Тестовое задание. Backend, Python

Необходимо реализовать асинхронное веб приложение в парадигме REST API.

Время выполнения задачи 5 дней.

Стек:

- База данных - postgresql
- sqlalchemy - для работы с базой данных
- sanic - веб фреймворк(рекомендуемый, допускается альтернативный веб фрейморк НО НЕ DJANGO)
- docker compose

Необходимо реализовать работу со следующими сущностями:

1. Пользователь
2. Администратор
3. Счет - имеет баланс, привязан к пользователю
4. Платеж(пополнение баланса) - хранит уникальный идентификатор и сумму пополнения счета пользователя

Пользователь должен иметь следующие возможности:

1. Авторизоваться по email/password
2. Получить данные о себе(id, email, full_name)
3. Получить список своих счетов и балансов
4. Получить список своих платежей

Администратор должен иметь следующие возможности:

1. Авторизоваться по email/password
2. Получить данные о себе (id, email, full_name)
3. Создать/Удалить/Обновить пользователя
4. Получить список пользователей и список его счетов с балансами

Для работы с платежами должен быть реализован роут эмулирующий обработку вебхука от сторонней платежной системы.
Структура json-объекта для обработки вебхука должна состоять из следующих полей:

- `transaction_id` - уникальный идентификатор транзакции в “сторонней системе”
- `account_id` - уникальный идентификатор счета пользователя
- `user_id` - уникальный идентификатор счета пользователя

- `amount` - сумма пополнения счета пользователя
- `signature` - подпись объекта

`signature` должна формироваться через `SHA256` хеш, для строки состоящей из конкатенации значений объекта в алфавитном порядке ключей и “секретного ключа” хранящегося в конфигурации проекта (`{account_id}{amount}{transaction_id}{user_id}{secret_key}`).

Пример, для secret_key `gfdmhghif38yrf9ew0jkf32`:

```
{
  "transaction_id": "5eae174f-7cd0-472c-bd36-35660f00132b",
  "user_id": 1,
  "account_id": 1,
  "amount": 100,
  "signature":
"7b47e41efe564a062029da3367bde8844bea0fb049f894687cee5d57f2858bc8"
}
```

При обработке вебхука необходимо:

1. Проверить подпись объекта
2. Проверить существует ли у пользователя такой счет - если нет, его необходимо создать
3. Сохранить транзакцию в базе данных
4. Начислить сумму транзакции на счет пользователя

Транзакции являются уникальными, начисление суммы с одним transaction_id должно производиться только один раз.

Для тестирования приложения в миграции должен быть создан:

1. Тестовый пользователь
2. Счет тестового пользователя
3. Тестовый администратор

Для развертывания проекта необходимо реализовать docker compose конфигурацию состоящую из сервиса postgresql и сервиса приложения.
К реализованному заданию должна прилагаться краткая инструкция по запуску проекта в двух вариантах - с использованием docker compose и без него. В инструкции также должны быть предоставлены email/password для пользователя и администратора по умолчанию созданных в миграции.