Auth Fortress

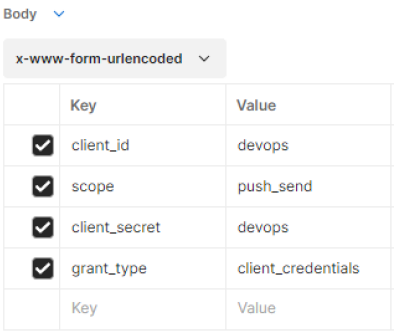
Your task is to build simple oauth2-based authorization system, consisting of two main endpoints for token management.

Part #1. 9:05-10:30

API description

Endpoint	Request Type	Description
/token	POST	Issues a new token for a client that have 2 hours lifetime and will be associated with specific scopes provided by the client
/check	GET	Verifies if the provided token is still valid and returns the associated scopes. If the token has expired or is invalid, an error message is returned.

/token request example (form-encoded) using postman



Grant type is static, always `client_credentials`

/token response example (successful response)

```
{
    "access_token": "e3b0c44298fc1c149afbf4c8...",
    "expires_in": 7200,
    "refresh_token": "",        # static, always empty string
    "scope": "push_send",
    "security_level": "normal", # static, always normal
    "token_type": "Bearer"      # static, always Bearer
}
```

/check request example, just pass Header

```
"Authorization": "Bearer e3b0c44298fc1c149afbf4c8..."
```

/check response example

```
{
    "ClientID": "devops",
    "Scope": "push_send"
}
```

Your API must run inside docker container and should be configured to run as a service managed by systemd on the server.

Set up a GitLab CI/CD pipeline to deploy the application to a remote server using SSH.

Use PostgreSQL to store clients, tokens and other relevant data. All database migrations (creating tables, adding constraints) should be managed with Flyway.

The initial version of your migrations (V1.0.0) should include basic tables such as clients and tokens. You are free to optimize tables as needed, ex adding/removing columns or tables (but these changes mustn't be included in V1.0.0).

Initial tables

```
create table if not exists public.token(
     client_id varchar(50)
    ,access_scope varchar[]
    ,access_token varchar default SUBSTR(UPPER(md5(random()::text)), 2, 22)
    ,expiration_time timestamp default current_timestamp + interval '2hours'
);


create table if not exists public.user (
     client_id varchar(50) unique
    ,client_secret varchar(100)
    ,scope varchar[]
);
```

Each team must provide their opponents with a short documentation that includes the necessary information on how to use their API.

This documentation should specify the available endpoints, required parameters, scopes and test users (with client_id and client_secret) that the opposing team can use for testing.

Part #2. 10:40-11:15

Test robustness of the other team's API while protecting your own.

Prepare report with 3 parts

1. Attack demonstration – show how your team attempted to break the other team's API, detailing the strategies used and the outcomes
2. Defense explanation – describe how you defended your own API against the other team's attacks, including any optimizations or fixes made during the process
3. Opponent's attack summary - explain how the other team successfully attacked your API, detailing the vulnerabilities they exploited and any weaknesses identified in your system. Discuss the steps you took (or plan to take) to address those issues

Part #3. 11:25-11:45

Each team should select one representative to present the results.

Notes:

1. You will have small number of users (<1000). Each user represents a system that use your authorizations to send notifications (push/sms) to their own clients.
2. Your users may not use the system efficiently and will always request a new token before each /check call, even if the previous token is still valid
3. You should handle at least 1000 simultaneous connections
4. First goal is to achieve 10 000 RPS, second is 50 000 RPS (requests per second)
5. Minimize the amount of data stored in the `token` table

6. Avoid storing secrets in your code
7. You can use `locust` for stress testing
8. You are free to use any resources you like, including ChatGPT