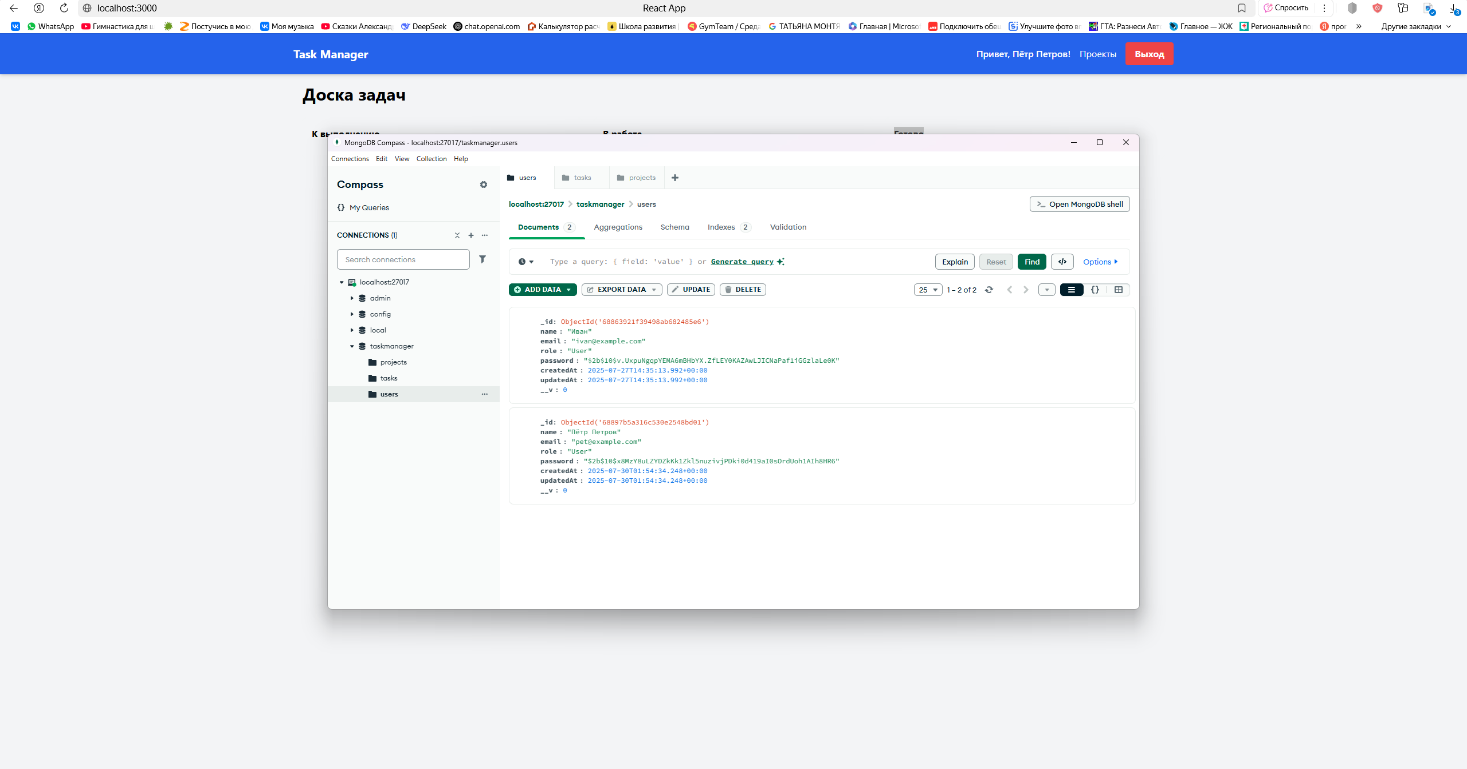
**Введение**

**Актуальность темы.** В современной цифровой экономике эффективное управление проектами является ключевым фактором успеха для компаний любого размера. С ростом популярности удаленной и гибридной работы, потребность в гибких, доступных и функциональных инструментах для совместной работы и отслеживания задач резко возросла. Существующие на рынке решения, такие как Jira, Asana или Trello, часто либо избыточны по функциональности и сложны для небольших команд, либо имеют высокую стоимость. Это создает потребность в разработке легковесных, интуитивно понятных и экономически эффективных веб-приложений для управления задачами, которые могут быть адаптированы под конкретные нужды команды.

**Проблема исследования.** Проблема заключается в поиске оптимального баланса между функциональностью, производительностью и сложностью разработки при создании современного full-stack веб-приложения для управления проектами. Необходимо выбрать подходящий стек технологий, спроектировать масштабируемую архитектуру и реализовать интуитивно понятный пользовательский интерфейс, отвечающий основным потребностям пользователей.

**Цель проекта:** Спроектировать, разработать и развернуть полнофункциональное веб-приложение для управления задачами и проектами, используя современный стек технологий MERN (MongoDB, Express.js, React, Node.js).

**Задачи проекта:**

1. Провести анализ существующих решений и методологий управления проектами для определения ключевых требований к приложению.
2. Выбрать и обосновать выбор стека технологий для full-stack разработки.
3. Спроектировать архитектуру приложения, включая схему базы данных и дизайн API.
4. Разработать серверную часть (backend) приложения, отвечающую за бизнес-логику и обработку данных.
5. Разработать клиентскую часть (frontend) с интерактивным и отзывчивым пользовательским интерфейсом.
6. Реализовать систему регистрации, аутентификации и авторизации пользователей.
7. Провести тестирование основных функциональных возможностей приложения.
8. Подготовить приложение к развертыванию на облачной платформе.

**Инструменты и технологии:**

* **Backend:** Node.js, Express.js
* **Frontend:** React, Redux Toolkit, Tailwind CSS
* **База данных:** MongoDB с использованием Mongoose
* **Аутентификация:** JSON Web Tokens (JWT)
* **Контроль версий:** Git, GitHub
* **Тестирование:** Jest, Supertest, React Testing Library
* **Развертывание:** Docker, Heroku

**Состав команды.** Данный дипломный проект был выполнен одним разработчиком, который совмещал роли: Full-stack разработчика, UI/UX дизайнера, инженера по тестированию и DevOps-инженера.

**Глава 1. Теоретические основы разработки веб-приложений для управления проектами**

**1.1. Анализ современных методологий управления проектами**

Для создания эффективного инструмента управления задачами необходимо понимать принципы, лежащие в основе современных методологий управления проектами.

* **Waterfall (Водопадная модель):** Классический, последовательный подход, где каждый этап (сбор требований, проектирование, реализация, тестирование) выполняется строго один за другим. Подходит для проектов с четко определенными и неизменными требованиями.
* **Agile (Гибкая методология):** Итеративный подход, ориентированный на гибкость, постоянное взаимодействие с заказчиком и быструю поставку работающего продукта. Работа делится на короткие циклы (спринты).
* **Scrum (Скрам):** Один из самых популярных фреймворков в рамках Agile. Определяет конкретные роли (Владелец продукта, Скрам-мастер, Команда разработки), события (Спринт, Планирование спринта, Ежедневный скрам) и артефакты (Бэклог продукта, Бэклог спринта).
* **Kanban (Канбан):** Метод, фокусирующийся на визуализации рабочего процесса. Задачи проходят через стадии (например, "К выполнению", "В процессе", "Готово") на Kanban-доске. Основная цель — ограничение незавершенной работы (WIP) и оптимизация потока задач.

