Вебинар №3

Фреймворк – архитектурный каркас приложения, включающий набор готовых решений, в виде функций, модулей или подсистем. Например, подсистема регистрации и авторизации пользователей.

- Помогают быстро начать разработку программной системы.
- Упрощают взаимодействие с базой данных.
- Реализуют шаблоны проектирования.
- Повышают степень повторного использования кода.
- Обладают документацией и поддержкой сообщества.

- Универсальный и мощный фреймворк
- Обладает подробной документацией и развитым сообществом
- Имеет свою ORM
- Обладает встроенной панелью управления сайтом

В Django присутствует каталог на сотни плагинов, которые решают задачи от настройки sitemap до развёртывания полноценной CMS и создания REST API.

Помимо предусмотренных разработчиками фреймворка решений, достаточно просто подключаются аналоги (например Jinja2 вместо стандартного шаблонизатора Django). Хотя заменить, например, ORM достаточно тяжело.

«Из коробки» встроены ORM, шаблонизатор, мультиязычность, админ-панель, автоматическая документация и так далее.

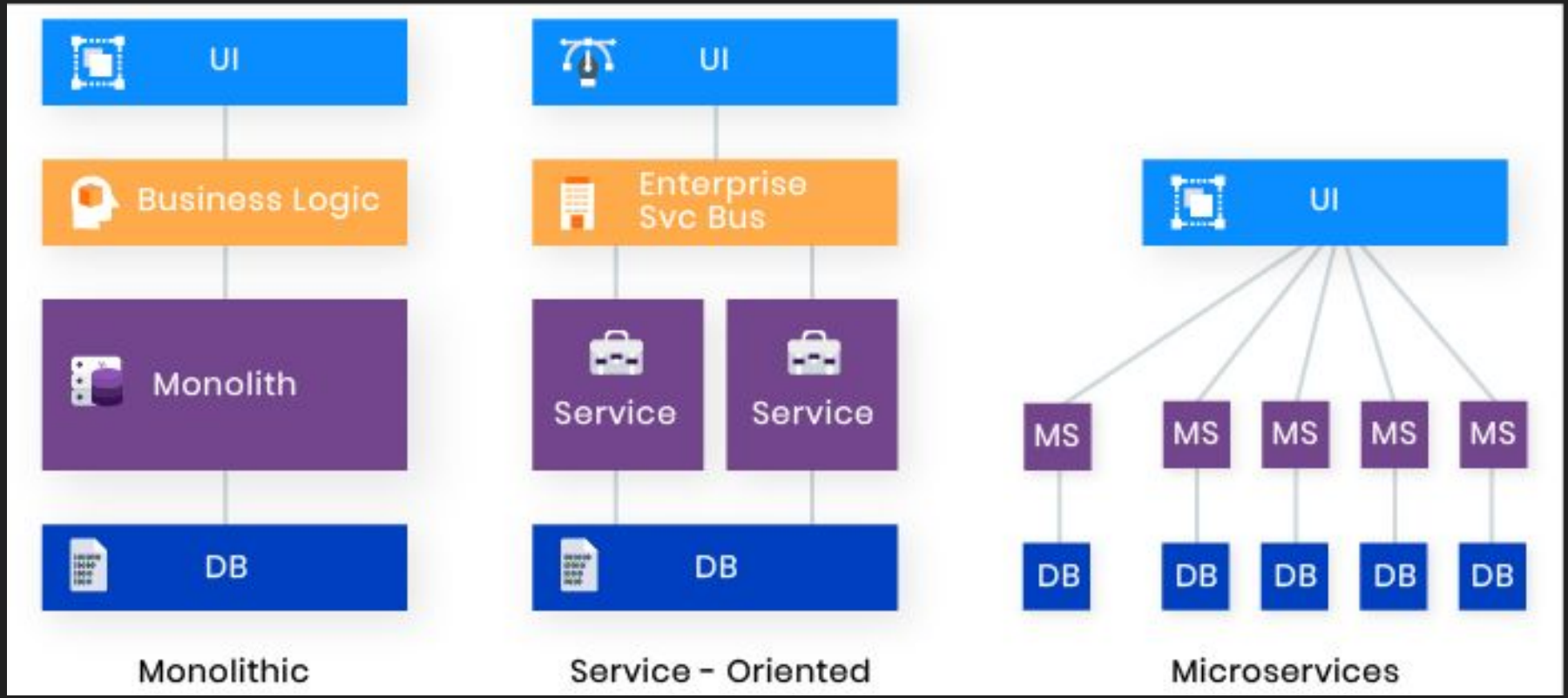
- Как Django предоставляет сервер разработки
- Как Django обладает подробной документацией и развитым сообществом
- Не настолько тяжелый как Django
- Возможность разработки веб-сайтов с использованием шаблонизатора (Jinja2)

Flask — это скелет, на который разработчик может навесить любой удобный для него инструментарий. Причём этот инструментарий не навязывается фреймворком, как в случае с Django. Например, выбирая Flask для проекта, разработчик волен в выборе ORM. Как правило, выбор падает на SQLAlchemy.

