Вводная лекция по использованию баз данных

**Цель занятия**

* Показать, как сохранять данные с фронтенда (React) через FastAPI в реальную базу данных.
* Научить использовать **SQLite** через Python-библиотеку **SQLAlchemy** — без SQL.
* Продолжить мини-приложение «Заметки»: теперь все добавленные заметки будут сохраняться не в памяти, а в базе.

**1. Что изменилось**

На прошлой лекции:

* Заметки хранились в списке в оперативной памяти.
* После перезапуска сервера они исчезали.

Теперь:

* Заметки будут храниться в файле базы данных (notes.db).
* Можно будет сохранять, читать и расширять список без потери данных.

**2. Что такое SQLite и SQLAlchemy**

**SQLite**

* Простая база данных, хранится в одном файле.
* Не требует отдельного сервера.
* Отлично подходит для учебных и небольших проектов.
* Устанавливается вместе с Python — ничего дополнительно не нужно.

**SQLAlchemy**

* Библиотека Python, которая позволяет описывать таблицы и работать с данными как с объектами Python.
* Вместо SQL-запросов — обычные методы Python.
* Идеально подходит для начинающих, можно понять логику работы с базой, а не синтаксис SQL.

**3. Установка зависимостей**



**4. Настройка проекта**

Проект можно организовать так (создать файлы):

project/

│

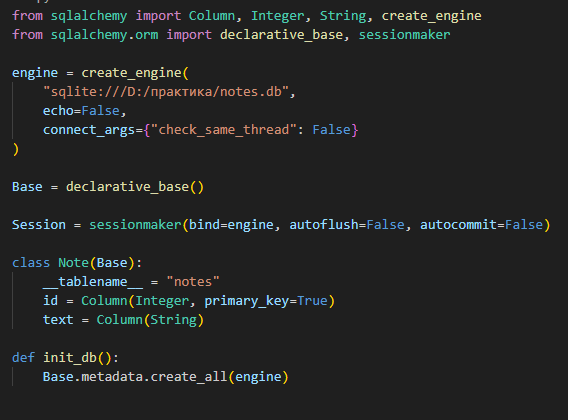
├── main.py # FastAPI-сервер

├── database.py # всё, что связано с базой

└── notes.db # сама база

**5. Создание базы и модели Note**

**Файл database.py**

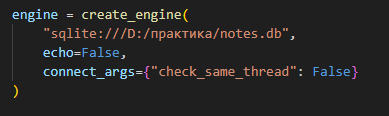


Подробные комментарии к программе:



Импортируем необходимые классы и функции из библиотеки SQLAlchemy:

* Column, Integer, String — для описания структуры таблицы (модели).
* create\_engine — для создания подключения к базе данных.
* declarative\_base — для определения базового класса модели.
* sessionmaker — для создания объектов сессий, которые управляют транзакциями.



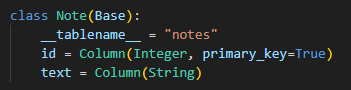
* Создаем «движок» (engine), который отвечает за соединение с базой данных.
* "sqlite:///D:/практика/notes.db" — строка подключения к базе SQLite, где указано местоположение файла с базой.
* Параметр echo=False отключает вывод в консоль всех SQL-запросов (для начинающих часто удобнее не видеть много лишней информации).
* connect\_args={"check\_same\_thread": False} — специальный параметр для SQLite, который позволяет использовать соединение из разных потоков, что полезно в некоторых многопоточных приложениях.



* Создаем базовый класс для всех моделей (таблиц) нашей базы данных. От него будут наследоваться все модели.



* Определяем фабрику сессий (Session), с помощью которой создаются объекты сессий.
* bind=engine — сессия привязывается к нашему движку (подключению).
* autoflush=False — отключает автоматическую отправку изменений в базу до вызова commit() (упрощает контроль за сохранением данных).
* autocommit=False — отключает автоматическое подтверждение транзакций, что позволяет вручную контролировать когда данные сохраняются.



