

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.



django



- 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.



NETFLIX



UBER

- 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)

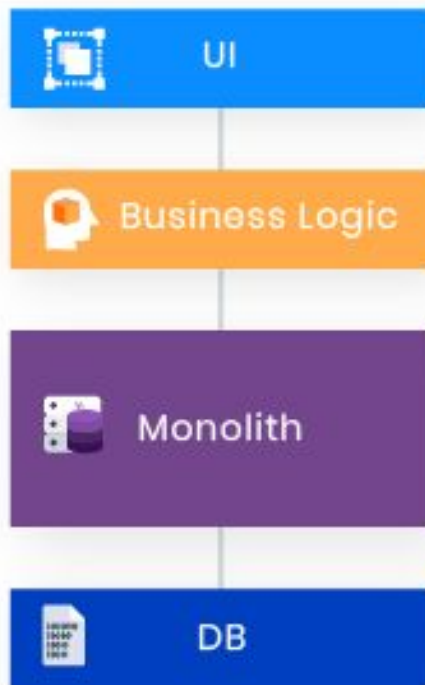
Flask — это скелет, на который разработчик может навесить любой удобный для него инструментарий. Причём этот инструментарий не навязывается фреймворком, как в случае с Django. Например, выбирая Flask для проекта, разработчик волен в выборе ORM. Как правило, выбор падает на SQLAlchemy.



THINKOFF

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.

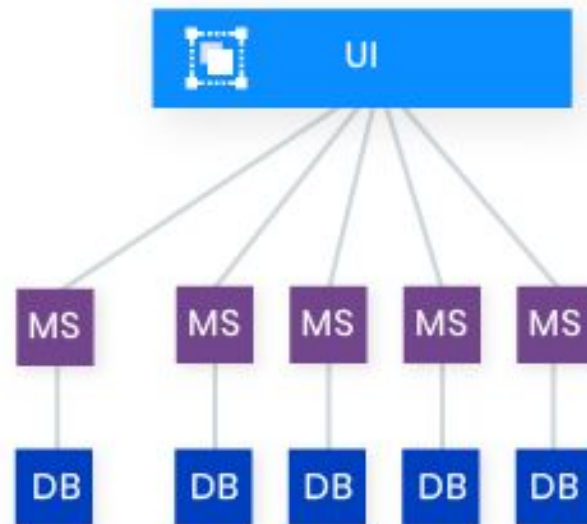
- Встроенная асинхронность
- Встроенные фоновые задачи и веб-сокеты
- Возможность из коробки работать с GraphQL
- Авто-документация



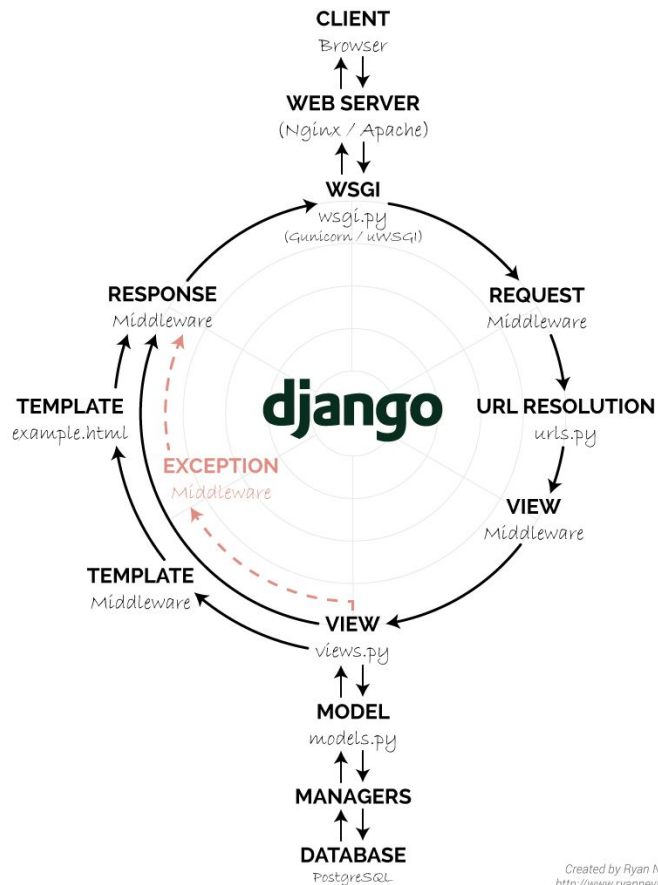
Monolithic



Service - Oriented



Microservices



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.eyJ1c2VyIjoiaDA5OTEwMDIwMzAiLCJpYXQiOiJyMDE2OTIzNzcsImV4cCI6MTUxNjIzOTAyMiwiYm9sZSI6InN1YnNjcmlZlXlifQ.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 договорённостей 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

