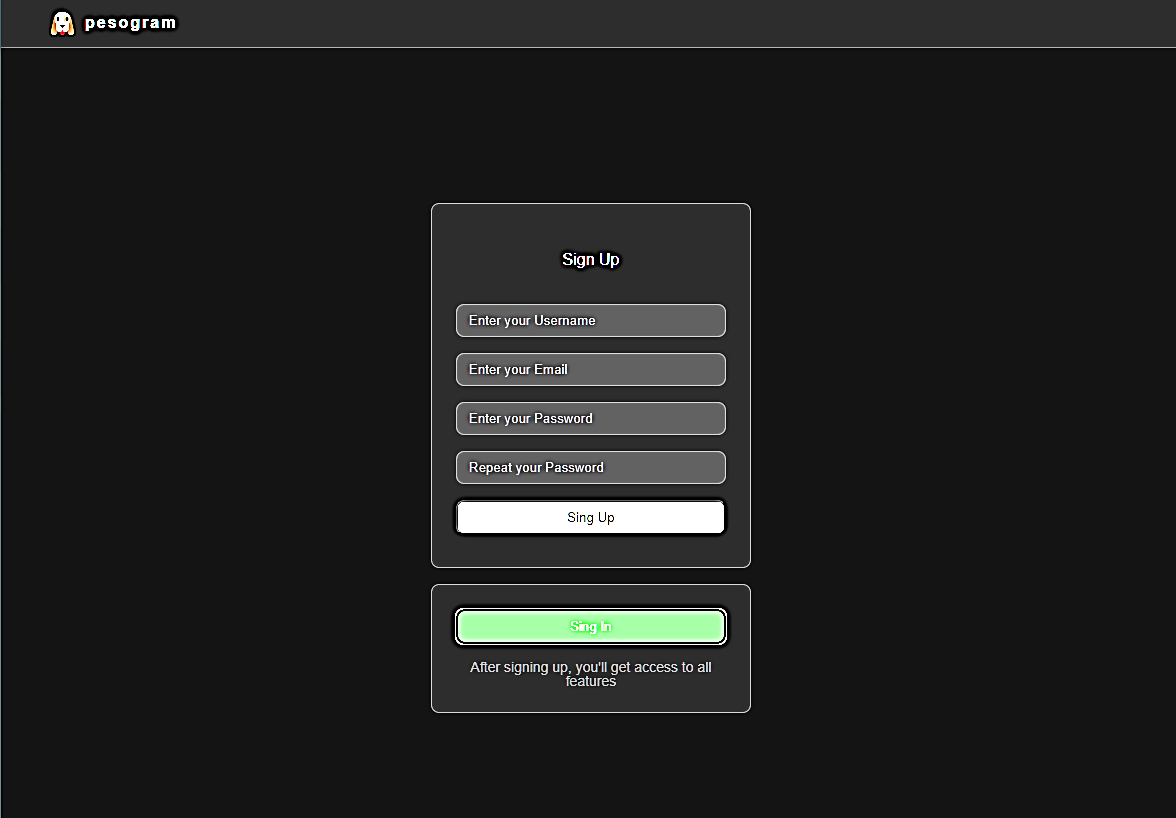
Эта форма использует компоненты Ant Design для создания интерфейса регистрации нового пользователя.

**8. ТЕСТИРОВАНИЕ**

Тестирование системы регистрации и авторизации пользователей является ключевым этапом для обеспечения ее надежности и безопасности. В данной главе будут описаны методы и подходы, использованные для проверки работоспособности основных функций системы, а также стратегии для выявления и устранения потенциальных проблем.

