Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

«Лабораторная работа №1»

Автор: Кононов Степан

Факультет: ИКТ

Группа: К33392

Санкт-Петербург 2024

Task manager

Цель работы

Целью данной работы была разработка простого приложения для тайм менеджмента с использованием FastAPI, PostgreSQL и SQLAlchemy. В приложение включены следующие функции:

1. Управление пользователями с возможностью регистрации и авторизации.
2. Создание и управление задачами.
3. Управление проектами.
4. Присвоение задачам приоритетов и категорий.

Модель данных

В базе данных были реализованы следующие сущности:

1. **User** (Пользователь)
    - Поля: ID, имя, email, хэшированный пароль, дата создания.
2. **Task** (Задача)
    - Поля: ID, заголовок, описание, дата создания, дедлайн, время начала, время окончания, пользователь (foreign key на User), проект (foreign key на Project), приоритет (foreign key на Priority), категории (many-to-many связь с Category).
3. **Project** (Проект)
    - Поля: ID, название, описание, дата создания.
4. **Priority** (Приоритет)
    - Поля: ID, уровень приоритета, описание.
5. **Category** (Категория)
    - Поля: ID, название, описание.

Схема базы данных включает связи many-to-many и one-to-many. Ассоциативной сущностью является связь задач и категорий.

Технологии

1. **FastAPI** - Веб-фреймворк для создания API.
2. **SQLAlchemy** - ORM для работы с базой данных.
3. **PostgreSQL** - Реляционная база данных для хранения данных.
4. **Alembic** - Инструмент для миграций базы данных.
5. **Pydantic** - Валидация данных, аннотация типов.

Реализованные функции

Управление пользователями

1. **Регистрация**: позволяет создать нового пользователя с именем, email и паролем. Пароль хэшируется.
2. **Аутентификация**: пользователи могут аутентифицироваться с помощью email и пароля.
3. **JWT-токены**: для аутентификации пользователей используются JWT-токены.
4. **Получение информации о пользователе**: позволяет получить информацию о пользователе по ID.
5. **Получение списка пользователей**: позволяет получить список всех пользователей.
6. **Смена пароля**: позволяет пользователю сменить свой пароль.

Управление задачами

1. **Создание задачи**: позволяет создать новую задачу с заголовком, описанием, дедлайном, временем начала и окончания, приоритетом, категориями.
2. **Получение задачи**: позволяет получить информацию о задаче по ID.
3. **Получение списка задач**: позволяет получить список всех задач.
4. **Обновление задачи**: позволяет обновить данные существующей задачи.
5. **Удаление задачи**: позволяет удалить задачу по ID.

Управление проектами

1. **Создание проекта**: позволяет создать новый проект с названием и описанием.
2. **Получение проекта**: позволяет получить информацию о проекте по ID.

3. **Получение списка проектов**: позволяет получить список всех проектов.
4. **Обновление проекта**: позволяет обновить данные существующего проекта.
5. **Удаление проекта**: позволяет удалить проект по ID.

Управление приоритетами и категориями

1. **Создание приоритета**: позволяет создать новый приоритет для задач.
2. **Получение приоритета**: позволяет получить информацию о приоритете по ID.
3. **Создание категории**: позволяет создать новую категорию для задач.
4. **Получение категории**: позволяет получить информацию о категории по ID.

Система миграций

Для управления миграциями базы данных был использован Alembic. Настроена система миграций, позволяющая управлять изменениями схемы базы данных.

Пример модели данных

```
class Project(Base):
    __tablename__ = "projects"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    description = Column(String)
    created_at = Column(DateTime, default=datetime.utcnow)

    tasks = relationship("Task", back_populates="project")
```

Пример эндпоинта

```
@router.put("/projects/{project_id}", response_model=schema
s.Project)
def update_project(project_id: int, project: schemas.Projec
tCreate, db: Session = Depends(get_db)):
    db_project = db.query(models.Project).filter(models.Pro
ject.id == project_id).first()
    if db_project is None:
        raise HTTPException(status_code=404, detail="Projec
```

```
t not found")
    for key, value in project.dict().items():
        setattr(db_project, key, value)
    db.commit()
    db.refresh(db_project)
    return db_project
```

Документация

Документация для API была создана в Postman и включает следующие коллекции:

1. **Auth**: регистрация и авторизация пользователей.
2. **Users**: управление пользователями.
3. **Tasks**: управление задачами.
4. **Projects**: управление проектами.
5. **Priorities**: управление приоритетами.
6. **Categories**: управление категориями.

Файловая структура проекта

```
project/
|-- alembic/
|-- app/
|   |-- crud/
|   |   |-- users.py
|   |   |-- tasks.py
|   |   |-- projects.py
|   |   |-- priorities.py
|   |   |-- categories.py
|   |-- models/
|   |   |-- user.py
|   |   |-- task.py
|   |   |-- project.py
|   |   |-- priority.py
|   |   |-- category.py
|   |-- schemas/
|   |   |-- user.py
```

```
|   |   |-- task.py
|   |   |-- project.py
|   |   |-- priority.py
|   |   |-- category.py
|   |-- main.py
|-- tests/
|-- alembic.ini
|-- requirements.txt
README.md
```

Вывод

В рамках данной лабораторной работы было успешно разработано простое приложение для тайм менеджмента, включающее управление пользователями, задачами, проектами, приоритетами и категориями. API приложения документировано с помощью Postman, а база данных оснащена системой миграций. Реализованы все требуемые функции, а также обеспечена безопасность аутентификации с использованием JWT-токенов и хэширования паролей.