**Вывод:** Разрабатываемое приложение должно быть достаточно гибким, чтобы поддерживать принципы различных методологий. Наиболее универсальным и наглядным является подход с использованием Kanban-доски, который будет взят за основу визуализации задач в проекте.

**1.2. Обзор архитектурных паттернов для веб-приложений**

* **Монолитная архитектура:** Приложение разрабатывается как единое целое. Все компоненты (UI, бизнес-логика, доступ к данным) тесно связаны и развертываются вместе.
  + *Преимущества:* Простота разработки и развертывания на начальных этапах.
  + *Недостатки:* Сложность масштабирования и обновления отдельных частей, технологическая зависимость.
* **Микросервисная архитектура:** Приложение состоит из набора небольших, независимо развертываемых сервисов. Каждый сервис отвечает за свою бизнес-задачу и может быть написан на своем стеке технологий.
  + *Преимущества:* Гибкость, масштабируемость, надежность.
  + *Недостатки:* Сложность управления и взаимодействия между сервисами.
* **Клиент-серверная архитектура с REST API:** Четкое разделение на "толстый" клиент (например, одностраничное приложение - SPA), отвечающий за пользовательский интерфейс, и сервер (backend), предоставляющий данные через стандартизированный интерфейс (API).

**Вывод:** Для данного проекта выбрана клиент-серверная архитектура с REST API. Backend на Node.js будет представлять собой модульный монолит, а frontend на React — независимое SPA. Этот подход обеспечивает хорошее разделение ответственности, гибкость в разработке и является отраслевым стандартом для современных веб-приложений.

**1.3. Сравнительный анализ технологий для full-stack разработки на примере стека MERN**

**MERN** — это набор технологий, использующих JavaScript на всех уровнях разработки.

* **MongoDB (База данных):** Документо-ориентированная NoSQL база данных. Ее гибкая схема идеально подходит для проектов на ранних стадиях и итеративной разработки, так как позволяет легко изменять структуру данных.
* **Express.js (Веб-фреймворк для backend):** Минималистичный и гибкий фреймворк для Node.js, ставший де-факто стандартом для создания API. Предоставляет надежный набор функций для веб- и мобильных приложений.
* **React (Библиотека для frontend):** Популярная JavaScript-библиотека для создания пользовательских интерфейсов. Ее компонентный подход позволяет строить сложные и переиспользуемые UI-элементы.
* **Node.js (Среда выполнения):** Среда выполнения JavaScript, построенная на движке V8 от Google Chrome. Ее неблокирующая, управляемая событиями архитектура делает ее идеальной для создания быстрых и масштабируемых сетевых приложений.

**Обоснование выбора:** Стек MERN позволяет использовать один язык программирования (JavaScript/TypeScript) для всей разработки, что упрощает процесс и сокращает время на переключение контекста. Он обладает огромным сообществом, большим количеством библиотек и инструментов, что ускоряет разработку и решение возникающих проблем.

**1.4. Принципы обеспечения безопасности в веб-приложениях**

Безопасность является критически важным аспектом любого веб-приложения.

* **Аутентификация и авторизация:** Аутентификация — это процесс проверки личности пользователя (кто вы?). Авторизация — это процесс предоставления доступа к определенным ресурсам (что вам разрешено делать?). В проекте используется аутентификация на основе **JSON Web Tokens (JWT)**. JWT — это компактный, безопасный способ передачи информации между сторонами в виде JSON-объекта.
* **Хеширование паролей:** Пароли пользователей никогда не должны храниться в открытом виде. Используется криптографический алгоритм **bcrypt** для создания необратимого хеша пароля.
* **Защита маршрутов (API endpoints):** Реализовано middleware, которое проверяет наличие и валидность JWT токена в каждом запросе к защищенным ресурсам.
* **Предотвращение распространенных уязвимостей:**
  + **Cross-Site Scripting (XSS):** React по умолчанию экранирует данные, вставляемые в JSX, что обеспечивает базовую защиту.
  + **Cross-Site Request Forgery (CSRF):** Использование JWT в заголовках авторизации вместо cookie помогает снизить риск CSRF-атак.
  + **Настройка CORS (Cross-Origin Resource Sharing):** На сервере настроен механизм CORS, который разрешает запросы только с доверенного домена (домена frontend-приложения).

**Глава 2. Проектирование и практическая реализация веб-приложения для управления задачами**

**2.1. Формулирование требований и проектирование системы**

**2.1.1. Функциональные и нефункциональные требования**

**Функциональные требования:**

* **Управление пользователями:** Регистрация, вход в систему, выход.
* **Управление проектами:** Создание, просмотр, редактирование и удаление проектов.
* **Управление задачами:** Создание задач в рамках проекта, назначение статуса, редактирование и удаление.
* **Визуализация:** Отображение задач на Kanban-доске с колонками "К выполнению", "В процессе", "Готово".
* **Интерактивность:** Возможность перемещать задачи между колонками с помощью drag-and-drop.

**Нефункциональные требования:**

* **Производительность:** Быстрая загрузка страниц и отклик интерфейса.
* **Безопасность:** Защита данных пользователей и обеспечение безопасного доступа.
* **Отзывчивость (Responsiveness):** Корректное отображение интерфейса на устройствах разного размера (десктоп, планшет, мобильный).
* **Масштабируемость:** Архитектура должна позволять добавление нового функционала в будущем.

**2.1.2. Проектирование архитектуры приложения**

Как упоминалось ранее, выбрана клиент-серверная архитектура.

* **Backend (Node.js/Express):** Предоставляет RESTful API для всех операций с данными (CRUD для пользователей, проектов, задач).
* **Frontend (React):** Одностраничное приложение (SPA), которое взаимодействует с backend через API, управляет состоянием и отрисовывает пользовательский интерфейс.
* **База данных (MongoDB):** Хранит все данные приложения.

**2.1.3. Проектирование схемы базы данных**

Были спроектированы три основные коллекции: Users, Projects, Tasks.

* **Users:**
  + name (String)
  + email (String, unique)
  + password (String, hashed)
* **Projects:**
  + name (String)
  + description (String)
  + user (ObjectId, ref: 'User') - ссылка на создателя проекта
* **Tasks:**
  + title (String)
  + description (String)
  + status (String, enum: ['To Do', 'In Progress', 'Done'])
  + project (ObjectId, ref: 'Project') - ссылка на проект, к которому относится задача
  + user (ObjectId, ref: 'User') - ссылка на создателя задачи

(Подробная схема представлена в Приложении Б).

**2.2. Разработка серверной части (Backend)**

Процесс разработки backend включал следующие шаги:

1. **Настройка окружения:** Инициализация Node.js проекта, установка зависимостей (express, mongoose, bcryptjs, jsonwebtoken, cors, dotenv).
2. **Создание структуры проекта:** Организация кода по модулям (models, routes, controllers, middleware) для лучшей читаемости и поддержки.
3. **Реализация моделей Mongoose:** Описание схем для User, Project, Task в соответствии с проектом.
4. **Разработка контроллеров:** Создание функций для обработки бизнес-логики (например, registerUser, loginUser, getProjects, createTask).
5. **Настройка маршрутизации (Routes):** Определение эндпоинтов API (например, POST /api/users/register) и их привязка к соответствующим контроллерам.
6. **Реализация JWT аутентификации:** Создание middleware для защиты маршрутов. При успешном логине генерируется токен, который клиент должен прикреплять к заголовкам последующих запросов. Middleware проверяет этот токен перед предоставлением доступа.