FastAPI

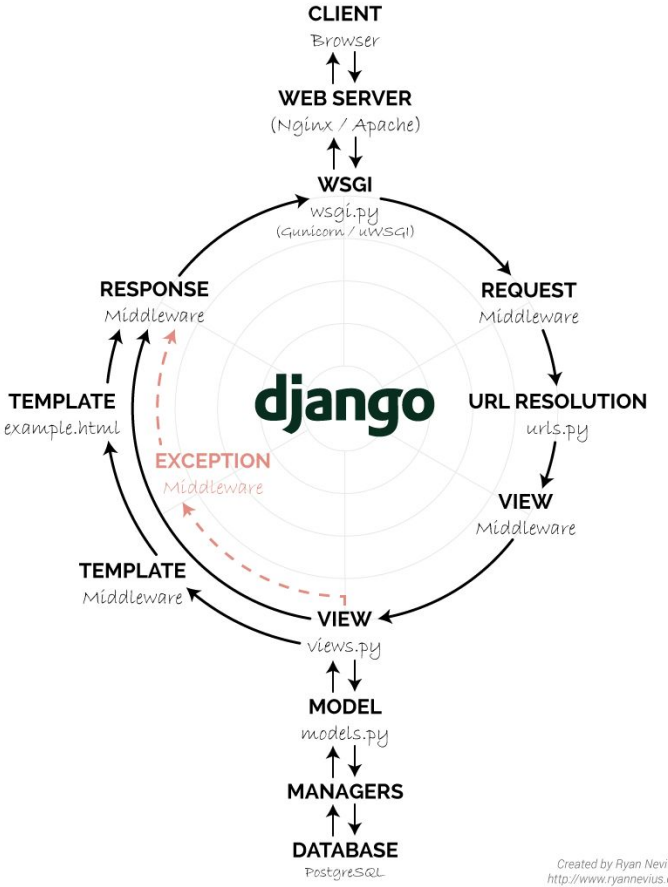
FastAPI — фреймворк Python для лёгкого создания API-серверов со встроенными валидацией, сериализацией и асинхронностью. Стоит он на плечах двух других фреймворков. Работой с web в FastAPI занимается Starlette, за валидацию отвечает Pydantic.

- Встроенная асинхронность
- Встроенные фоновые задачи и веб-сокеты
- Возможность из коробки работать с GraphQL
- Авто-документация



https://habr.com/ru/company/otus/blog/477930/

django
Request-Response Cycle



Created by Ryan Nevius
http://www.ryannevius.com

Идентификация, аутентификация и авторизация – три процесса защищающие Ваши данные или денежные средства от доступа посторонних лиц.

Понимание процессов придет быстрее, если дать им определения.

- Идентификация — процесс распознавания пользователя по его идентификатору.
- Аутентификация — процедура проверки подлинности, доказательство что пользователь именно тот, за кого себя выдает.
- Авторизация — предоставление определенных прав.

Способ авторизации заключается в создании токена, корректность которого сможет проверить ваш проект.

JWT расшифровывается как JSON Web Token. Такой токен легко узнать — это строка, разделенная на три части двумя точками:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiODA5OTEwMDIwMzAiLCJpYXQiOjEyMDE2OTIzNzcsImV4cCI6MTUxNjIzOTAyMiwicm9sZSI6InN1YnNjcmliZXIifQ.wnN9e4rPDxVSReh9Qvigkmly2aQ_PNwQ-MNNpdsixgA

1) Часть до первой точки — это заголовок токена или Header. Он состоит из JSON, закодированного при помощи Base64url. Там вы найдёте два параметра: тип токена (JWT) и алгоритм хеширования, который следует использовать для проверки подлинности токена.

2) Второй и самый длинный участок — это передаваемая информация или Payload.Никаких договоренностей о том, что должна содержать эта часть, нет. Вы можете описывать в ней любые поля, необходимые вашей бизнес-логике. Но в стандарте JWT описано несколько служебных ключей, которые могут быть вам полезны. В примере выше это ключи `iat` и `exp`, которые обозначают дату создания и срок действия токена.

3) Третья часть — цифровая подпись токена. Она генерируется на основе первых двух частей токена, зашифрованных с секретным ключом. Как вы заметили, данные в JWT находятся в закодированном, но незашифрованном виде. При отсутствии подписи любой человек мог бы попытаться отправить вашему сервису токен с подмененным логином или попробовать «повысить» уровень прав. Именно подпись позволяет сервису, которому отправили запрос с токеном, убедиться, что данные не были модифицированы злоумышленником и дошли до сервиса в неизменном виде.

Финальное задание

Надо написать логику авторизаций пользователей на основе JWT-токенов. К существующему проекту: https://github.com/BernarBerdikul/ylab_hw/tree/main/webinar_num_3.

- В качестве SQL DB: использовать Postgres.
- В качестве NoSQL DB: использовать Redis.
- В качестве Фреймворка: использовать FastAPI.

Обращайте внимание на HTTP коды ответов и тела ответов.

К презентаций будет приложена коллекция Postman. Ориентируясь на запросы в коллекций спроектировать авторизацию.

Анонимный пользователь может:

- Создать аккаунт, если выбранный username еще не зарегистрирован в системе.
- Войти в свой аккаунт по username и паролю.
- Просмотреть список постов.
- Просмотреть детально определенный пост.