POST

{{APIURL}}/api/v1/signup

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

1

2

3

4

5

Body

Cookies

Headers (5)

Test Results (9/9)

Status: 201 CREATED

Time: 33 ms

Size: 400 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

1

2

3

4

5

6

7

8

9

10

11

12

13

```
{
  "msg": "User created.",
  "user": {
    "username": "TestUser",
    "roles": [],
    "created_at": "2022-07-14T16:15:20.254544",
    "is_superuser": false,
    "uuid": "7a143097-4b9c-4c18-8c0d-91d23a50683a",
    "is_totp_enabled": false,
    "is_active": true,
    "email": "TestUser@example.com"
  }
}
```

Заходит на сайт

Обновляет токен

Смотрит свой профиль

POST Login TestUser

POST Register TestUser

POST Refresh Tokens

ylab-hw4

GET Profile

+

...

ylab-hw4

▼

⌵

▼ / Users / Me / Profile

Save

▼

...

GET

▼

{{APIURL}}/api/v1/users/me

Send

▼

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests ●

Settings

Cookies

Type

Inherit auth f...

▼

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

This request is using Bearer Token from collection [AuthService](#).

Body

Cookies

Headers (5)

Test Results (7/7)

🌐 Status: 200 OK Time: 11 ms Size: 404 B

Save Response

▼

Pretty

Raw

Preview

Visualize

JSON

▼

```
1  {
2    "user": {
3      "uuid": "7a143097-4b9c-4c18-8c0d-91d23a50683a",
4      "username": "TestUser",
5      "email": "TestUser@example.com",
6      "is_superuser": false,
7      "created_at": "2022-07-14T16:15:20.254544",
8      "roles": []
9    }
10 }
```

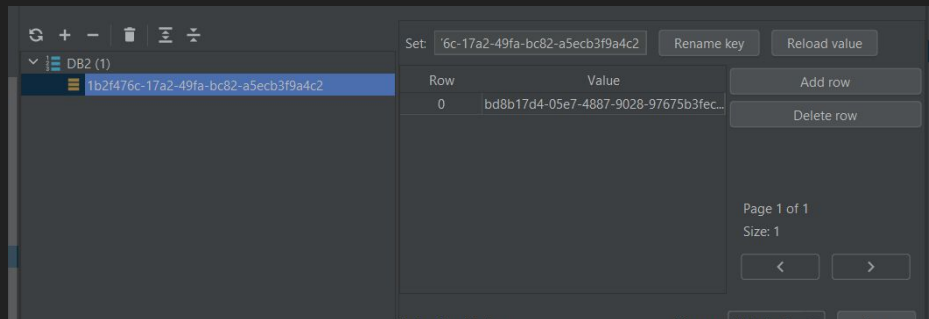
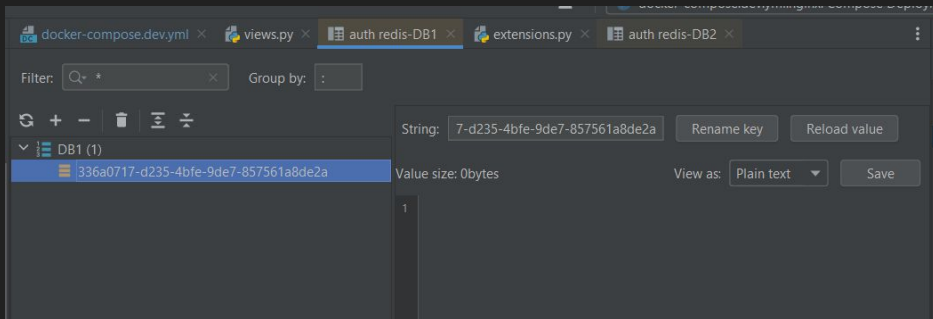
Обновляет информацию о себе


```

17 blocked_access_tokens = redis.Redis(
18     host=settings.redis.host, port=settings.redis.port, db=1, decode_responses=True,
19 )
20 active_refresh_tokens = redis.Redis(
21     host=settings.redis.host, port=settings.redis.port, db=2, decode_responses=True,
22 )

