WASAPhoto

Emanuele Panizzi, Enrico Bassetti

Contents

WASA Project: “WASAPhoto”

version 5

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! You can upload your photos directly from your PC, and they will be visible to everyone following you.

Functional design specifications

Each user will be presented with a stream of photos (images) in reverse chronological order, with information about when each photo was uploaded (date and time) and how many likes and comments it has. The stream is composed by photos from “following” (other users that the user follows). Users can place (and later remove) a “like” to photos from other users. Also, users can add comments to any image (even those uploaded by themself). Only authors can remove their comments.

Users can ban other users. If user Alice bans user Eve, 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 profiles. The personal profile page for the user shows: the user’s photos (in reverse chronological order), how many photos have been uploaded, and the user’s followers and following. Users can change their usernames, upload photos, remove photos, and follow/unfollow other users. Removal of an image will also remove likes and comments.

A user can search other user profiles via username.

A user can log in just 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 outside the project scope. 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 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. On the other hand, in this project, you need not integrate an identity provider (as it is 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`

CORS

The backend **must** reply to CORS pre-flight requests with the appropriate setting.

To avoid problems during the homework grading, you should allow all origins and you should set the “Max-Age” attribute to 1 second. See the example code in the Fantastic Coffee decaffeinated repository.

Addendum

OpenAPI for simplified login

```
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"
```

Change log

Version 1

Initial version

Version 2

Fix some typos and grammar.

Version 3

Clarify that the stream is composed by photos from following.

Version 4

Add note about CORS settings.

Version 5

Add year 2023/24. Modify consequently so that it fits both years.