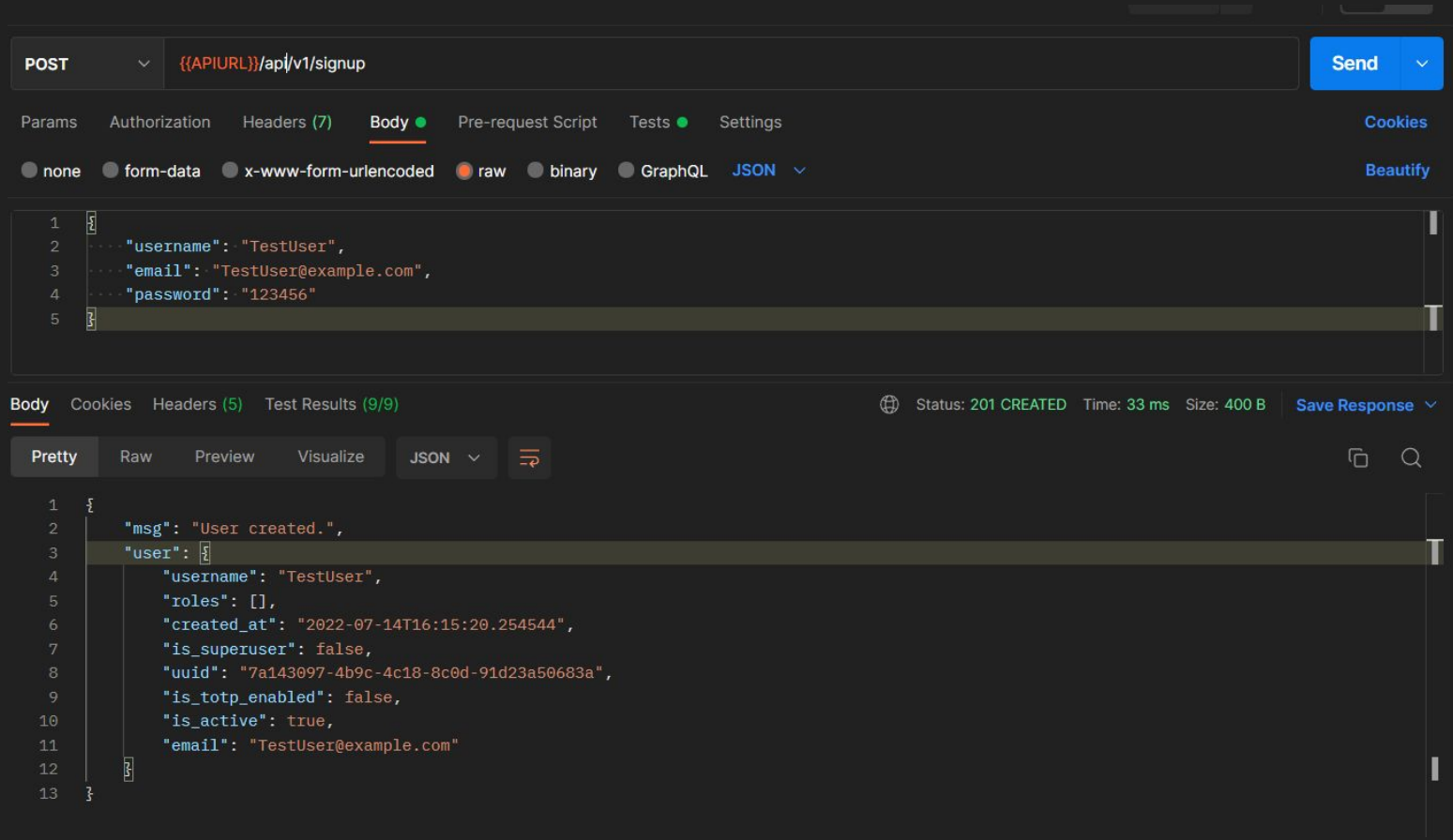
Авторизованный пользователь может:

- Просмотреть свой профиль.
- Создать пост.
- Изменить свои личные данные — email, username и др. И получить новый access_token

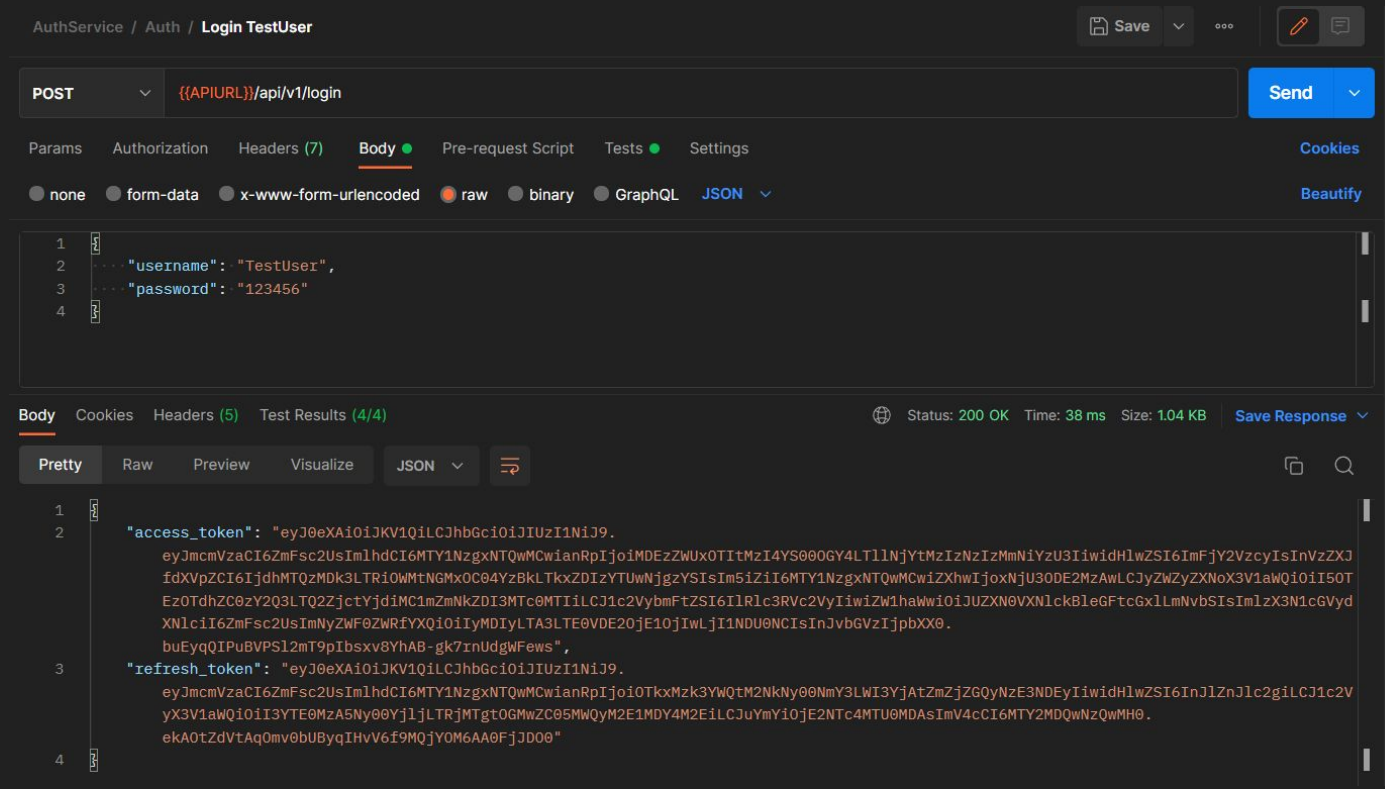
Задание со звездочкой:

