**Documentation**

**MessagesApp:**

**MessageController Utility Routes for populating the database with dummy data**

1. populateRandomMessages
   * Route: POST /messages/populateRandomMessages
   * Inputs: None
   * Outputs: None
2. populateRandomChats
   * Route: POST /messages/populateRandomChats
   * Inputs: None
   * Outputs: None
3. createDummyData
   * Route: POST /messages/createDummyData
   * Inputs: None
   * Outputs: None

Authentication Required: All routes listed below require an Authorization header with a valid authentication token unless explicitly stated otherwise.

**MessageService and ChatService Routes**

1. getAllMessages
   * Route: GET /messages/getAllMessages
   * Inputs: None
   * Outputs: List of Message objects
2. sendMessage
   * Route: POST /messages/send/{chatId}/{senderId}/{type}
   * Inputs:
     + chatId (UUID)
     + senderId (UUID)
     + content (String) from request body
     + type (String)
   * Outputs: Created Message object
3. getChatMessages
   * Route: GET /messages/chat/openChat/{chatId}
   * Inputs:
     + chatId (UUID)
   * Outputs: List of Message objects for the specified chat
4. markMessagesAsSeen
   * Route: PUT /messages/seeChat/{chatId}/{userId}
   * Inputs:
     + chatId (UUID)
     + userId (UUID)
   * Outputs: List of Message objects after marking as seen
5. updateMessageStatus
   * Route: PUT /messages/changeStatus/{messageId}/{status}
   * Inputs:
     + messageId (UUID)
     + status (MessageStatus enum)
   * Outputs: Updated Message object with new status
6. deleteMessage
   * Route: DELETE /messages/deleteMessage/{messageId}
   * Inputs:
     + messageId (UUID)
   * Outputs: Optional Message object that was deleted
7. editTextMessage
   * Route: PUT /messages/editMessage/{messageId}
   * Inputs:
     + messageId (UUID)
     + newContent (String) from request body
   * Outputs: Updated TextMessage object with edited content
8. createChat
   * Route: POST /messages/chat/createChat
   * Inputs:
     + name (String) from request body
     + participantIds (List) from request body
   * Outputs: Created Chat object
9. addParticipant
   * Route: PUT /messages/chat/addUser/{chatId}/{userId}
   * Inputs:
     + chatId (UUID)
     + userId (UUID)
   * Outputs: Updated Chat object with added participant
10. removeParticipant
    * Route: PUT /messages/chat/removeParticipant/{chatId}/{userId}
    * Inputs:
      + chatId (UUID)
      + userId (UUID)
    * Outputs: Updated Chat object with removed participant
11. getUserChats
    * Route: GET /messages/chat/user/{userId}
    * Inputs:
      + userId (UUID)
    * Outputs: List of Chat objects where the user is a participant
12. pinChat
    * Route: PUT /messages/chat/pin/{chatId}
    * Inputs:
      + chatId (UUID)
    * Outputs: Updated Chat object marked as pinned
13. unpinChat
    * Route: PUT /messages/chat/unpin/{chatId}
    * Inputs:
      + chatId (UUID)
    * Outputs: Updated Chat object marked as unpinned
14. deleteChat
    * Route: DELETE /messages/deleteChat/{chatId}
    * Inputs:
      + chatId (UUID)
    * Outputs: Optional Chat object that was deleted
15. updateChat
    * Route: PUT /messages/chat/updateName/{chatId}
    * Inputs:
      + chatId (UUID)
      + newName (String) from request body
    * Outputs: Updated Chat object with new name

**SearchApp:**

### **Posts Related Endpoints**

### **searchPosts**

**Route:** POST /search/post/search  
**Inputs:**

* searchQuery (String) — as a query parameter  
  **Outputs:**
* List of PostDTO objects matching the search query

### **filterPosts**

**Route:** POST /search/post/filter  
**Inputs:**

* posts (List<PostDTO>) — from request body
* likes (Boolean)
* minLikes (Integer)
* maxLikes (Integer)
* shares (Boolean)
* minShares (Integer)
* maxShares (Integer)
* date (Boolean)
* startDate (LocalDate)
* endDate (LocalDate)  
  **Outputs:**
* Filtered list of PostDTO objects