**2.3. Разработка клиентской части (Frontend)**

1. **Настройка окружения:** Создание приложения с помощью Create React App, установка зависимостей (axios, react-router-dom, @reduxjs/toolkit, react-redux, tailwindcss, react-beautiful-dnd).
2. **Настройка управления состоянием:** Использование Redux Toolkit для централизованного управления состоянием приложения. Были созданы "срезы" (slices) для аутентификации, проектов и задач. Асинхронные операции (запросы к API) обрабатываются с помощью createAsyncThunk.
3. **Реализация маршрутизации:** Использование React Router для создания навигации между страницами (Логин, Регистрация, Панель управления, Страница проекта).
4. **Создание UI-компонентов:** Разработка переиспользуемых компонентов с помощью React и стилизация с помощью Tailwind CSS. Ключевые компоненты: ProjectCard, TaskCard, KanbanColumn, Modal.
5. **Интеграция с API:** Создание сервисного слоя с использованием Axios для отправки HTTP-запросов на backend.
6. **Реализация Drag-and-Drop:** Использование библиотеки react-beautiful-dnd для реализации перетаскивания задач между колонками на Kanban-доске.

**2.4. Тестирование и обеспечение качества**

* **Модульное тестирование (Unit Testing):**
  + **Backend:** С помощью Jest и Supertest были написаны тесты для проверки API эндпоинтов: корректность кодов ответа, форматов данных и работы логики.
  + **Frontend:** С помощью Jest и React Testing Library тестировались отдельные React-компоненты на предмет правильной отрисовки и реакции на действия пользователя.
* **Ручное тестирование:** Проведение комплексного тестирования по основным пользовательским сценариям (регистрация -> создание проекта -> добавление задач -> изменение статуса). Все найденные дефекты фиксировались и исправлялись.

**2.5. Развертывание (деплой) приложения**

Для развертывания приложения был выбран подход с использованием контейнеризации **Docker**.

1. **Создание Dockerfile:** Были созданы отдельные Dockerfile для backend и frontend.
   * Для backend: создается образ Node.js, устанавливаются зависимости и запускается сервер.
   * Для frontend: используется многоэтапная сборка. На первом этапе собирается статический билд приложения, на втором — эти файлы помещаются в легковесный веб-сервер (Nginx).
2. **Развертывание на Heroku:**
   * Backend-приложение было развернуто на Heroku как Docker-контейнер.
   * Frontend-приложение было развернуто на платформе Vercel, которая оптимизирована для хостинга статических сайтов и SPA.
   * Настроены переменные окружения (например, строка подключения к базе данных, секретный ключ JWT) на хостинг-платформах.

**Заключение**

В ходе выполнения данной дипломной работы была успешно достигнута поставленная цель — спроектировано и разработано полнофункциональное веб-приложение для управления задачами и проектами на стеке MERN.

**Основные результаты:**

1. Проведен анализ предметной области, на основе которого были сформированы требования к приложению.
2. Выбран и обоснован современный и востребованный на рынке стек технологий.
3. Разработана и реализована клиент-серверная архитектура, обеспечивающая гибкость и масштабируемость системы.
4. Создан backend с RESTful API для управления данными и backend-логикой.
5. Реализован безопасный механизм аутентификации пользователей на основе JWT.
6. Разработан интерактивный и адаптивный frontend с использованием React и Redux, ключевой особенностью которого является Kanban-доска с функцией drag-and-drop.
7. Проведено тестирование и развертывание приложения, подтвердившее его работоспособность.

**Практическая значимость** работы заключается в создании готового к использованию продукта, который может служить альтернативой более сложным и дорогим системам для небольших команд или индивидуальных разработчиков. Кроме того, данный проект является комплексным примером разработки современного full-stack приложения и может служить образовательным материалом.

**Перспективы дальнейшего развития проекта:**

* Внедрение real-time обновлений с помощью WebSockets для мгновенного отображения изменений у всех участников проекта.
* Добавление системы уведомлений (внутри приложения и по email).
* Реализация ролевой модели доступа (администратор, участник).
* Создание дашбордов с аналитикой и отчетами по проектам.
* Интеграция со сторонними сервисами, такими как Git-репозитории (GitHub, GitLab) или мессенджеры (Slack, Telegram).

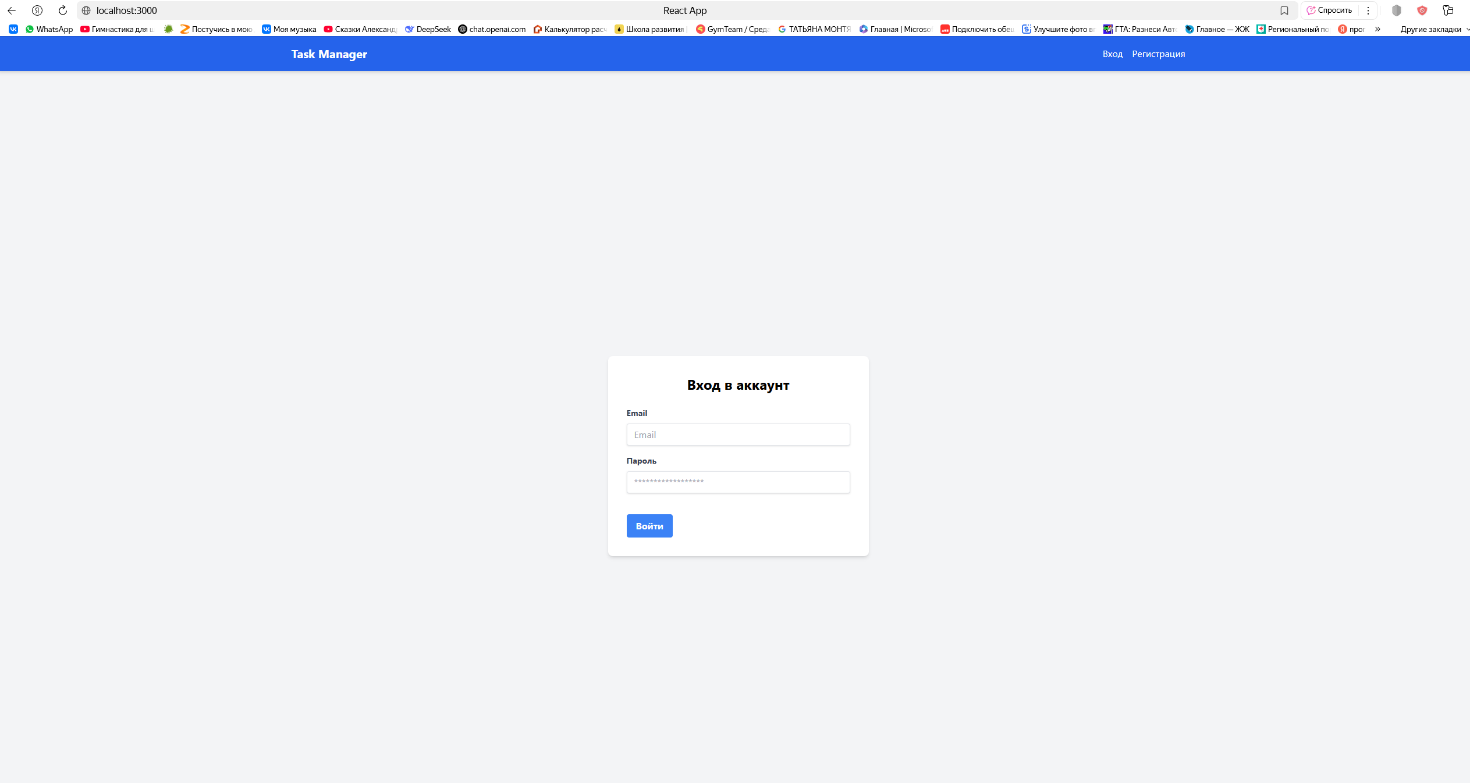
Таким образом, все поставленные в начале работы задачи были успешно выполнены, а полученный результат подтверждает актуальность и практическую ценность выбранной темы.