- Валидировать данные при регистраций и обновлений данных пользователя
- Хранить в Redis заблокированные access_tokens (удобно хранить это в разных базах)
- Хранить в Redis активные refresh_tokens для определенного пользователя (удобно хранить это в разных базах)
- Добавить метод выхода из аккаунта (на этом этапе надо добавить access_token в список заблокированных токенов в Redis, удалить refresh_token из списка активных токенов в Redis определенного пользователя)
- Выйти со всех устройств. (удалить все активные refresh_token для определенного пользователя)

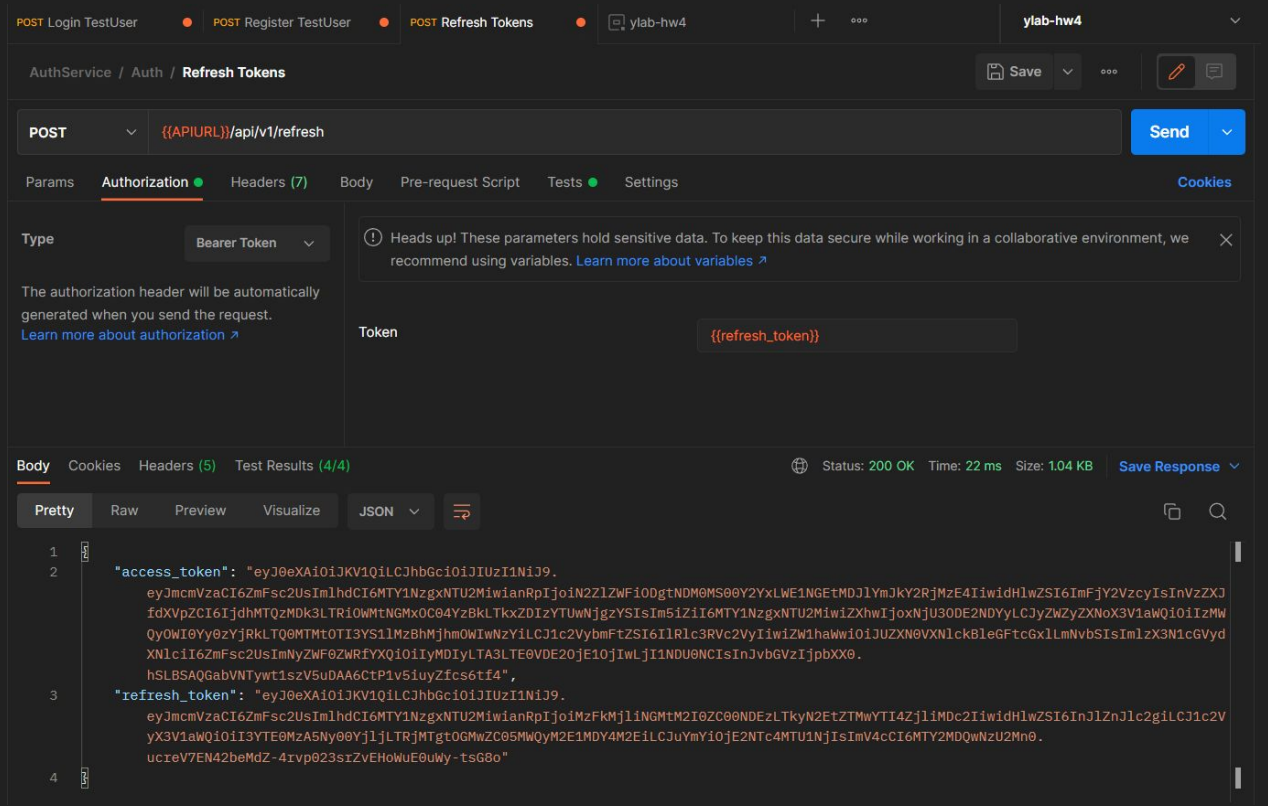
Регистрируется на сайте



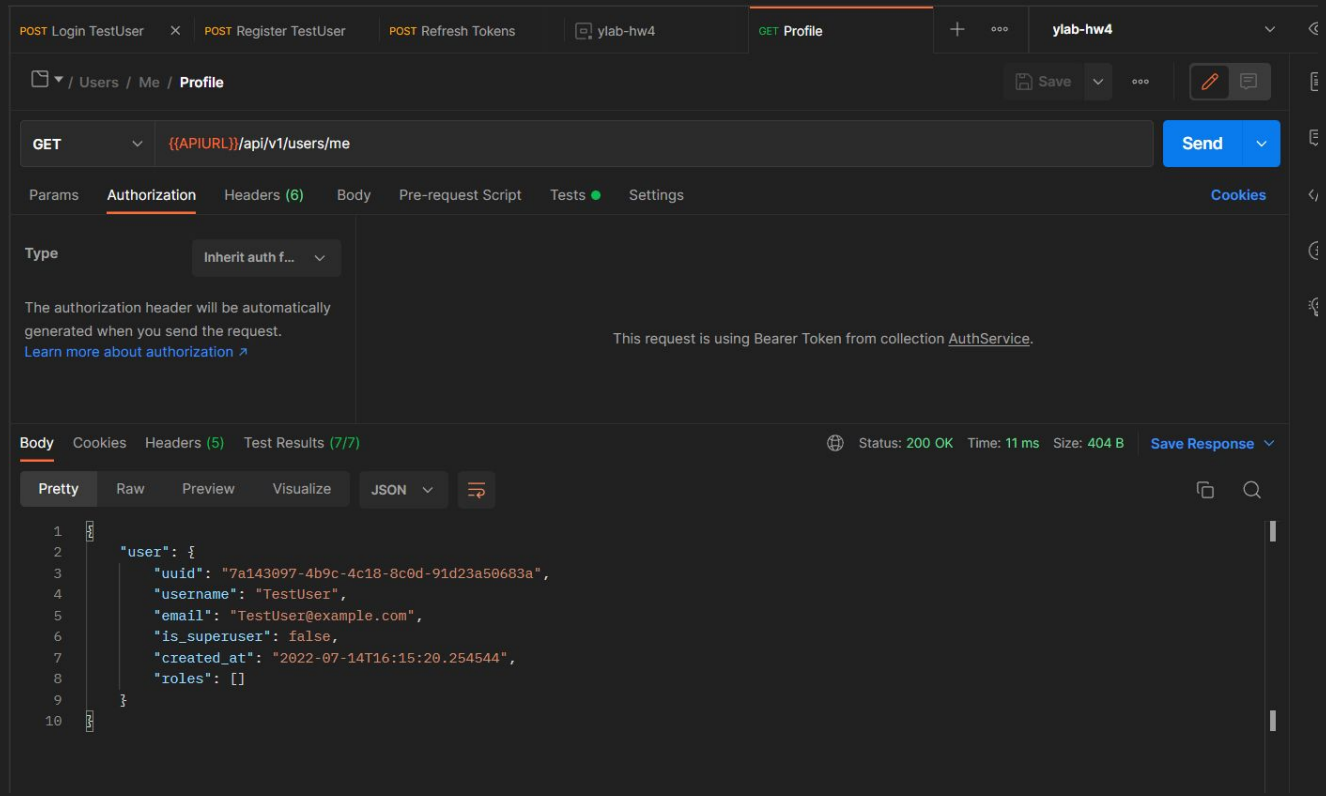
Заходит на сайт



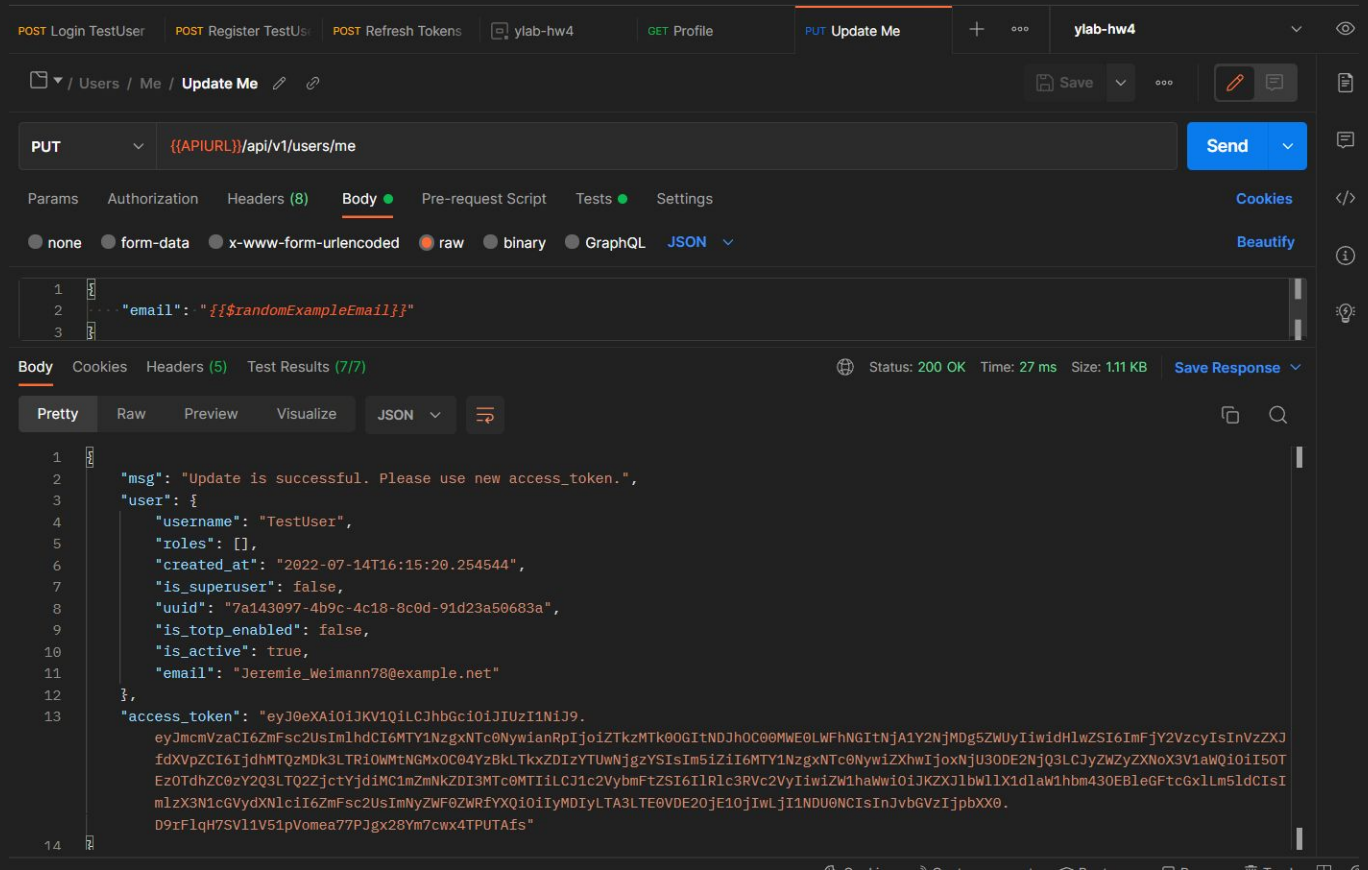
Обновляет токен



Смотрит свои профиль

