**DreamJournal Documentation**

This documentation provides an overview of the application's functionality, describing the user actions on the frontend and the corresponding backend processes.

**1. Register Page**

**Frontend**

* **User Action**:
  + The user navigates to the registration page.
  + The user fills out the registration form with details such as username, email, password, and optionally uploads a profile picture.
  + The user clicks the "Register" button.
* **Frontend Process**:
  + The form input is validated on the client side (e.g., checking for empty fields, valid email format, and password strength).
  + If validation passes, the frontend sends a POST request to the backend with the user's registration details.

**Backend**

* **Backend Process**:
  + The backend receives the POST request at the /register endpoint.
  + The backend validates the input (e.g., checking if the username or email already exists in the database).
  + If validation passes:
    - The user's details are saved in the database.
    - A success response is sent back to the frontend.
  + If validation fails (e.g., duplicate email or invalid input), an error response is sent back to the frontend.

**2. Login Page**

**Frontend**

* **User Action**:
  + The user navigates to the login page.
  + The user enters their username and password.
  + The user clicks the "Login" button.
* **Frontend Process**:
  + The form input is validated on the client side (e.g., checking for empty fields).
  + If validation passes, the frontend sends a POST request to the backend with the user's login credentials.

**Backend**

* **Backend Process**:
  + The backend receives the POST request at the /login endpoint.
  + The backend validates the credentials:
    - It checks if the username and password combination exists in the database.
  + If validation passes:
    - A user DTO is sent back to the frontend.
    - The user's data (e.g., username, profile picture) is included in the response.
  + If validation fails, an error response is sent back to the frontend.

**3. Creating a Post**

**Frontend**

* **User Action**:
  + The user navigates to the /app url.
  + The user fills out the post form at the top with a title, content, and optionally uploads images and/or selects tags.
  + The user clicks the "Post" button.
* **Frontend Process**:
  + The form input is validated on the client side (e.g., checking for empty title or content, limiting the number of tags).
  + If validation passes, the frontend sends a POST request to the backend with the post details.

**Backend**

* **Backend Process**:
  + The backend receives the POST request at the /dream/create-dream endpoint.
  + The backend saves the dream in the database

**4. Viewing Posts**

**Frontend**

* **User Action**:
  + The user navigates to the feed or profile page.
  + The user scrolls through the list of posts.
  + The user clicks on a post to view its details.
* **Frontend Process**:
  + The frontend fetches posts from the backend using a GET request.
  + The posts are displayed in a list or grid format.
  + When the user clicks on a post, a modal or new page is opened to display the post details.

**Backend**

* **Backend Process**:
  + (Feed posts): The backend receives the GET request at the /dream/list-dream-all/{id}, the id resembling the user’s id, to exclude dreams from blocked users.
  + (Individual post): The backend recieves the GET request at the /dream/get-post-by-id/{id}, id resembling the dream’s id. This ensures the latest state a post is in upon viewing in the modal.
  + The backend retrieves the posts from the database, optionally filtering them based on user preferences (e.g., tags, followed users).
  + The posts are sent back to the frontend in the response.

**5. Deleting a Post**

**Frontend**

* **User Action:**
  + The user clicks on the hamburger menu in the top-right corner of the post made by them.
  + The user clicks on the delete option
* **Frontend Process:**
  + The Frontend sends a DELETE request to the /dream/delete-dream/{id}, id resembling the id of the post to be deleted.

**Backend**

* **Backend processes:**
  + The backend deletes the post matching the data received in the request.

**6. Liking a Post/Removing Like from a Post**

**Frontend**

* **User Action**:
  + The user clicks the "Like/Remove like" button on a post.
* **Frontend Process**:
  + Depending on wether the post is liked, the Frontend sends a request to the backend to add a like, or remove a like.
* **Backend Process**:
  + The backend receives the POST request at the /dream/like-post/{id}, id resembling the dream’s id / DELETE request at the /dream/remove-like-post/{id}, id resembling the post’s id
  + (Liking): The backend adds a new row in the reactions table in the database, containing the post’s id, the user’s id, and the time of action.
  + (Removing Like): The backend delete’s the row where the dream\_id and the user\_id is matching the data in the request.

**7. Commenting on a Post**

**Frontend**

* **User Action**:
  + The user clicks the "Comment" button on a post.
  + The user enters a comment in the input field.
  + The user clicks the "Submit" button.
* **Frontend Process**:
  + The frontend sends a POST request to the backend with the post ID, user ID, and comment content.

**Backend**

* **Backend Process**:
  + The backend receives the POST request at the /dream/comment-on-post endpoint.
  + The backend saves the comment in the database.

**8. Filtering Posts by Tags**

**Frontend**

* **User Action**:
  + The user selects tags from the sidebar or tag selector. The user can choose to filter posts for tags posted within the last 7 days (Trending), or regardless of date of posting (All-time).
* **Frontend Process**:
  + The selected tags are sent to the backend as query parameters or in the request body.
  + The frontend updates the displayed posts based on the backend's response.
  + By default, selecting tags using the Tag selector, it filters for posts regardless of date of posting.

**Backend**

* **Backend Process**:
  + (Trending): The backend receives the request at the /dream/get-trending-post-by-tag/{tag} endpoint, tag resembling the tag(s) selected.
  + The backend retrieves posts from the database that match the selected tags and are posted within the last seven days.
  + The filtered posts are sent back to the frontend in the response.
  + (All-time): The backend recieves the request at the /dream/get-all-post-by-tag/{tag} endpoint, tag resembling the tag(s) selected
  + The backend retrieves posts from the database that match the selected tags.
  + The filtered posts are sent back to the frontend in the response.

**9. Blocking / Unblocking user**

**Frontend**

* **User action:**
  + (Blocking): The user clicks on the hamburger menu in the top-right corner of the post, and clicks on the block option.
  + (Unblocking): The user clicks on the hamburger menu in the top-right corner of the post, and clicks on the unblock option. // The user clicks on the Blocks menu point of the left side bar, and chooses which users they would like to unblock.
* **Frontend process:**
  + (Blocking): The Frontend sends a POST request to the /user/block-user endpoint along with the user’s id and the id of the user to be blocked.
  + (Unblocking): The Frontend sends a POST request to the /user/unblock-user
  + endpoint, along with the user’s id and the id of the user to be unblocked.

**Backend**

* **Backend process:**
  + (Blocking): The Backend creates a new row in the blocks table with the blocker\_id and the blocked\_id matching the data received in the request.
  + (Unblocking): The Backend deletes the row containing the blocker\_id and the blocked\_id matching the data received in the request.

**10. Following/Unfollowing a User**

**Frontend**

* **User actions:**
  + The user clicks on the hamburger menu in the top-right corner of the post made by any other user.
  + (Following): The user clicks on the follow option
  + (Unfollowing): The user clicks on the follow option
* **Frontend process**
  + (Following): The Frontend sends a POST request to the backend containing the user’s id and the id of the user being followed.
  + (Unfollowing): The Frontend sends a POST request to the backend containing the user’s Id and the id of the user being unfollowed

**Backend**

* **Backend process**
  + (Following): The Backend saves a new row in the database for the follower\_id and the following\_id matching the ids of the data received.
  + (Unfollowing): The Backend deletes the row containing the follower\_id and the followed\_id matching the ids of the data received.