**Список используемой литературы**

1. Флэнаган Д. JavaScript. Подробное руководство. 7-е издание. – СПб.: Питер, 2021. – 720 с.
2. Бэнкс А., Порселло Е. React и Redux. Функциональная веб-разработка. – СПб.: Питер, 2019. – 336 с.
3. Хортон А. Node.js, MongoDB и Angular. Разработка полноценных веб-приложений. – М.: Вильямс, 2019. – 672 с.
4. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. – 352 с.
5. Официальная документация React. [Электронный ресурс]. URL: <https://reactjs.org/>
6. Официальная документация Node.js. [Электронный ресурс]. URL: <https://nodejs.org/>
7. Официальная документация Express.js. [Электронный ресурс]. URL: <https://expressjs.com/>
8. Официальная документация MongoDB. [Электронный ресурс]. URL: <https://www.mongodb.com/docs/>
9. Гайд по Kanban. Atlassian. [Электронный ресурс]. URL: <https://www.atlassian.com/ru/agile/kanban>

**Приложения**

**Приложение А: Скриншоты пользовательского интерфейса**

Рисунок А.1 - Страница входа

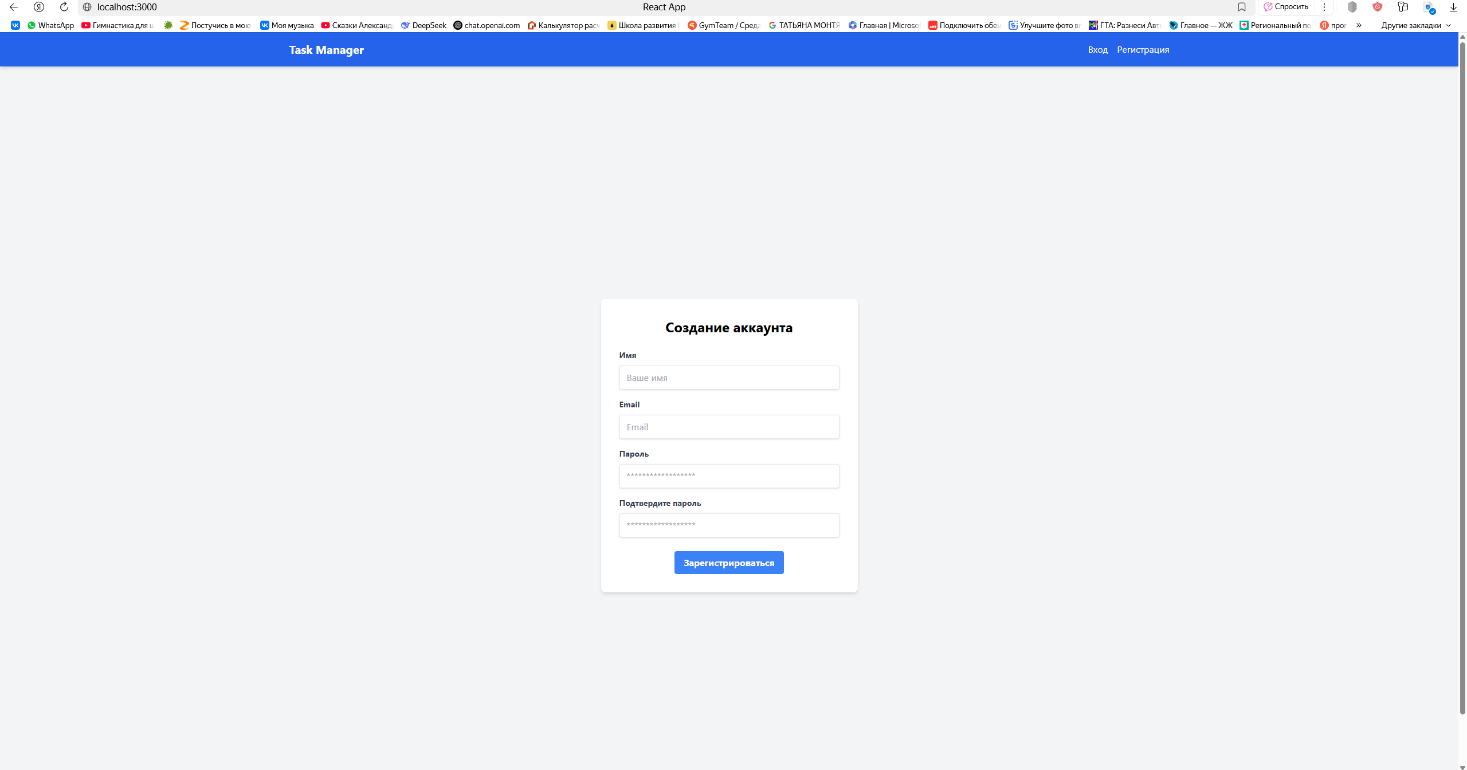


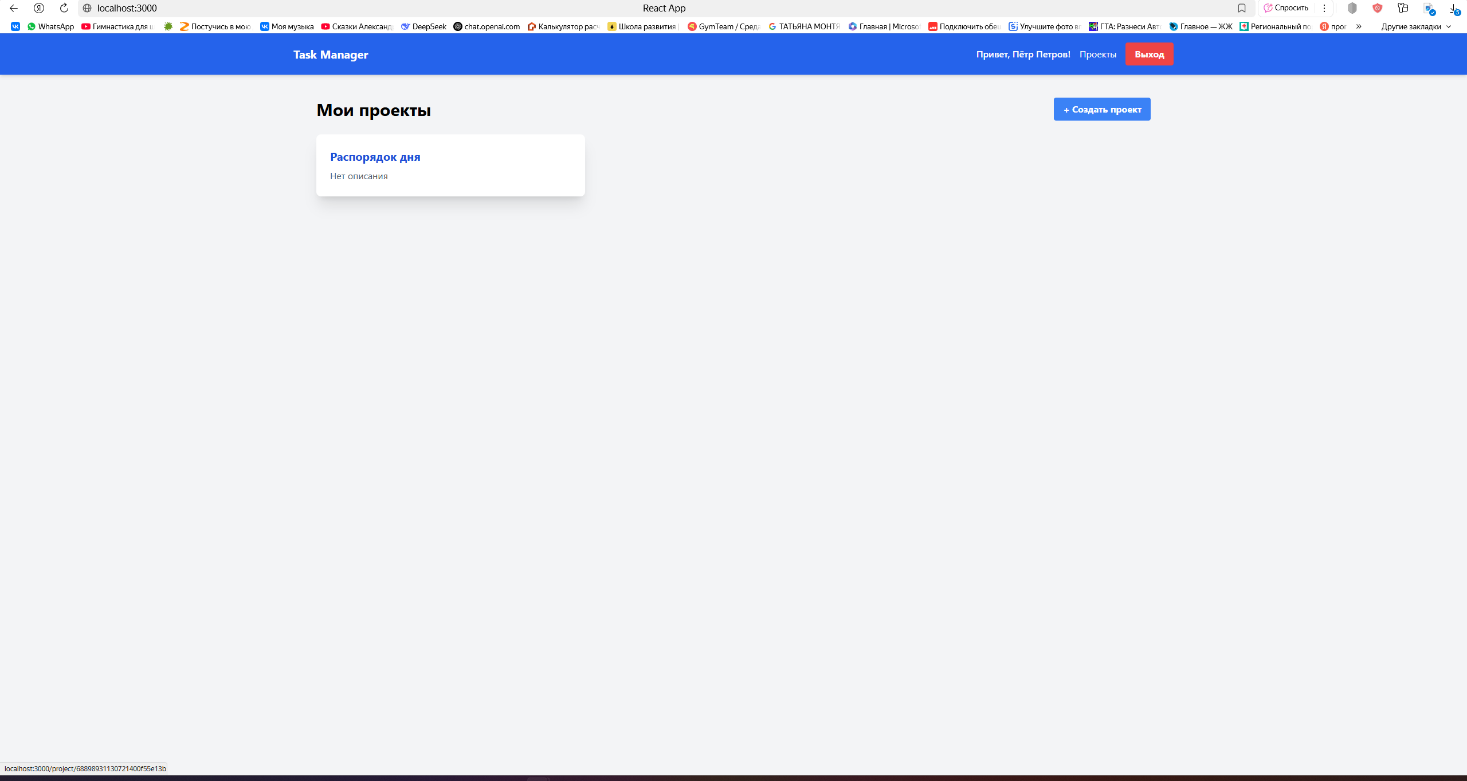
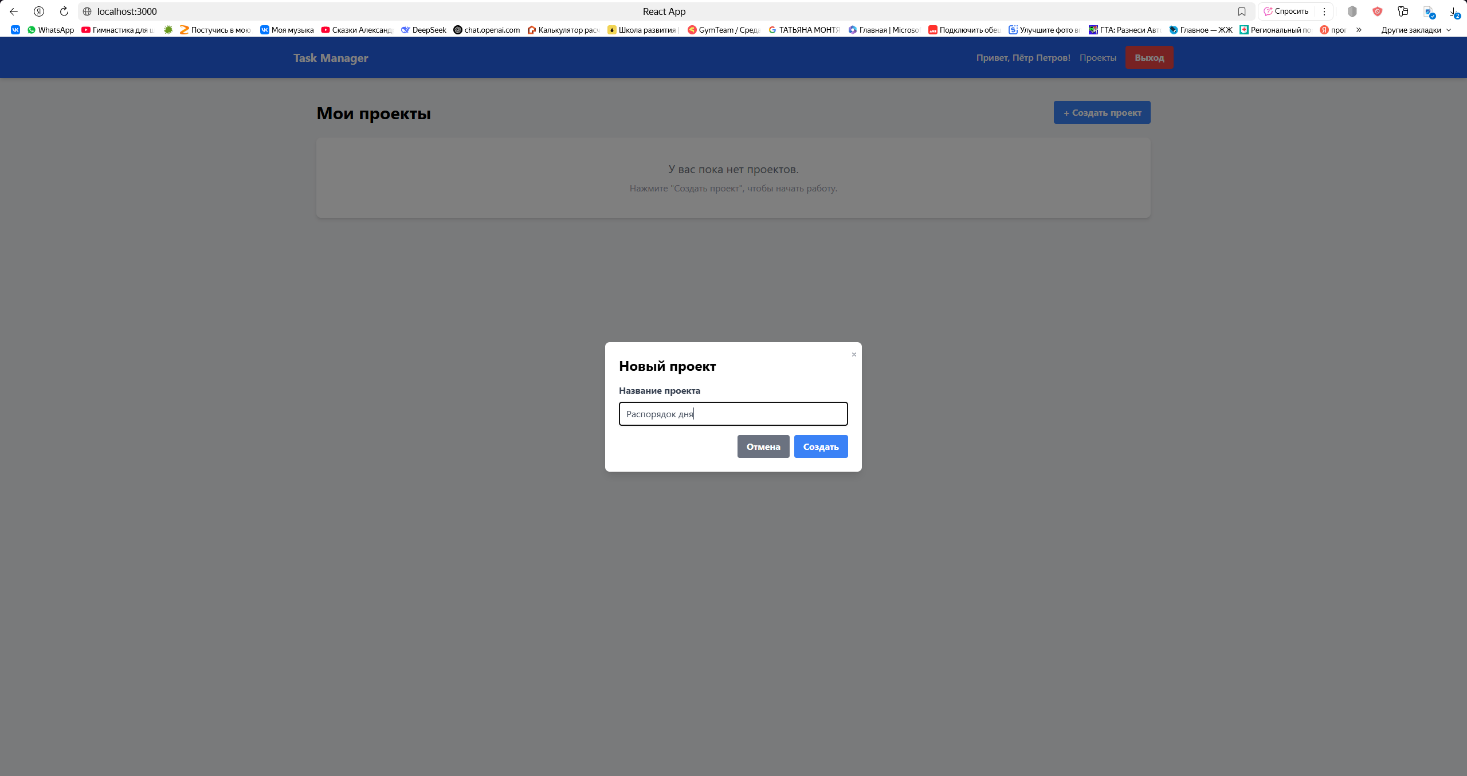
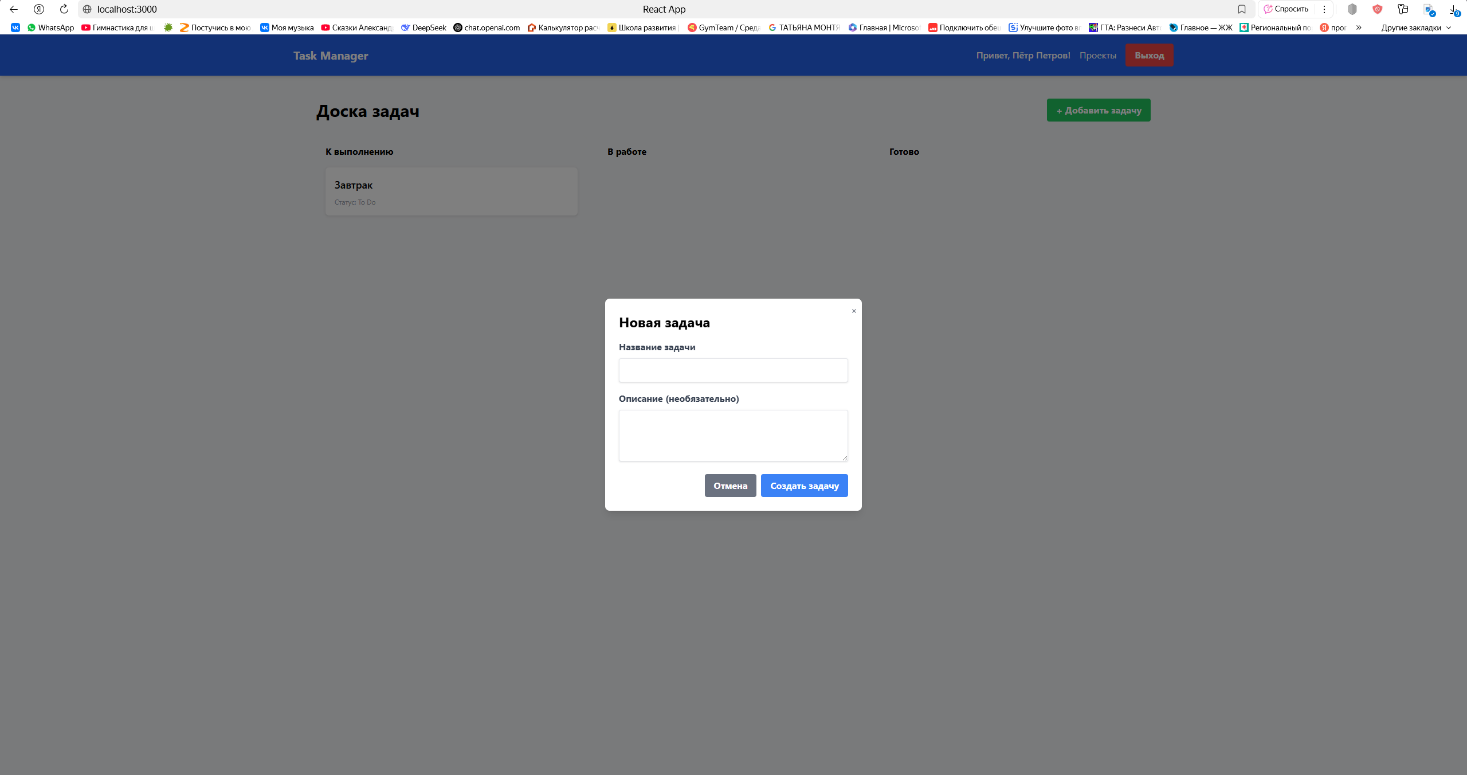
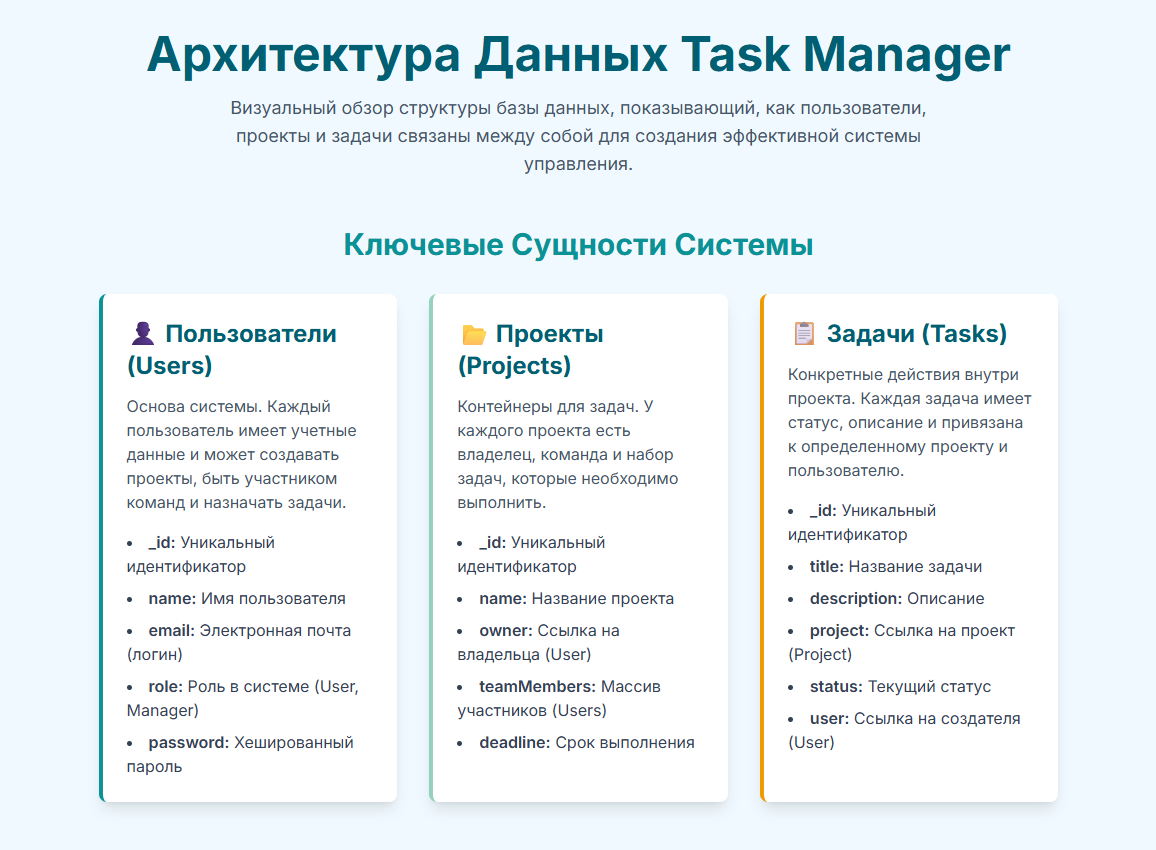
Рисунок А.2 - Панель управления

