Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Разработка сетевых приложений

Отчет по курсовой работе

Работу выполнил:
Коробейников Д.А.
Группа: 3530901/70202
Преподаватель:
Алексюк А.О.

Санкт-Петербург
2021

Содержание

1. Цель работы

Целью данной курсовой работы является разработка веб-приложения для обработки операций по постановке и снятию автомобилей с парковки.

2. Программа работы

- Перевести сервис, разработанный ранее на TCP, на протокол HTTP;
- Реализовать аутентификацию с помощью токенов (JWT);
- Документировать API с помощью средств спецификации OpenAPI;
- Выполнить развертывание разработанного ПО на сервере;
- Настроить шифрование передаваемых данных с помощью протокола TLS и сертификатов Let's Encrypt;
- Загрузить спецификацию API в Swagger UI.

3. Ход работы

3.1. Реализация на HTTP

В ходе данной работы парковочный сервис был реализован при помощи Flask. В качестве базы данных использовалась MongoDB. Данный сервис позволяет ставить или снимать с парковки автомобиль в режиме доступа "пользователь"(без аутентификации), либо просматривать лог сервера и испорию парковок, общую собранну сумму в режиме администратора (аутентификация на токенах). Для описания путей использовались декораторы app.route.

Листинг 1: Описание методов постановки и снятия с парковки

```
@app.route('/user/park', methods=['POST'])
@use_kwargs({'number': fields.Str()})
@marshal_with(AcceptSchema)
def handle_park(**kwargs):
    request_data = request.get_json()
    return request_handler.park(request_data)


@app.route('/user/unpark', methods=['POST'])
@use_kwargs({'number': fields.Str()})
@marshal_with(CheckoutSchema)
def handle_unpark(**kwargs):
    request_data = request.get_json()
    return request_handler.unpark(request_data)
```

3.2. Аутентификация

Для аутентификации использовался фреймворк flask_jwt_extended. Чтобы ограничить доступ к путям, с которыми пользователи не должны взаимодействать без аутентификации, ипользовались декораторы jwt_required.

Листинг 2: Пример метода с аутентификацией

```
@jwt_required()
def handle_history(**kwargs):
    return request_handler.history()
```

3.3. Документирование OpenAPI

Для создания автоматической документации использовался фреймворк flask_apispec. Для успешного автоматичекскиого документирования, необходимо определить схемы запросов и ответов, после чего использовать соответствующие декораторы перед каждым методом.

Листинг 3: Пример использования декораторов для автоматической документации

```
@doc(
    description='Token access',
    params={
        'Authorization': {
            'description':
            'Authorization: Bearer <jws-token>',
            'in': 'header',
            'type': 'string',
            'required': True
        }
    })
@app.route('/admin/total', methods=['GET'])
@marshal_with(TotalSchema)
@jwt_required()
```

3.4. Deployment

Развертывание ПО выполнялось на удаленном сервере с Ubuntu 20.04, в индивидуальном контейнере. В ходе развертывавания были настроены и запущены два сервиса, один для поддержки подключений по протоколу HTTP, другой для HTTPS.

Листинг 4: Сервис для HTTP

```
[Unit]
Description=Flask parking application daemon for http
After=network.target
After=mongod.service

[Service]
Type=simple
DynamicUser=true
WorkingDirectory=/var/lib/falsk-parking
ExecStart=/var/lib/falsk-parking/venv/bin/gunicorn \
    --bind 0.0.0.0:8012 wsgi:app \
    --access-logfile - \

[Install]
WantedBy=multi-user.target
```

