Паспорт проекта QuestionMark

Краткое описание:

Бот позволяет пользователям создавать вопросы с несколькими вариантами ответов, отвечать на вопросы других пользователей, просматривать статистику ответов, а также ставить лайки, дизлайки и жаловаться на вопросы. Реализована система ролей (незарегистрированный, зарегистрированный, модератор), хранение данных в SQLite через SQLAlchemy, ведение журнала событий и ошибок.

Необходимые библиотеки

telebot - для работы с Telegram API.

python-telegram-bot — для асинхронного взаимодействия с Telegram (Application, MessageHandler, CommandHandler и др.).

sqlalchemy - для ORM и работы с базой данных SQLite.

logging - для логирования событий и ошибок.

datetime - для работы с датой и временем.

json — для хранения и загрузки вспомогательных данных (например, связей между пользователями и вопросами).

os (по необходимости) - для работы с файловой системой.

telegram - для объектов Telegram (например, ReplyKeyboardMarkup).

flask - для создания веб-сервиса.

schedule - для планирования периодических задач.

random - для перемешивания данных.

Общая структура кода по файлам

Файл 1 — main.py (DrAeyJ + Adamusas)

**Назначение:**  
Основная логика Telegram-бота: запуск, обработка команд, регистрация пользователей, взаимодействие с вопросами, ведение журнала.

**Основные компоненты:**

* Импорт библиотек и моделей.
* Настройка логирования (файлы логов для предупреждений и ошибок).
* Инициализация базы данных.
* Глобальные переменные для текущего режима, текущего вопроса и ответов.
* Асинхронные обработчики команд:
  + /start — запуск/перезапуск бота, регистрация пользователя, определение роли и соответствующих команд. (DrAeyJ)
  + /help — вывод справки и списка доступных команд в зависимости от роли пользователя. (Adamusas)
  + /get\_question — получение случайного вопроса, формирование клавиатуры для ответов. (DrAeyJ)
  + /add\_question — добавление нового вопроса (реализуется отдельно). (Adamusas)
  + /moderate — модерация вопросов (для модераторов). (Adamusas)
* Функции для регистрации пользователя и работы с JSON-файлами связей.
* Обработка ответов на вопросы, подсчет статистики, лайков/дизлайков/жалоб.
* Обработка ошибок и исключений с логированием.

Файл 2 — \_\_all\_models.py (DrAeyJ)

**Назначение:**  
Описание ORM-моделей для пользователей и вопросов.

**Основные компоненты:**

* Импорт datetime и SQLAlchemy.
* Класс User:
  + id, telegram\_id, имя, тип (роль), счетчики активности, дата окончания блокировки.
* Класс Questions:
  + id, author\_tg\_id, содержимое, варианты ответов (до 4), счетчики по каждому ответу, лайки, дизлайки, жалобы.
* Методы **repr**для удобного вывода информации о моделях.

Файл 3 - db\_session.py (Adamusas)

**Назначение:**  
Инициализация и создание сеансов SQLAlchemy для работы с базой данных.

**Основные компоненты:**

Импорт SQLAlchemy.

Создание базового класса моделей.

Глобальная фабрика сессий.

Функция global\_init(db\_file) — инициализация базы данных, создание таблиц.

Функция create\_session() - получение новой сессии для работы с БД.

Файл 4 - logger\_filter.py (DrAeyJ)

**Назначение:**  
Расширение стандартной системы ведения журнала Python: фильтрация сообщений по уровню ведения журнала и расширенное форматирование исключений.

**Основные компоненты:**

Класс MaxLevelFilter(logging.Filter) — фильтр, пропускающий только сообщения с уровнем не выше заданного (например, чтобы записывать предупреждения в один журнал, а ошибки — в другой).

Класс ExceptionFormatter(logging.Formatter) — это форматтер, который добавляет дополнительные отступы к трассировке исключения для лучшей читаемости в логах.

**Использование:**  
Эти классы используются в main.py при настройке ведения журнала, чтобы разделять логи по уровням и улучшать читаемость ошибок.

Файл 5 — remote\_database\_controller.py (DrAeyJ)

**Назначение:**  
Веб-сервис на Flask для удалённого управления ролями пользователей в базе данных приложения.

**Основные компоненты:**

Инициализация приложения Flask и подключение к базе данных с помощью модуля db\_session.

Импорт моделей пользователей из модуля \_\_all\_models.

Определение маршрута /change\_user\_type///, который принимает параметры: user\_type, telegram\_id, admin\_passcode.

Проверка корректности кода администратора и допустимости нового типа пользователя.

Изменение типа пользователя в базе данных и фиксация изменений.

Возврат ответа в формате JSON с подтверждением или ошибкой.

Функция check(code\_given) для проверки переданного кода администратора на основе файла ../db/code.txt.

Запуск Flask-приложения.

Код администратора хранится в открытом виде в текстовом файле, что может быть уязвимостью для безопасности.

Файл 6 — code\_shuffler.py (Адамусас)

**Назначение:**  
Автоматическая регулярная смена кода администратора для повышения безопасности доступа к функциям администрирования.

**Основные компоненты:**

* Импорт модулей datetime, random, schedule.
* Глобальная переменная last\_update для отслеживания времени последнего обновления.
* Функция shuffle():
  + Считывает текущий код администратора из файла ../db/code.txt.
  + Перемешивает символы кода случайным образом.
  + Записывает новый (перемешанный) код обратно в файл и добавляет к нему временную метку.
  + Выводит в консоль время и длину нового кода.
* Планирование автоматического вызова функции shuffle() каждые 3600 секунд (1 час) с помощью библиотеки schedule.
* Бесконечный цикл, в котором выполняются запланированные задачи.