WASA Project 2022/23

Emanuele Panizzi, Enrico Bassetti

2022-10-31

Contents

WASA Project 2022/23: “WASAPhoto”

version 1

Introduction

As part of the Web and Software Architecture exam, you will:

1. define APIs using the OpenAPI standard
2. design and develop the server side (“backend”) in Go
3. design and develop the client side (“frontend”) in JavaScript
4. create a Docker container image for deployment

WASAPhoto

Keep in touch with your friends by sharing photos of special moments, thanks to WASAPhoto! Directly from your PC, you can upload your photos, and they will be visible to everyone who is following you.

Functional design specifications

Each user will be presented with a stream of photos in reverse chronological order, with information about when it was uploaded (date and time) and how many likes and comments it has. Users can place (and later remove) a “like” to photos from other users. Also, users can add comments to any photo (even those uploaded by themself). Comments can be removed by the author only.

Users can ban other users. If Alice (user) bans Eve (user), Eve won’t be able to see any information about Alice. Alice can decide to remove the ban at any moment.

Users will have their own personal profiles. The personal profile page for user detail: user’s photos (in reverse chronological order), their followers and following, and how many photos have been uploaded. Each user can change his/her own username, upload photos, remove photos, and follow/unfollow other users. Removal of a photo will also remove likes and comments.

User profiles can be searched via username.

The login is performed by specifying the username. See the “Simplified login” section for details.

Simplified login

In real-world scenarios, new developments avoid implementing registration, login, and password-lost flows as they are a security nightmare, cumbersome, error-prone, and they're outside the scope of projects. So, why lose money and time on implementing those? The best practice is now to delegate those tasks to a separate service ("identity provider"), either in-house (owned by the same company) or a third party (like "Login with Apple/Facebook/Google" buttons).

In this project, we do not have an external service like this. Instead, we decided to provide you with a specification for a login API so that you won't spend time dealing with the design of the endpoint. The provided OpenAPI document is at the end of this PDF.

The login endpoint accepts a username – like "Maria" – without any password. If the username already exists, the user is logged in. If the username is new, the user is registered and logged in. The API will return the user identifier that you need to pass into the `Authorization` header in any other API.

This authentication method is named "Bearer Authentication" (however, in this project, you should use the user identifier in place of the token):

- https://swagger.io/docs/specification/authentication/bearer-authentication/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

There is no need either for HTTP sessions or session cookies.

> What about "security"? What if a user logs in using the name of another user?
>
> In real-world projects, the identity provider is in charge of authenticating the user. We don't want you to focus on integrating any identity provider (as it's straightforward and not very interesting).

Further details

OpenAPI

You will need to define different APIs from the requirements above. For each API, you **must** define the `operationId` key. We expect to find at least these operation IDs:

- `doLogin` (see simplified login)
- `setMyUserName`
- `uploadPhoto`
- `followUser`
- `unfollowUser`

- `banUser`
- `unbanUser`
- `getUserProfile`
- `getMyStream`
- `likePhoto`
- `unlikePhoto`
- `commentPhoto`
- `uncommentPhoto`
- `deletePhoto`

Addendum

OpenAPI for simplified login

```yaml
openapi: 3.0.3
info:
  title: Simplified login API specification
  description: |-
    This OpenAPI document describes the simplified login API.
    Copy and paste the API from the `paths` key to your OpenAPI document.
  version: "1"
paths:
  /session:
    post:
      tags: ["login"]
      summary: Logs in the user
      description: |-
        If the user does not exist, it will be created,
        and an identifier is returned.
        If the user exists, the user identifier is returned.
      operationId: doLogin
      requestBody:
        description: User details
        content:
          application/json:
            schema:
              type: object
              properties:
                name:
                  type: string
                  example: Maria
                  pattern: '^.*?$'
                  minLength: 3
                  maxLength: 16
        required: true
      responses:
        '201':
          description: User log-in action successful
          content:
            application/json:
              schema:
```

```
                type: object
                properties:
                  identifier:
                    # change here if you decide to use an integer
                    # or any other type of identifier
                    type: string
                    example: "abcdef012345"
```