CSC309 Project P2 Documentation

Azalea Gui **LinKai** **Will Jarvis-Cross**

1 Project Structure

1.1 Framework Detail

This project has a backend written in python with the Django REST framework. The dependencies are managed with poetry. The frontend is written in TypeScript with React and Sass, built by Vite and served as static files separately from the backend. Testing of the backend is done with the frontend SDKs using the jest framework.

1.2 Installation

To install dependencies on Ubuntu 20.04, please run `startup.sh`.

On other operating systems, please first install node 20, npm, python 3.12, and pip. Then, run the following command:

```
poetry install
npm install -g yarn
yarn install
```

1.3 Running the project

```
# Run the developmental backend
./run.sh

# Run the frontend
yarn dev

# Run tests after the backend is running
yarn jest
```

1.4 File Structure

The backend source code and python module root is in the `backend` directory. Directly inside it are the django configuration files. Our API implementations and models are located in `backend/meetsphere`.

The frontend sources are located in the `src` directory. The main entry point is `src/main.tsx`. The UI components and pages are located in `src/components` and `src/pages` respectively. The API client is located in `src/lib`. The types used in the API client are located in `src/lib/types.ts`. (As of P2, the frontend clinet SDK is fully connected with the backend, but the UI components has not been updated. Things are not expected to work on the UI at this moment.)

The tests are written in TypeScript and located in `src/lib/tests`. These tests are very comprehensive, covering most usage scenarios and edge cases. The complete test suite can be run with the command `jest` or `yarn jest`.

2 API Endpoints

All API endpoints except for profile picture uploading accept parameters through the JSON body. The documentation below references many object types that are defined in `src/lib/types.ts`.

Please review `src/lib/tests/complete.test.ts` for a typical usage scenario of the API client.

2.1 User Features

2.1.1 Register

This endpoint is used to register a new user.

Endpoint: POST /register
Payload: string username, string password, string email
Return: None

2.1.2 Login

This endpoint is used to log in a user. It will return a JWT token that should be stored in the local storage and used for all future authenticated requests.

Endpoint: POST /login
Payload: string username, string password
Return: string JWT token

2.1.3 Logout

This endpoint is used to log out the current user. It will also remove the user's JWT token from the local storage.

Endpoint: POST /logout (Authenticated)
Payload: None
Return: None

2.1.4 Get current logged-in user

This endpoint is used to get the currently logged-in user's information.

Endpoint: GET /user (Authenticated)
Payload: None
Return: UserSelf object

2.1.5 Update user info

This endpoint is used to update the currently logged-in user's information.

Endpoint: POST /user (Authenticated)
Payload: Partial UserSelf object
Return: Updated UserSelf object

2.1.6 Upload profile picture

This endpoint is used to upload a new profile picture for the currently logged-in user.

Endpoint: POST /user/pfp (Authenticated)
Payload: Profile picture file (multipart form data)
Return: None

2.2 Contact Features

2.2.1 List contacts

This endpoint is used to get the user's contacts.

Endpoint: GET /contacts (Authenticated)
Payload: None
Return: Array of Contact objects

2.2.2 Add a new contact

This endpoint adds a new contact to the current user's contacts.

Endpoint: POST /contacts (Authenticated)
Payload: NewContact object
Return: None

2.2.3 Delete a contact

This endpoint deletes a contact from the current user's contacts.

Endpoint: DELETE /contacts (Authenticated)
Payload: { id: number }
Return: None

2.2.4 Update a contact

This endpoint is used to update a contact in the current user's contacts.

Endpoint: PATCH /contacts (Authenticated)
Payload: Partial Contact object
Return: None

2.3 Calendar Features

2.3.1 List calendars

This endpoint is used to get the current user's calendars.

Endpoint: GET /calendar (Authenticated)
Payload: None
Return: Array of Calendar objects

2.3.2 Create a new calendar

This endpoint adds a new calendar to the current user's calendar.

Endpoint: POST /calendar (Authenticated)
Payload: NewCalendar object
Return: None

2.3.3 Delete a calendar

This endpoint deletes a calendar from the current user's calendar.

Endpoint: DELETE /calendar (Authenticated)
Payload: { id: number }
Return: None

2.3.4 Update a calendar

This endpoint is used to update a calendar in the current user's calendar.

Endpoint: PATCH /calendar (Authenticated)
Payload: Partial Calendar object
Return: None

2.4 Meeting Features

2.4.1 List meetings

This endpoint is used to get the current user's meetings.

Endpoint: GET /meetings (Authenticated)
Payload: None
Return: Array of Meeting objects

2.4.2 Create a new meeting

This endpoint adds a new meeting to the current user's meetings.

Endpoint: POST /meetings (Authenticated)
Payload: NewMeeting object
Return: None

2.4.3 Delete a meeting

This endpoint deletes a meeting from the current user's meetings.

Endpoint: DELETE /meetings (Authenticated)
Payload: { id: string }
Return: None

2.4.4 Update a meeting

This endpoint is used to update a meeting in the current user's meetings.

Endpoint: PATCH /meetings (Authenticated)
Payload: Partial Meeting object
Return: None

2.4.5 Send out an invitation for a meeting

This endpoint is used to send out an invitation email to ask the invitee to select a time for the meeting.

Endpoint: POST /meetings/{id}/invite (Authenticated)
Payload: None
Return: None

2.4.6 Get a meeting (from an invitation link)

This endpoint is used to get the meeting information from a UUID sent in an invitation email.

Endpoint: GET /meetings/{id}
Payload: None
Return: Meeting object

2.4.7 Accepting an invitation

This endpoint is used to accept an invitation to a meeting.

Endpoint: POST /meetings/{id}/accept (Authenticated)
Payload: { time: string }
Return: None