- JWT делает возможным предоставление одновременного доступа к различным доменам и сервисам. В нашем случае JWT-токены использованы по следующей схеме:

1. Клиент проходит аутентификацию в приложении (к примеру, с использованием логина и пароля).

2. В случае успешной аутентификации, сервер отправляет клиенту access- и refresh-токены.

3. При дальнейшем обращении к серверу, клиент использует access-токен. Сервер проверяет токен на валидность и предоставляет клиенту доступ к ресурсам

4. В случае, если access-токен перестает быть валидным, клиент отправляет refresh-токен, и в ответ получает два обновленных токена.

5. В случае, если refresh-токен перестает быть валидным, клиент снова должен пройти процесс аутентификации.

Решение использовано для доступа клиента к API сервиса на 24 часа без его повторной аутентификации, что ускорить обработку запросов. JWT-токены поддерживается всеми платформами. Именно за счет их универсальности и возможности избежать множественной процедуры аутентификации они были выбраны как основной механизм оптимизации доступа к API сервиса дипломного проекта.

М.Ю. Кравцов (ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **А.В. Воруев**, канд. техн. наук, доцент

PYTHON FLASK REST-СЕРВИС

Flask - это микрофреймворк для Python, основанный на Werkzeug и Jinja 2. Он позволяет быстро создавать небольшие веб-сервисы, используя свой гибкий функционал. Для написания REST-сервиса используется пакет flask-restful, который содержит необходимые инструменты для реализации CRUD-архитектуры приложения.

Для реализации CRUD-архитектуры в HTTP используются четыре вида запросов:

- GET – для чтения данных
- POST – для создания данных
- PUT или PATCH – для обновления данных
- DELETE – для удаления данных

Flask-restful позволяет создать модель данных, в которой можно определить ее поведение при обработке http-запросов, описанных

выше, а затем просто зарегистрировать эту модель в приложении. Так-же данный пакет имеет встроенный функционал для валидации входных данных и их форматирования, что значительно упрощает разработку сервиса и сокращает время на его создание.

Пример простого REST-сервиса, оперирующего задачами, написанного с использованием flask-restful:

```
from flask import Flask
from flask_restful import reqparse, abort, Api, Resource

app = Flask(__name__)
api = Api(app)

TODOS = {
    'todo1': {'task': 'build an API'},
    'todo2': {'task': '?????'},
    'todo3': {'task': 'profit!'},
}

def abort_if_todo_doesnt_exist(todo_id):
    if todo_id not in TODOS:
        abort(404, message="Todo {} doesn't ex-
ist".format(todo_id))

parser = reqparse.RequestParser()
parser.add_argument('task')


class Todo(Resource):
    def get(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        return TODOS[todo_id]

    def delete(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        del TODOS[todo_id]
        return '', 204

    def put(self, todo_id):
        args = parser.parse_args()
        task = {'task': args['task']}
        TODOS[todo_id] = task
        return task, 201

class TodoList(Resource):
    def get(self):
        return TODOS

    def post(self):
```

```
        args = parser.parse_args()
        todo_id = int(max(TODOS.keys()).lstrip('todo')) + 1
        todo_id = 'todo%i' % todo_id
        TODOS[todo_id] = {'task': args['task']}
        return TODOS[todo_id], 201

api.add_resource(TodoList, '/todos')
api.add_resource(Todo, '/todos/<todo_id>')


if __name__ == '__main__':
    app.run(debug=True)
```

Получение задачи:

```
$ curl http://localhost:5000/todos/todo3
{"task": "profit!"}
```

Создание новой задачи:

```
$ curl http://localhost:5000/todos -d "task=something new"
-X POST -v

> POST /todos HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0)
libcurl/7.19.7 OpenSSL/0.9.8l zlib/1.2.3
> Host: localhost:5000
> Accept: */*
> Content-Length: 18
> Content-Type: application/x-www-form-urlencoded
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 CREATED
< Content-Type: application/json
< Content-Length: 25
< Server: Werkzeug/0.8.3 Python/2.7.2
< Date: Mon, 01 Oct 2012 22:12:58 GMT
<
* Closing connection #0
{"task": "something new"}
```

Обновление задачи:

```
$ curl http://localhost:5000/todos/todo3 -d
"task=something different" -X PUT -v

> PUT /todos/todo3 HTTP/1.1
> Host: localhost:5000
> Accept: */*
> Content-Length: 20
> Content-Type: application/x-www-form-urlencoded
>
* HTTP 1.0, assume close after body
```

```
< HTTP/1.0 201 CREATED
< Content-Type: application/json
< Content-Length: 27
< Server: Werkzeug/0.8.3 Python/2.7.3
< Date: Mon, 01 Oct 2012 22:13:00 GMT
<
* Closing connection #0
{"task": "something different"}
```

Удаление задачи:

```
$ curl http://localhost:5000/todos/todo2 -X DELETE -v

> DELETE /todos/todo2 HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0)
libcurl/7.19.7 OpenSSL/0.9.8l zlib/1.2.3
> Host: localhost:5000
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 204 NO CONTENT
< Content-Type: application/json
< Content-Length: 0
< Server: Werkzeug/0.8.3 Python/2.7.2
< Date: Mon, 01 Oct 2012 22:10:32 GMT
```

А.А. Крук (ГГУ имени Ф. Скорины, Гомель)
Науч. рук. **В.Н. Кулинченко,** ст. преподаватель

ОПИСАНИЕ НАСТРОЙКИ ПОДСИСТЕМ СЕТИ ДЛЯ МОНИТОРИНГА И СТРИМИНГА ТЕЛЕКАНАЛОВ

Для мониторинга состояния получения и стриминга телеканалов по HLS-ссылкам, администратор сетей с помощью http-запроса подключается к web-интерфейсу мультипротокольного видеостримингового сервера Astra. Astra принимает телеканалы, на этом этапе идёт мониторинг стабильности и качества вещания, далее сервер перенаправляет трафик на мультикаст группу, где происходит маршутизация мультикаст-трафика. Трафик вещается сразу абонентам в виде цифрового телевидения по коаксиальному кабелю, а также идёт на middleware Ministra TV, к которой в дальнейшем подключаются абоненты с помощью клиентского оборудования: компьютеры (соответствующие системным требованиям), специализированные ТВ приставки, медиа-плееры, телевизоры с технологией Smart TV, мобильные устройства.