**«BusinessPlanner»**

1. Архитектура приложения

Приложение «BusinessPlanner» планируется построить по архитектуре клиент-сервер. Эта архитектура позволяет разделить фронтенд и бэкенд, что обеспечивает гибкость, масштабируемость и возможность независимой разработки каждого компонента.

* Клиент (Frontend): Пользовательский интерфейс, реализованный с использованием технологий, таких как React, который взаимодействует с сервером через HTTP-запросы.
* Сервер (Backend): Обработка логики приложения, запросов от клиента и взаимодействие с базой данных, реализованное с использованием Node.js и Express.
* База данных: Хранение данных о пользователях, задачах и тегах с использованием PostgreSQL.

2. API

Пользователи (Users)

* POST /api/users: Создание нового пользователя.
* GET /api/users: Получение списка пользователей.
* GET /api/users/:id: Получение информации о пользователе по ID.
* PUT /api/users/:id: Обновление информации о пользователе.

Задачи (Tasks)

* POST /api/tasks: Создание новой задачи.
* GET /api/tasks: Получение списка задач.
* GET /api/tasks/:id: Получение информации о задаче по ID.
* PUT /api/tasks/:id: Обновление информации о задаче.
* DELETE /api/tasks/:id: Удаление задачи.

Теги (Tags)

* POST /api/tags: Создание нового тега.
* GET /api/tags: Получение списка тегов.

Связь Задач и Тегов (TaskTags)

* POST /api/tasktags: Создание связи между задачей и тегом.
* GET /api/tasktags/task/:taskId: Получение тегов по ID задачи.

3. Схема взаимодействия

1. Клиент (UI): отправляет HTTP-запросы к серверу для выполнения операций (создание, получение, обновление и удаление данных).
2. Сервер: обрабатывает запросы, взаимодействует с базой данных, выполняет бизнес-логику и возвращает ответы клиенту.
3. База данных: хранит информацию о пользователях, задачах и тегах.

**4. Схема базы данных**

**Таблица: Users (Пользователи)**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | integer | Уникальный идентификатор пользователя (PK) |
| name | varchar | Имя пользователя |
| email | varchar | Электронная почта пользователя |
| password | varchar | Защищенный пароль пользователя |

**Таблица: Tasks (Задачи)**

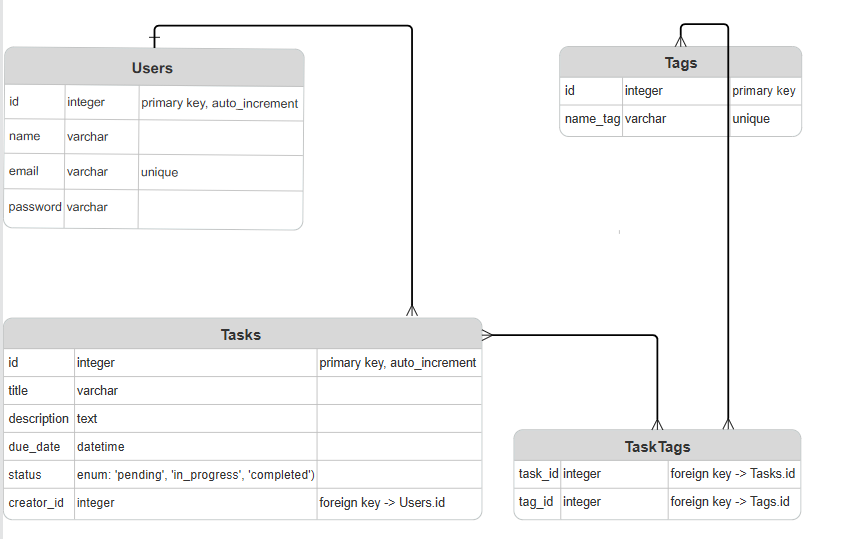
|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | integer | Уникальный идентификатор задачи (PK) |
| title | varchar | Название задачи |
| description | text | Описание задачи |
| dueDate | datetime | Дата завершения задачи |
| status | Enum (‘pending’, ‘in\_progress’, ‘completed’) | Статус задачи |
| creator\_id | integer | Идентификатор пользователя, создавшего задачу |

**Таблица: Tags (Теги)**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | integer | Уникальный идентификатор тега (PK) |
| name | varchar | Название тега |

**Таблица: TaskTags (Связь Задач и Тегов)**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| task\_id | integer | Идентификатор задачи (FK) |
| tag\_id | integer | Идентификатор тега (FK) |



**5. Стек технологий**

* **Frontend**:
  + **HTML**: основной язык разметки для создания структуры веб-страниц
  + **CSS**: язык каскадных стилей, используемый для оформления и стилизации HTML-страниц.
  + **React**: для создания интерактивного пользовательского интерфейса.
* **Backend**:
  + **Java**: основной язык программирования для разработки серверной части приложения
  + **PostgreSQL**: реляционные базы данных для хранения данных
* **Аутентификация**:
  + **JWT (JSON Web Tokens)**: для аутентификации пользователей.
* **Дополнительные технологии**:
  + **Docker**: для контейнеризации приложения.
  + **Postman**: для тестирования API.
  + **Git**: для управления версиями кода.