Листинг 5: Сервис для HTTPS

```
[Unit]
Description=Flask parking application daemon
After=network.target
After=mongod.service

[Service]
Type=simple
DynamicUser=true
WorkingDirectory=/var/lib/falsk-parking
ExecStartPre=+/usr/bin/chown -R $USER \
```

```
11                /var/lib/falsk-parking/keys/littledima.duckdns.org.cer   \
12                /var/lib/falsk-parking/keys/littledima.duckdns.org.key \
13                /var/lib/falsk-parking/keys/ca.cer
14 ExecStart=/var/lib/falsk-parking/venv/bin/gunicorn \
15     --certfile=/var/lib/falsk-parking/keys/littledima.duckdns.org.cer   \
16     --keyfile= /var/lib/falsk-parking/keys/littledima.duckdns.org.key \
17     --ca-cert=  /var/lib/falsk-parking/keys/ca.cer \
18     --bind 0.0.0.0:8013 wsgi:app \
19     --access-logfile -
20
21 [Install]
22 WantedBy=multi-user.target
```

В настройках данных сервисов также указано, что они должны запускаться после запуска сервиса базы данных - mongod. В сервисе для HTTPS указаны пути сертификатов, получение которых будет описано в следующем разделе.

3.5. Настройка шифрования

Для настройки шифрования была использована утилита acme.sh. Были запрошены сертификаты для домена littledima.duckdns.org. В результате, после настройки сервиса для ssl-подключений, подключение по зашифрованному канало проходит успешно.

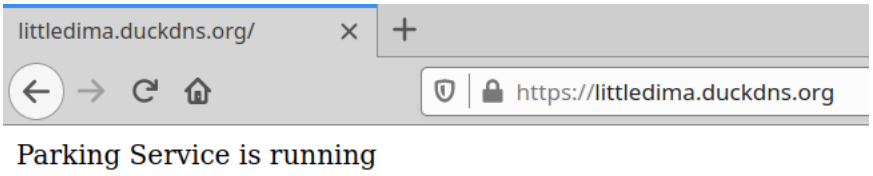

Рис. 3.1. Безопасное подключение

3.6. SwaggerUI

Фреймфорк, использованный для автоматического документирования, позволяет также встроить SwaggerUI в свой сервер. При переходе по адресу /api, отображается страница с доступными методами.

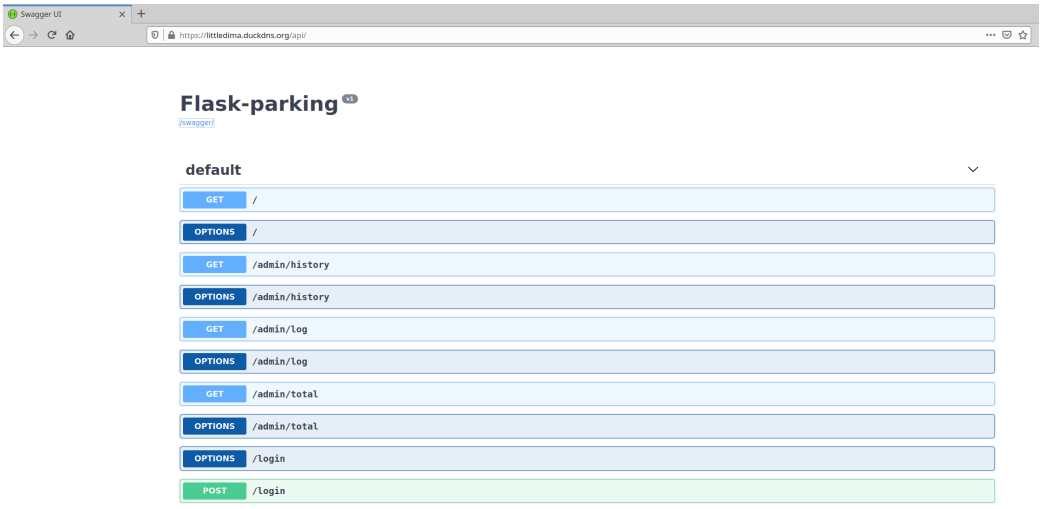

Рис. 3.2. SwaggerUI

Выводы

В ходе данной курсовой работы было реализовано веб-приложение на HTTP. Была реализована аутентификация с помощью токенов (JWT), документирован API с помощью средств спецификации OpenAPI, выполнено развертывание разработанного ПО на сервере, настроено шифрование передаваемых данных с помощью протокола TLS и сертификатов Let's Encrypt, продемонтсрирована работа SwaggerUI.