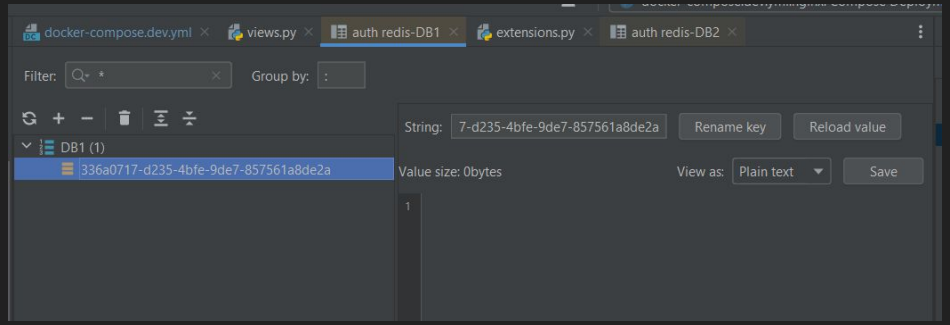


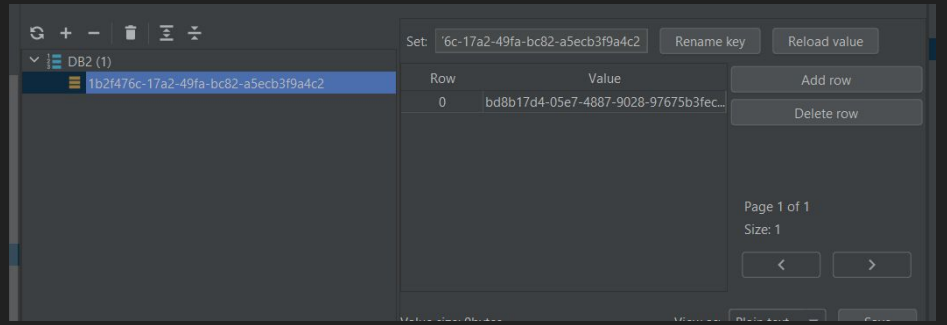
Обновляет информацию о себе



```python
blocked_access_tokens = redis.Redis(
    host=settings.redis.host, port=settings.redis.port, db=1, decode_responses=True,
)
active_refresh_tokens = redis.Redis(
    host=settings.redis.host, port=settings.redis.port, db=2, decode_responses=True,
)
```