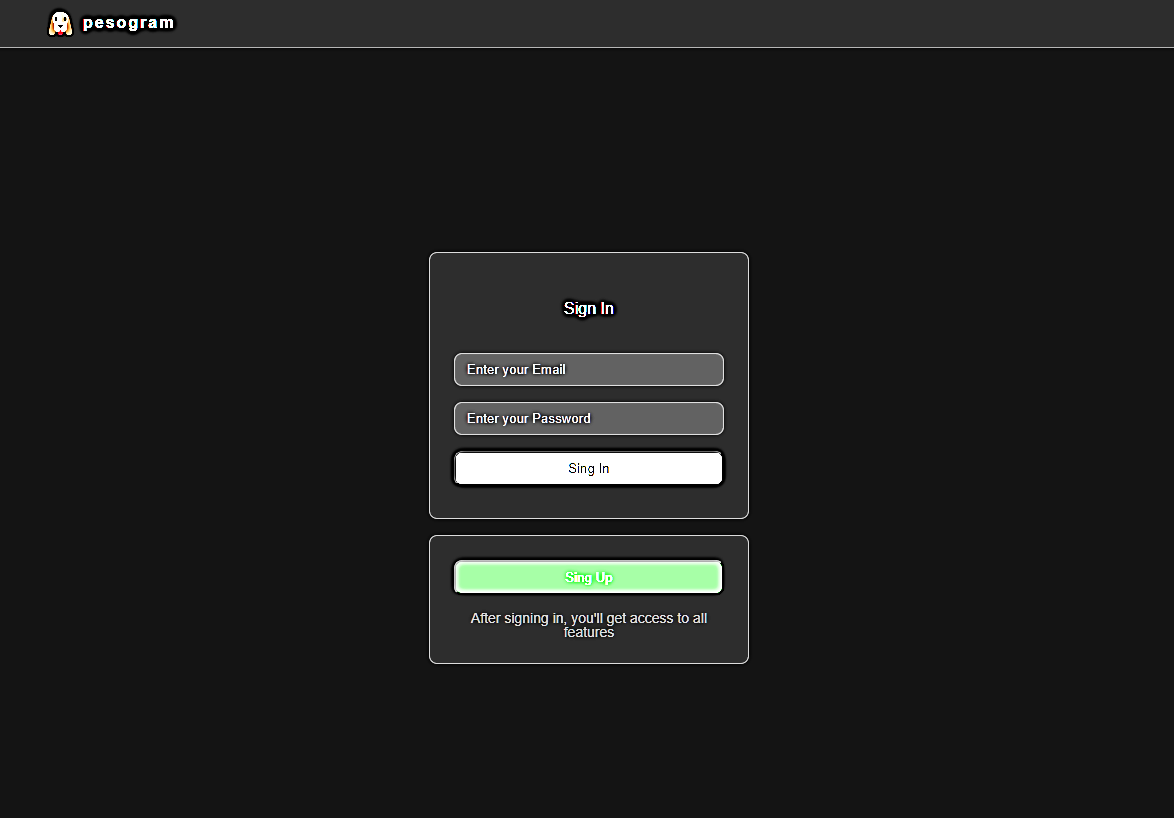
**Протестируем основные потоки исполнения:**

Пользователь не зарегистрирован на сайте и хочет создать свою учетную запись:



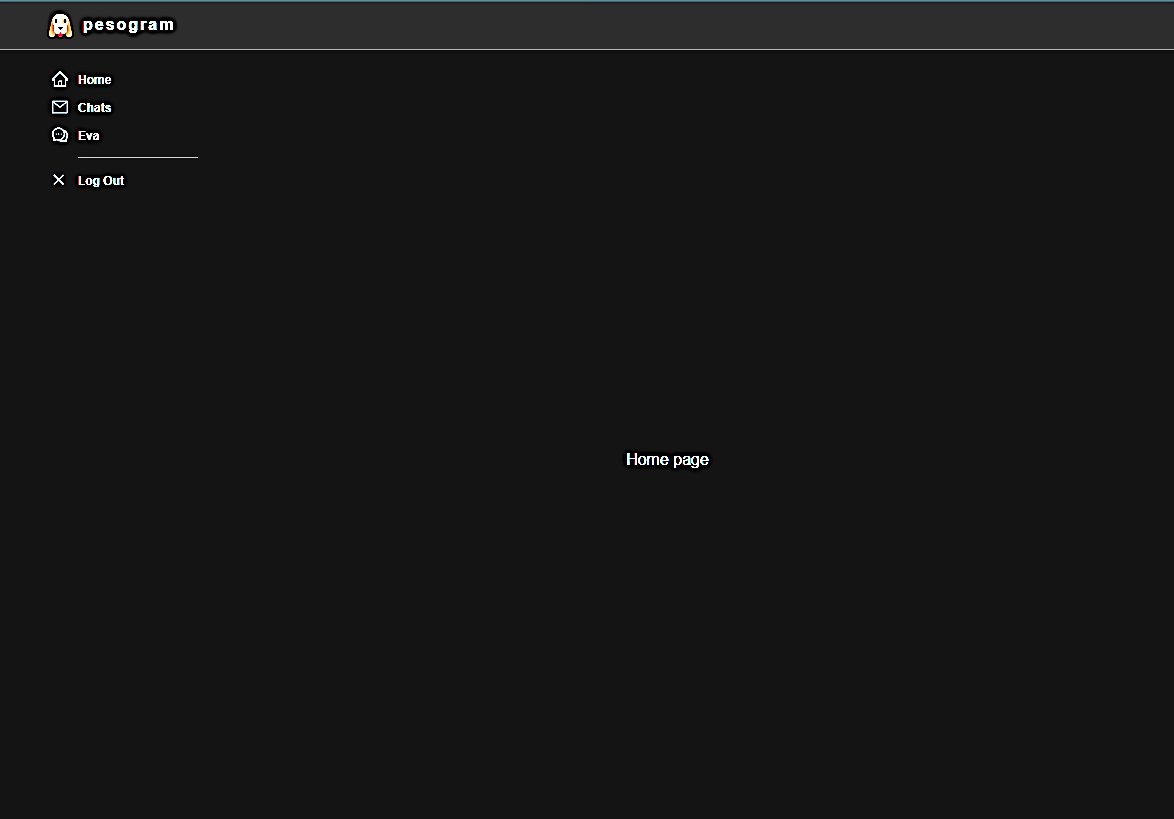
**Ожидаемый результат:** Пользователь получает учетную запись и входит в систему.

