WASAText

Emanuele Panizzi, Enrico Bassetti

Contents

WASA Project: “WASAText”

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

WASAText

Connect with your friends effortlessly using WASAText! Send and receive messages, whether one-on-one or in groups, all from the convenience of your PC. Enjoy seamless conversations with text or GIFs and easily stay in touch through your private chats or group discussions.

Functional design specifications

The user is presented with a list of conversations with other users or with groups, sorted in reverse chronological order. Each element in the list must display the username of the other person or the group name, the user profile photo or the group photo, the date and time of the latest message, the preview (snippet) of the text message, or an icon for a photo message. The user can start a new conversation with any other user of WASAText, and this conversation will automatically be added to the list. The user can search for other users via the username and see all the existing WASAText usernames.

The user can create a new group with any number of other WASAText users to start a conversation. Group members can add other users to the group, but users cannot join groups on their own or even see groups they aren’t a part of. Additionally, users have the option to leave a group at any time.

The user can open a conversation to view all exchanged messages, displayed in reverse chronological order. Each message includes the timestamp, the content (whether text or photo), and the sender’s username for received messages, or one/two checkmarks to indicate the status of sent messages. Any reactions (comments) on messages are also displayed, along with the names of the users who posted them.

One checkmark indicates that the message has been received by the recipient (by all the recipients for groups) in their conversation list. Two checkmarks mean that the message has been *read* by the recipient (by all the recipients for groups) within the conversation itself.

The user can send a new message, reply to an existing one, forward a message, and delete any sent messages. Users can also react to messages (a.k.a. comment them) with an emoticon, and delete their reactions at any time (a.k.a. uncomment).

A user can log in simply by entering their username. For more information, refer to the "Simplified Login" section. Users also have the ability to update their name, provided the new name is not already in use by someone else.

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`
- `getMyConversations`
- `getConversation`
- `sendMessage`
- `forwardMessage`
- `commentMessage`
- `uncommentMessage`
- `deleteMessage`
- `addToGroup`
- `leaveGroup`
- `setGroupName`
- `setMyPhoto`
- `setGroupPhoto`

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

```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"
```

Change log

Version 1

Initial version