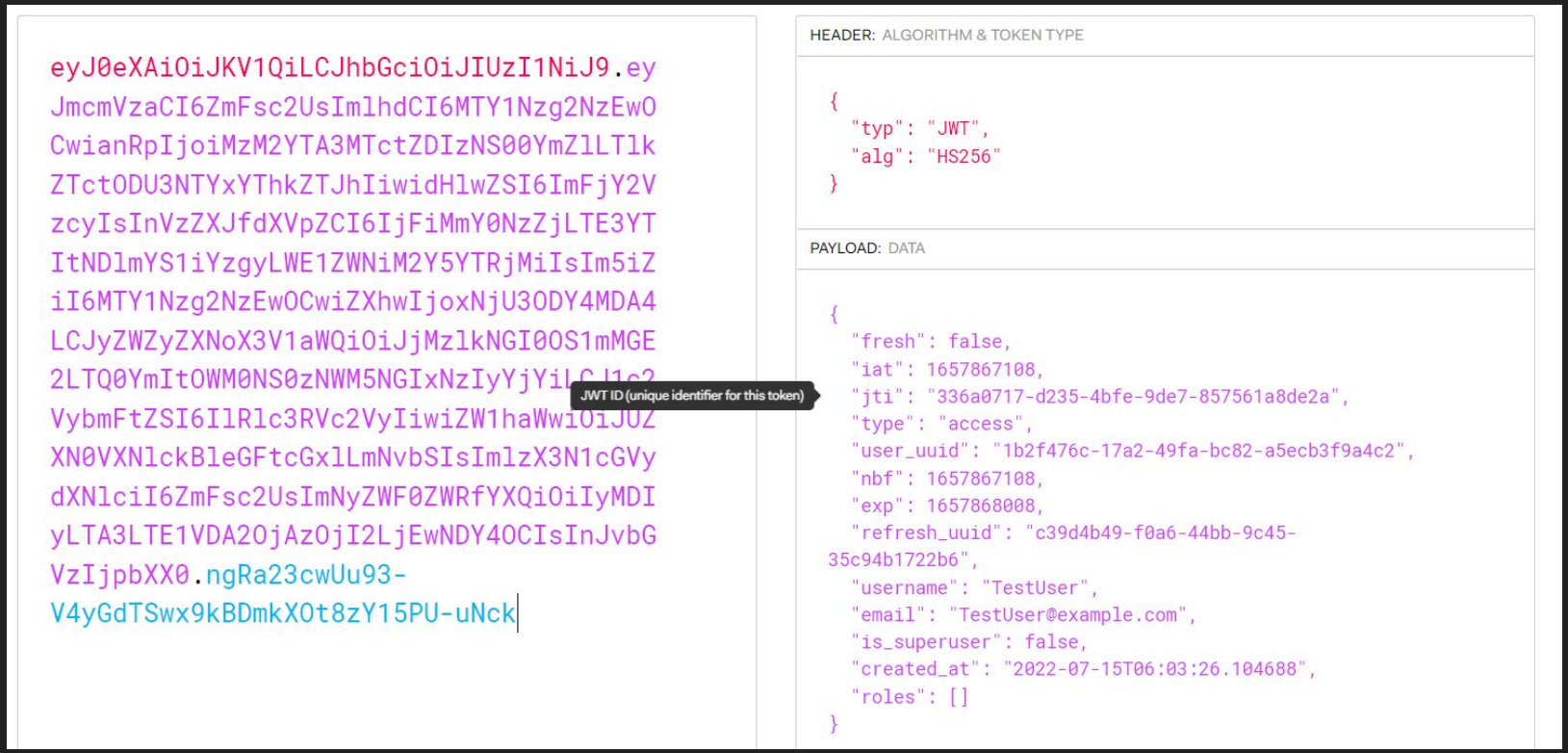
Настройки для Redis, 2 БД
1) для заблокированных access_token
2) для активных refresh_token





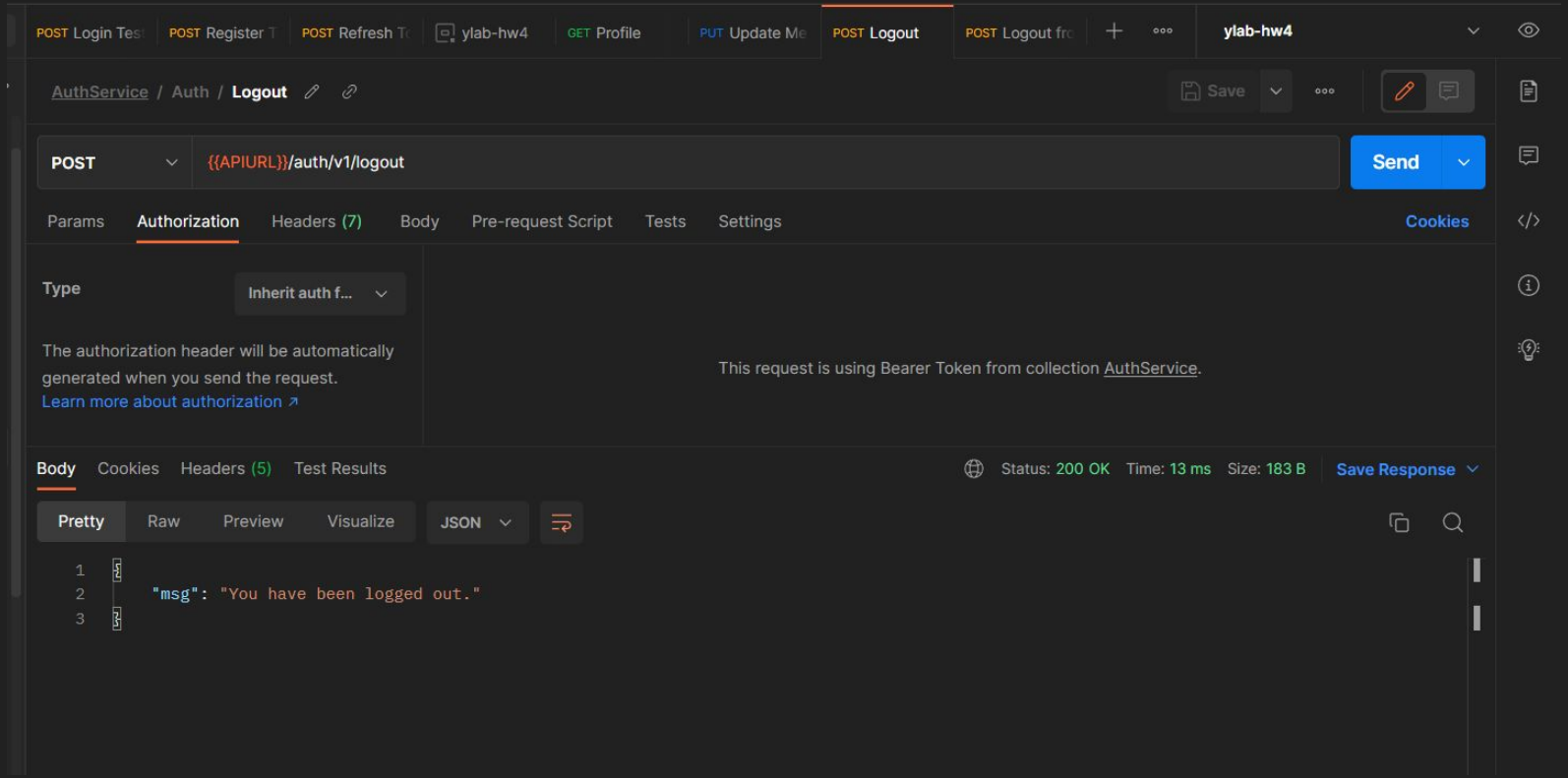
В первой БД храним только JWT ID заблокированных access_token