Рисунок А.3 - Kanban-доска проекта

**Приложение Б: Схема базы данных**

Рисунок Б.1 – Визуализация связей

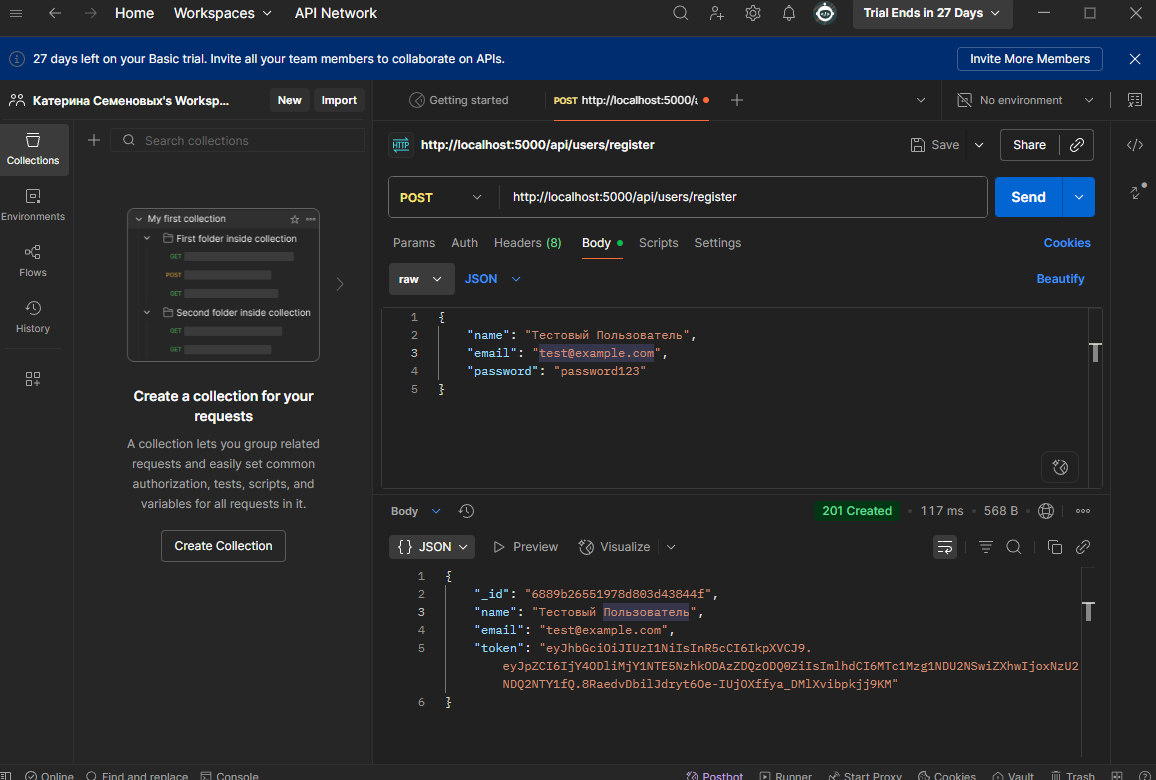


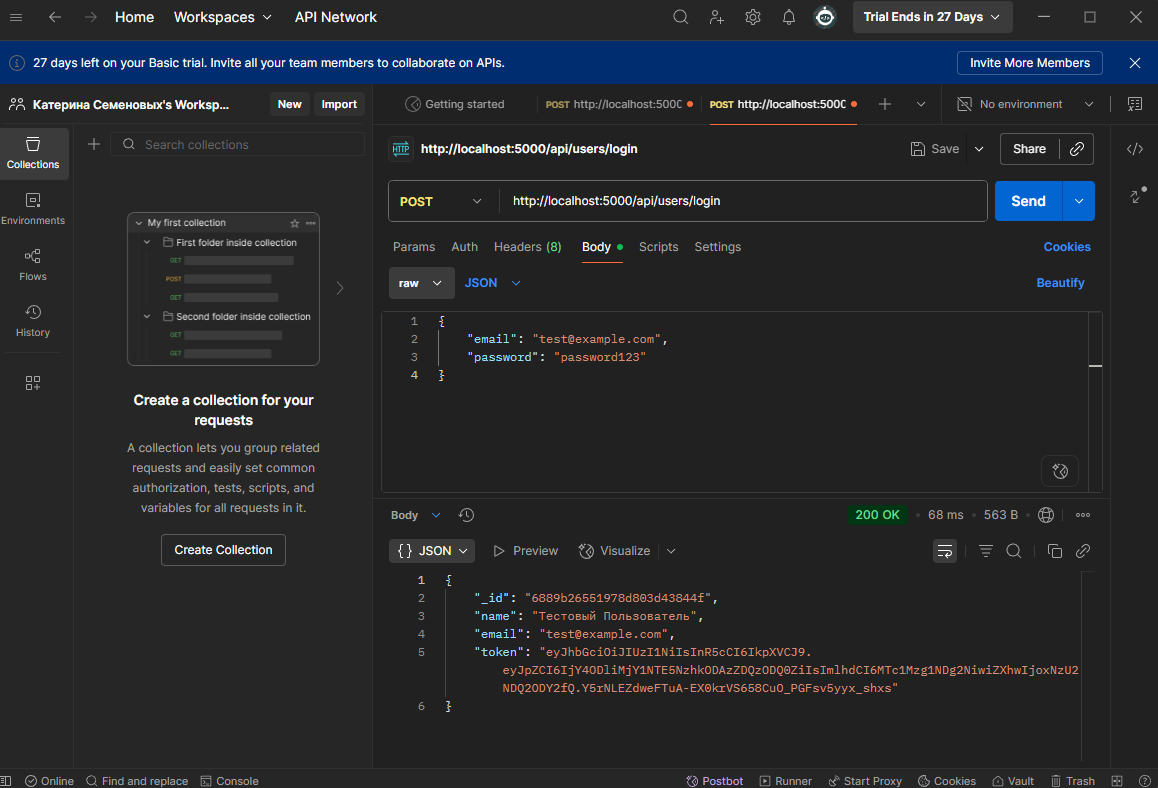
Рисунок Б.2– Архитектура данных 

**Приложение В: Документация API (пример)**

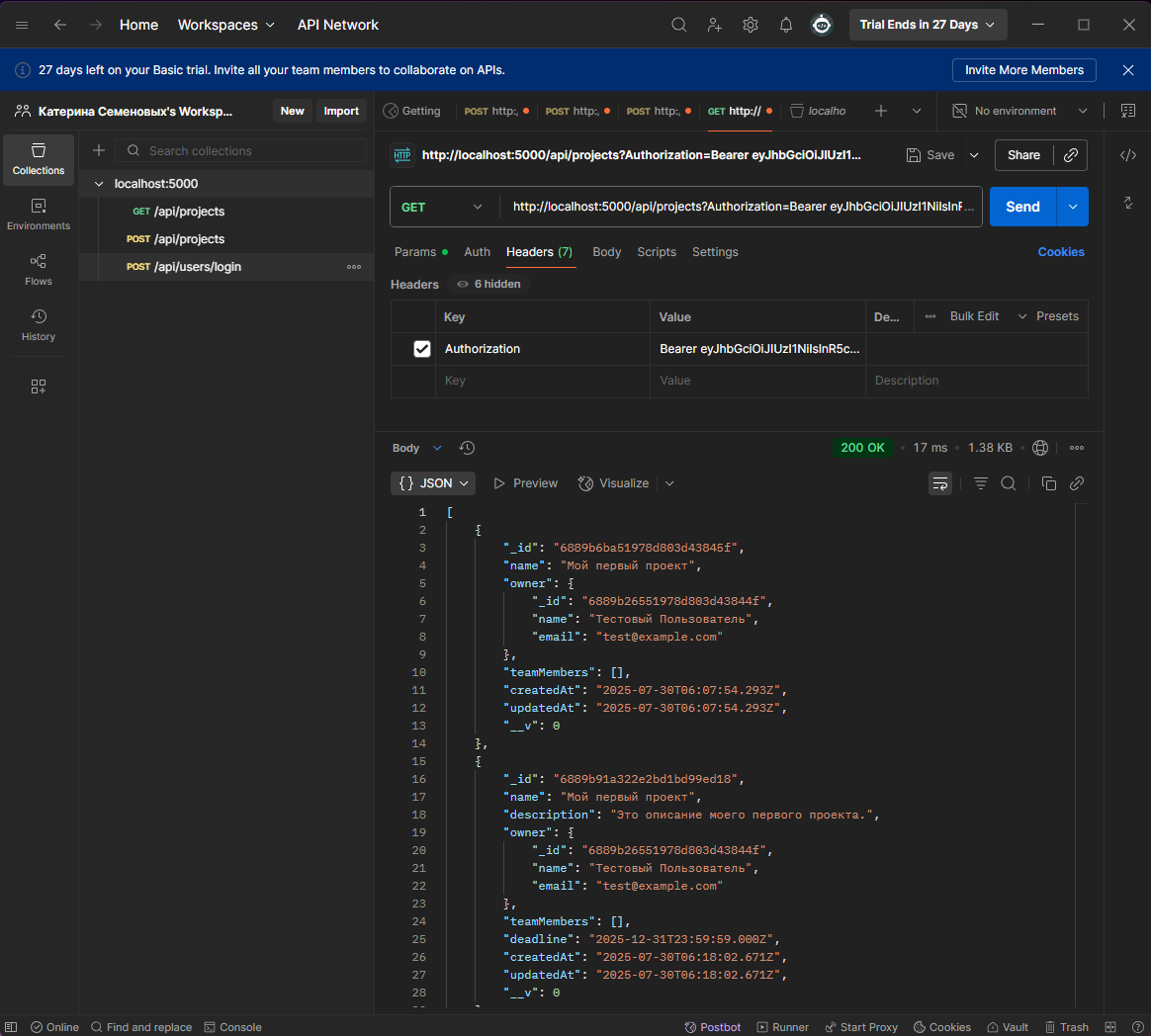
**Аутентификация**

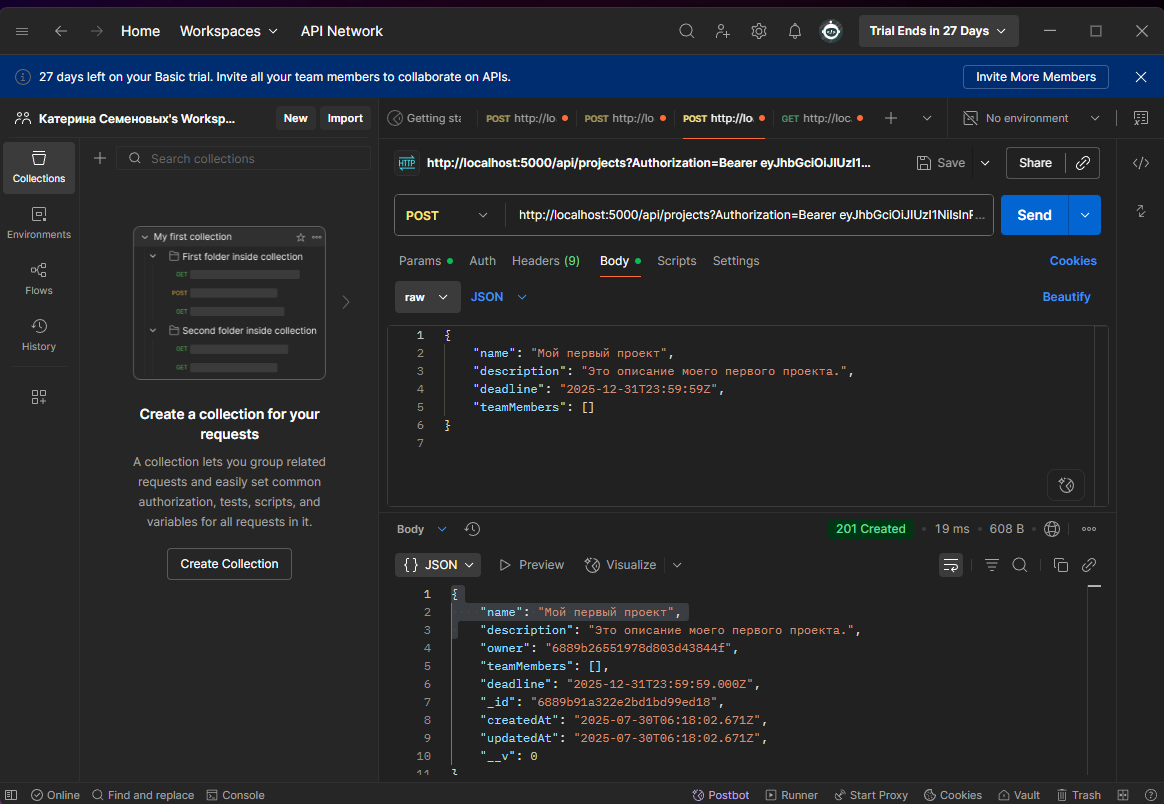
* POST /api/users/register - Регистрация нового пользователя.



* POST /api/users/login - Вход пользователя. 

**Проекты (требуется аутентификация)**

* GET /api/projects - Получить все проекты текущего пользователя.
* POST /api/projects - Создать новый проект.



**Задачи (требуется аутентификация)**

* GET /api/tasks/:projectId - Получить все задачи для указанного проекта.

Id progect: 6889b9ee322e2bd1bd99ed1b?

http://localhost:5000/api/tasks/ 6889b9ee322e2bd1bd99ed1b?Authorization=Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ODliMjY1NTE5NzhkODAzZDQzODQ0ZiIsImlhdCI6MTc1Mzg1NjI0MSwiZXhwIjoxNzU2NDQ4MjQxfQ.f\_pmXRWeKMkho-jGNsffyQHPjUF7uFOB-IPJKt-AxGQ&

Headers:

Key: Authorization

Value: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ODliMjY1NTE5NzhkODAzZDQzODQ0ZiIsImlhdCI6MTc1Mzg1NjI0MSwiZXhwIjoxNzU2NDQ4MjQxfQ.f\_pmXRWeKMkho-jGNsffyQHPjUF7uFOB-IPJKt-AxGQ

