**📝 Техническое задание**

**Проект:** Task Tracker **(Минимальный таск-трекер с календарём)**

**1. 🎯 Цель проекта**

Разработать минимальную систему управления задачами с календарём, позволяющую пользователям:

* проходить аутентификацию,
* создавать, просматривать и управлять своими задачами,
* привязывать задачи к конкретной дате и времени,
* видеть свои задачи в календарном виде.

Важно: каждый пользователь видит только свои задачи.

**2. 🧱 Структура данных (основные сущности и поля)**

**Сущность:** User

Представляет зарегистрированного пользователя.

|  |  |  |
| --- | --- | --- |
| id | Integer (PK) | Уникальный идентификатор |
| username | String (unique) | Уникальное имя пользователя |
| email | String (unique) | Электронная почта |
| password\_hash | String | Хэш пароля (не хранить в открытом виде!) |

**Сущность:** Task

Представляет задачу пользователя.

|  |  |  |
| --- | --- | --- |
| id | Integer (PK) | Уникальный идентификатор задачи |
| title | String | Краткое название задачи |
| description | Text (опционально) | Подробное описание |
| due\_datetime | DateTime | Дата и время выполнения задачи |
| is\_completed | Boolean | Статус выполнения (по умолчанию —False) |
| user\_id | Integer (FK) | Ссылка на владельца задачи (User.id) |

Примечание:

* due\_datetime может быть NULL, если задача не привязана ко времени (но по ТЗ — привязка обязательна, поэтому лучше сделать NOT NULL).
* Все задачи автоматически привязаны к пользователю через user\_id.

**3. 🏗️ Общая структура проекта**

**Выбор технологий**

|  |  |  |
| --- | --- | --- |
| Backend | Python + FastAPI | Современный, асинхронный, легко документируется (автоматическая генерация OpenAPI/Swagger UI), отлично подходит для REST API |
| База данных | PostgreSQL (через SQLAlchemy ORM) | Надёжная реляционная СУБД, поддержка дат/времени, масштабируемость |
| Frontend | React + Vite | Популярный SPA-фреймворк, удобен для календарных интерфейсов (можно использовать библиотеки вродеreact-big-calendar) |
| Аутентификация | JWT (JSON Web Tokens) | Простая, stateless аутентификация без сессий |
| Развёртывание (опционально) | Docker + docker-compose | Для локального запуска и будущего деплоя |

**Структура проекта (папки)**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

task\_tracker/

├── backend/

│ ├── app/

│ │ ├── main.py # точка входа FastAPI

│ │ ├── models/ # SQLAlchemy модели

│ │ ├── schemas/ # Pydantic схемы

│ │ ├── crud/ # операции с БД

│ │ ├── auth/ # логика аутентификации

│ │ └── database.py # подключение к БД

│ ├── requirements.txt

│ └── alembic/ # миграции БД

│

├── frontend/

│ ├── public/

│ ├── src/

│ │ ├── components/ # календарь, формы и т.д.

│ │ ├── services/ # API-клиент

│ │ └── App.jsx

│ └── package.json

│

└── docker-compose.yml # (опционально)

**4. 🔐 Логика авторизации и работы с задачами**

**Аутентификация**

1. Регистрация:
   * Пользователь отправляет username, email, password.
   * Сервер хэширует пароль (например, через bcrypt) и сохраняет в БД.
   * Возвращает сообщение об успехе.
2. Вход:
   * Пользователь отправляет username и password.
   * Сервер проверяет хэш пароля.
   * При успехе генерирует JWT-токен (срок жизни — например, 24 часа) и возвращает его.
3. Защита эндпоинтов:
   * Все эндпоинты для работы с задачами требуют заголовок Authorization: Bearer <token>.
   * Сервер декодирует токен → получает user\_id → фильтрует задачи только по этому ID.

**Работа с задачами**

* Создание задачи:
  + POST /tasks/ с телом: { "title": "...", "description": "...", "due\_datetime": "2025-10-20T15:00:00" }
  + Сервер привязывает задачу к user\_id из JWT.
* Получение задач:
  + GET /tasks/ → возвращает список задач текущего пользователя.
  + Можно добавить фильтрацию по дате: ?start=2025-10-01&end=2025-10-31
* Обновление / удаление:
  + PUT/PATCH и DELETE по /tasks/{task\_id} — только если задача принадлежит пользователю.
* Календарь на фронтенде:
  + Фронтенд запрашивает задачи за выбранный месяц.
  + Отображает их в виде событий на календаре (например, цветом, временем, заголовком).

**5. 📡 Пример API (REST)**

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /auth/register | Регистрация нового пользователя | ❌ |
| POST | /auth/login | Получение JWT-токена | ❌ |
| POST | /tasks/ | Создать новую задачу | ✅ |
| GET | /tasks/ | Получить все задачи пользователя | ✅ |
| GET | /tasks/?start=...&end=... | Получить задачи за период | ✅ |
| PUT | /tasks/{id} | Обновить задачу | ✅ |
| DELETE | /tasks/{id} | Удалить задачу | ✅ |

Все ответы в формате JSON.

**6. 🗃️ Пример таблиц в БД (PostgreSQL)**

sql

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

⌄

⌄

-- Таблица пользователей

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password\_hash TEXT NOT NULL

);

-- Таблица задач

CREATE TABLE tasks (

id SERIAL PRIMARY KEY,

title VARCHAR(100) NOT NULL,

description TEXT,

due\_datetime TIMESTAMP NOT NULL,

is\_completed BOOLEAN DEFAULT FALSE,

user\_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE

);

**7. ✅ Дополнительные соображения**

* Валидация: все входные данные проверяются через Pydantic (на бэкенде) и на фронтенде.
* Безопасность: пароли хэшируются, JWT подписываются секретным ключом, HTTPS в продакшене.
* Масштабируемость: можно легко добавить категории задач, приоритеты, повторяющиеся события и т.д.
* Тестирование: написать unit-тесты для CRUD-операций и аутентификации (pytest).

**📌 Заключение**

Проект реализует минимально жизнеспособный продукт (MVP) таск-трекера с календарём, соответствующий требованиям ТЗ. Архитектура разделена на фронтенд и бэкенд, обеспечивает безопасность, изоляцию данных пользователей и удобный интерфейс для работы с задачами.