Храним {user_id: [JWT_ID1, JWT_ID2, …]} для конкретного пользователя список его активных refresh_token

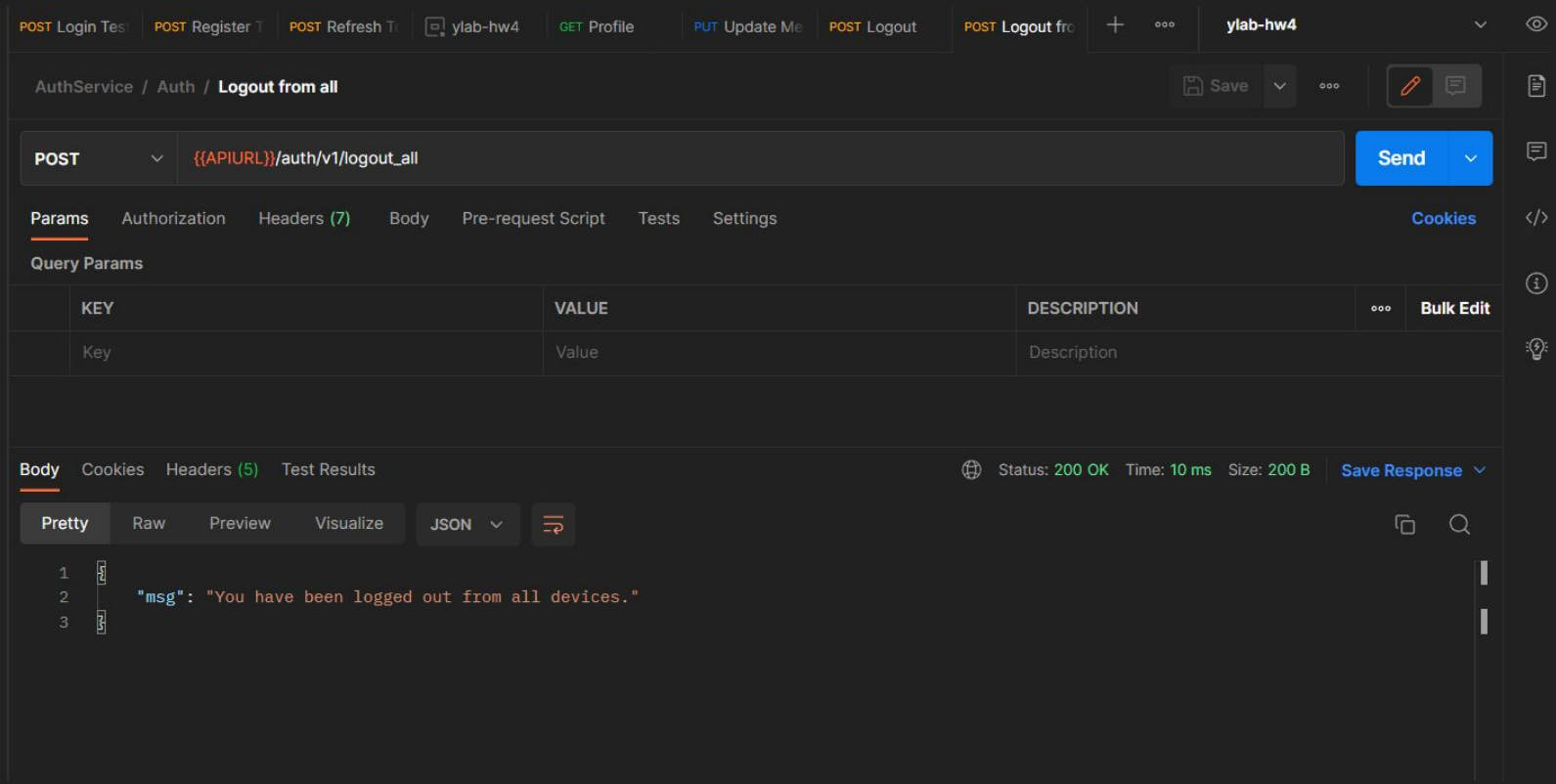


У каждого токена есть uuid (уникальный JWT ID) именно его мы храним в Redis

Выйти с одного устройства



Выйти со всех устройств



Ссылки для изучения

- https://university.ylab.site/tools/microservices/ (Микросервисы)
- https://university.ylab.site/tools/json/ (JSON)
- https://university.ylab.site/tools/docker/ (Docker)
- https://university.ylab.site/tools/docker-compose/ (Docker-compose)
- https://university.ylab.site/tools/docker-data-management/ (Управление данными в Docker)
- https://university.ylab.site/tools/dockerfile/ (DockerFile)
- https://university.ylab.site/tools/http-protocol/ (HTTP)
- https://university.ylab.site/js/web-app-types/ (Виды Веб приложения)
- https://university.ylab.site/js/web-app-works/ (Как работают Веб приложения)
- https://university.ylab.site/js/api/ (API)
- https://fastapi.tiangolo.com/ (FastApi)
- https://pydantic-docs.helpmanual.io/ (pydantic)
- https://www.sqlalchemy.org/ (sqlalchemy)
- https://jwt.io/ (JWT)