```

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**

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsIm1hdCI6MTY1Nzg2NzEwO  
CwianRpijoiMzM2YTA3MTctZDIzNS00YmZlLTlkZTctODU3NTYxYThkZTJhIiwidHlwZSI6ImFjY2V  
zcyIsInVzZXJfdXVpZCI6IjFiMmY0NzZjLTE3YTItNDlmYS1iYzgyLWE1ZWNiM2Y5YTRjMiIsIm5iZ  
iI6MTY1Nzg2NzEwOCwiZXhwIjoxNjU3ODY4MDA4LCJyZWZyZXNoX3V1aWQiOiJjMzlkNGI0S1mMGE  
2LTQ0YmItOWM0NS0zNWw5NGIxNzIyYjYiLCJ1c2VybmFtZSI6IiRlc3Rvc2VyIiwiaWwiZW1haWwiOiJUZ  
XN0VXNlckBleGFtcGx1LmNvbSIsIm1zX3N1cGVydXNlciI6ZmFsc2UsImNyZWZlZWRfYXQiOiIyMDI  
yLTA3LTE1VDA2OjAzOjI2LjEwNDY4OCIsInJvbGZvZjpbXX0.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsIm1hdCI6MTY1Nzg2NzEwO  
CwianRpijoiMzM2YTA3MTctZDIzNS00YmZlLTlkZTctODU3NTYxYThkZTJhIiwidHlwZSI6ImFjY2VzcyIsInVzZXJfdXVpZCI6IjFiMmY0NzZjLTE3YTItNDlmYS1iYzgyLWE1ZWNiM2Y5YTRjMiIsIm5iZiI6MTY1Nzg2NzEwOCwiZXhwIjoxNjU3ODY4MDA4LCJyZWZyZXNoX3V1aWQiOiJjMzlkNGI0S1mMGE2LTQ0YmItOWM0NS0zNWw5NGIxNzIyYjYiLCJ1c2VybmFtZSI6IiRlc3Rvc2VyIiwiaWwiZW1haWwiOiJUZ
```

JWT ID (unique identifier for this token)

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

PAYLOAD: DATA

```
{  
  "fresh": false,  
  "iat": 1657867108,  
  "jti": "336a0717-d235-4bfe-9de7-857561a8de2a",  
  "type": "access",  
  "user_uuid": "1b2f476c-17a2-49fa-bc82-a5ecb3f9a4c2",  
  "nbf": 1657867108,  
  "exp": 1657868008,  
  "refresh_uuid": "c39d4b49-f0a6-44bb-9c45-35c94b1722b6",  
  "username": "TestUser",  
  "email": "TestUser@example.com",  
  "is_superuser": false,  
  "created_at": "2022-07-15T06:03:26.104688",  
  "roles": []  
}
```

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

Выйти с одного устройства

The screenshot shows a REST client interface with a dark theme. At the top, a list of requests includes 'POST Login Test', 'POST Register T', 'POST Refresh T', 'ylab-hw4', 'GET Profile', 'PUT Update Me', 'POST Logout' (which is selected), and 'POST Logout fr'. The breadcrumb path is 'AuthService / Auth / Logout'. The request method is 'POST' and the URL is '{{APIURL}}/auth/v1/logout'. The 'Authorization' tab is active, showing a note about Bearer tokens and a link to learn more. The 'Body' tab is also active, displaying a JSON response in 'Pretty' format:

```
{  "msg": "You have been logged out."}
```

. The status bar at the bottom indicates a '200 OK' status, 13 ms time, and 183 B size.

POST Login Test POST Register T POST Refresh T ylab-hw4 GET Profile PUT Update Me **POST Logout** POST Logout fr + ... ylab-hw4

AuthService / Auth / Logout Save ...

POST {{APIURL}}/auth/v1/logout Send

Params **Authorization** Headers (7) Body Pre-request Script Tests Settings Cookies

Type Inherit auth f...
The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

This request is using Bearer Token from collection [AuthService](#).

Body Cookies Headers (5) Test Results Status: 200 OK Time: 13 ms Size: 183 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "You have been logged out."
3 }
```

Выйти со всех устройств

POST Login TestPOST Register TestPOST Refresh Tokenylab-hw4GET ProfilePUT Update MePOST LogoutPOST Logout from all

AuthService / Auth / Logout from all

Save

Send

POST

{{APIURL}}/auth/v1/logout_all

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY	VALUE	DESCRIPTION		Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 10 ms

Size: 200 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "msg": "You have been logged out from all devices."
3 }
```

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)