* Определяем класс модели Note, который описывает таблицу notes в базе.
* tablename\_\_ = "notes" — указывает имя таблицы.
* id = Column(Integer, primary\_key=True) — столбец id — целочисленный идентификатор, который является первичным ключом (уникальный идентификатор записи).
* text = Column(String) — столбец text для хранения текста заметки, тип — строка.



* Функция init\_db создает таблицы в базе данных, если они еще не существуют.
* Base.metadata.create\_all(engine) — просматривает все модели, унаследованные от Base, и создает соответствующие таблицы.

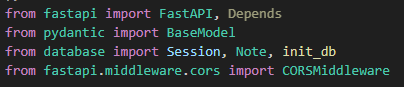
Этот код задает базовую структуру для работы с базой данных в SQLAlchemy: подключение к SQLite, создание класса модели таблицы и функцию инициализации базы. Это фундамент для взаимодействия с данными — добавления, изменения и получения записей. Такой подход удобен, т.к. позволяет описывать таблицы в виде обычных классов Python и работать с данными через объекты.

**6. FastAPI-сервер с базой данных**

**Файл main.py**

****

Подробные комментарии к программе:

****

Импортируем необходимые компоненты:

* FastAPI — основной класс для создания API.
* Depends — для внедрения зависимостей (здесь не используется, но часто применяется для работы с сессиями).
* BaseModel из Pydantic — для определения модели данных, которая валидирует входящие запросы.
* Из нашего модуля database импортируем сессию, модель и функцию инициализации базы.
* CORSMiddleware — посредник, который разрешает кросс-доменные запросы из браузера.

****

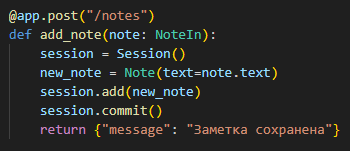
* Создаем экземпляр приложения app.
* Вызываем init\_db(), чтобы создать таблицы в базе, если их еще нет (важно запускать до работы с данными).

****

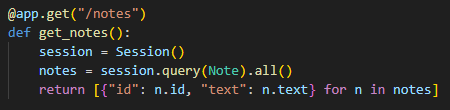
* Определяем модель данных NoteIn, которая описывает структуру входящего запроса на добавление заметки.
* В ней есть только одно поле text типа строка.
* Pydantic автоматически проверит, что в запросе есть это поле нужного типа.

****

* Подключаем посредник CORS (Cross-Origin Resource Sharing), который позволяет браузеру делать запросы к нашему API с фронтенда, запущенного на localhost:3000.
* Здесь разрешены все методы и заголовки, а также поддержка cookies и авторизации.

****

* Эндпоинт для добавления новой заметки.
* Принимает POST-запрос с телом, валидированным по модели NoteIn.
* Создается новая сессия, формируется объект заметки Note.
* Запись добавляется в базу с помощью session.add() и сохраняется вызовом session.commit().
* Возвращается простое сообщение о результате операции.

****

* Эндпоинт для получения списка всех заметок.
* Создается сессия, из базы извлекаются все записи из таблицы notes.
* Результат возвращается в формате списка словарей, где каждая заметка содержит id и text.
* Такой формат удобен для сериализации в JSON и передачи на фронтенд.

В этом коде реализован минимальный REST API для работы с заметками с использованием FastAPI и SQLAlchemy. Он показывает, как описывать модели данных для входящих запросов, как работать с сессиями базы, и как настраивать CORS для взаимодействия с фронтендом. Такой подход помогает понять базовые принципы построения серверной части веб-приложений и взаимодействия с базой данных.

**7. Проверка**

1. Запустить сервер:



1. Открыть <http://localhost:8000/docs>
2. Проверить:
   * **POST /notes** — добавить заметку;
   * **GET /notes** — получить список.