POST /api/tasks - Создать новую задачу.

Id progect: 6889b9ee322e2bd1bd99ed1b?

http://localhost:5000/api/tasks/ 6889b9ee322e2bd1bd99ed1b?Authorization=Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ODliMjY1NTE5NzhkODAzZDQzODQ0ZiIsImlhdCI6MTc1Mzg1NjI0MSwiZXhwIjoxNzU2NDQ4MjQxfQ.f\_pmXRWeKMkho-jGNsffyQHPjUF7uFOB-IPJKt-AxGQ&

Headers:

Key: Content-Type

Value: application/json

Key: Authorization

Value: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ODliMjY1NTE5NzhkODAzZDQzODQ0ZiIsImlhdCI6MTc1Mzg1NjI0MSwiZXhwIjoxNzU2NDQ4MjQxfQ.f\_pmXRWeKMkho-jGNsffyQHPjUF7uFOB-IPJKt-AxGQ

Body:

{ "title": "Сделать макет главной страницы", "description": "Разработать дизайн и структуру главной страницы.", "project": "6889b9ee322e2bd1bd99ed1b?" }

PUT /api/tasks/:taskId - Обновить задачу (например, изменить статус).

http://localhost:5000/api/tasks/<ID\_ЗАДАЧИ>

Headers:

Key: Content-Type

Value: application/json

Key: Authorization

Value: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4ODliMjY1NTE5NzhkODAzZDQzODQ0ZiIsImlhdCI6MTc1Mzg1NjI0MSwiZXhwIjoxNzU2NDQ4MjQxfQ.f\_pmXRWeKMkho-jGNsffyQHPjUF7uFOB-IPJKt-AxGQ

Body:

{ "status": "In Progress", "description": "Начата работа над макетом, основные элементы готовы." }