Пользователь зарегистрирован на сайте и хочет авторизироваться:



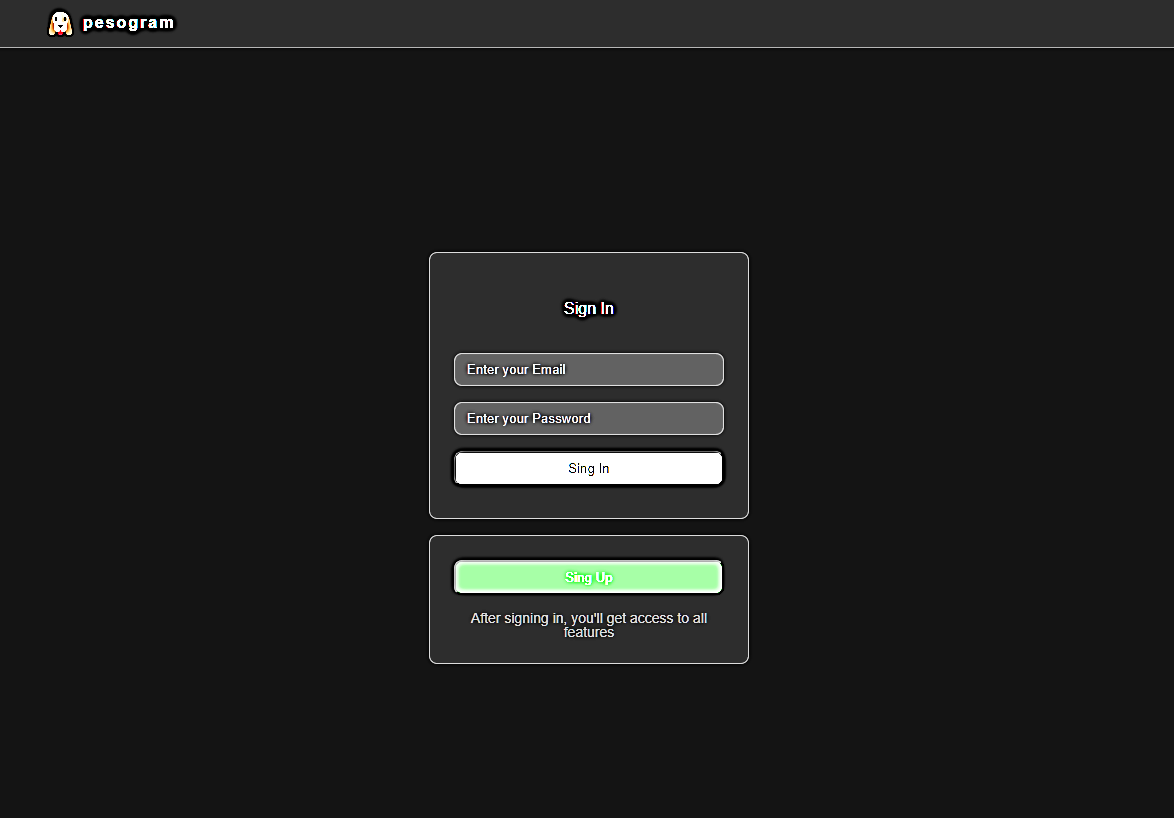
**Ожидаемый результат:** Пользователь входит в систему под существующей учетной записи.

Пользователь авторизован и хочет покинуть систему:



**Ожидаемый результат:** Пользователь выходит из учетной записи.

Пользователь переходит по защищенному url не авторизовавщись:



**Ожидаемый результат:** Пользователь возвращается на страницу регистрации/авторизации.

**ЗАКЛЮЧЕНИЕ**

Практика в компании **SenamaSoft** стала для меня бесценным опытом, который значительно углубил мои знания и навыки в области веб-разработки. Работая над реальными проектами и взаимодействуя с командой профессионалов, я смог на практике применить теоретические знания, полученные в процессе обучения, и узнать множество новых аспектов, которые остаются за рамками учебных программ.

В ходе прохождения практики я познакомился с полным циклом разработки программного обеспечения, начиная с анализа требований и проектирования, заканчивая внедрением и поддержкой решений. Работа с современным стеком технологий, включающим Next.js, MongoDB, Node.js, JWT, и другие инструменты, позволила мне не только освоить передовые подходы к созданию веб-приложений, но и понять, как правильно организовать процесс разработки для достижения высоких стандартов качества и безопасности.

**SenamaSoft** оказалась идеальным местом для развития моих профессиональных навыков. Компания не только предоставляет своим сотрудникам доступ к передовым технологиям, но и создает комфортную и дружелюбную рабочую атмосферу, что способствует максимальной продуктивности и росту. Я глубоко признателен компании за предоставленную возможность, поддержку и наставничество, которые я получил за время своей практики. Этот опыт стал важной вехой в моем карьерном пути, и я уверен, что знания и навыки, приобретенные в **SenamaSoft**, станут надежной основой для дальнейшего профессионального развития.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1) Официальный сайт SenamaSoft https://senamasoft.com (Дата обращения: 06.08.2024).

2) Официальный сайт NextJS https://nextjs.org (Дата обращения: 06.08.2024).