### 

### 

### **undoFilterPosts**

**Route:** POST /search/post/undoFilters  
**Inputs:**

* likes (Boolean)
* shares (Boolean)
* date (Boolean)  
  **Outputs:**
* Unfiltered (original) list of PostDTO objects

### **sortPosts**

**Route:** POST /search/post/sort  
**Inputs:**

* posts (List<PostDTO>) — passed as a query parameter (⚠️ this may need to be in the request body for larger payloads)
* sortCriteria (String) — e.g., "likes", "date"
* sortOrder (String) — "asc" or "desc"  
  **Outputs:**
* Sorted list of PostDTO objects

### **User Related Endpoints**

### **searchUsers**

**Route:** POST /search/user/search  
**Inputs:**

* searchQuery (String) — as a query parameter  
  **Outputs:**
* List of UserDTO objects matching the search query

### 

### **filterUsers**

**Route:** POST /search/user/filter  
**Inputs:**

* users (List<UserDTO>) — from request body
* age (Boolean)
* minAge (Integer)
* maxAge (Integer)
* gender (Boolean)
* genderToMatch (String) — e.g., "Male" or "Female"  
  **Outputs:**
* Filtered list of UserDTO objects

### **undoFilterUsers**

**Route:** POST /search/user/undoFilters  
**Inputs:**

* age (Boolean)
* gender (Boolean)  
  **Outputs:**
* Unfiltered (original) list of UserDTO objects

### **sortUsers**

**Route:** POST /search/user/sort  
**Inputs:**

* users (List<UserDTO>) — passed as a query parameter (again, consider using request body if size is large)
* sortCriteria (String) — e.g., "age", "gender"
* sortOrder (String) — "asc" or "desc"  
  **Outputs:**
* Sorted list of UserDTO objects

**UserApp:**

### **1. Populate Random Users**

**Route:** POST /user/populateRandom  
 **Input:** None  
 **Output:** None (void)  
 **Description:** Populates the database with random users by calling userService.populateRandomUsers() and databasePopulatorService.populateUsers(). Mainly for testing/demo purposes.

### **2. Create Dummy Users**

**Route:** POST /user/dummyData  
 **Input:** None  
 **Output:** None (void)  
 **Description:** Creates dummy user data for testing, via databasePopulatorService.createDummyUsers().

### **3. Delete All Users**

**Route:** DELETE /user/wipe  
 **Input:** None  
 **Output:**

* 200 OK with message: "All users have been deleted successfully."  
   **Description:** Deletes all users from the database. Use with caution.

### **4. Create User**

**Route:** POST /user/  
 **Headers:**

* Authorization (optional, but required if authenticationEnabled is true)  
   **Input:**
* JSON User object (fields as per User model)  
   **Output:**
* 200 OK with created user object on success
* 401 Unauthorized if no or invalid token  
   **Description:** Creates and saves a new user in the database.

### **5. Get All Users**

**Route:** GET /user/  
 **Headers:**

* Authorization (optional, but required if authenticationEnabled is true)  
   **Input:** None  
   **Output:**
* 200 OK with list of all users
* 401 Unauthorized if no or invalid token  
   **Description:** Fetches all users from the database.

### 

### 

### **6. Get User by ID**

**Route:** GET /user/{userId}  
 **Headers:**

* Authorization (optional, but required if authenticationEnabled is true)  
   **Input:**
* userId (UUID path variable)  
   **Output:**
* 200 OK with user object if found
* 401 Unauthorized if no or invalid token  
   **Description:** Fetches a specific user by their UUID.

### **7. Update User**

**Route:** PUT /user/{userId}  
 **Headers:**

* Authorization (optional, but required if authenticationEnabled is true)  
   **Input:**
* userId (UUID path variable)
* JSON User object with updated fields  
   **Output:**
* 200 OK with result message
* 401 Unauthorized if no or invalid token  
   **Description:** Updates the user information for the given userId.

### **8. Delete User**

**Route:** DELETE /user/{userId}  
 **Headers:**

* Authorization (optional, but required if authenticationEnabled is true)  
   **Input:**
* userId (UUID path variable)  
   **Output:**
* 200 OK with message "User deleted."
* 401 Unauthorized if no or invalid token  
   **Description:** Deletes a user by their UUID.

### **9. Check User Existence**

**Route:** GET /user/check/{userId}  
 **Input:**

* userId (UUID path variable)  
   **Output:**
* true if user exists, otherwise false  
   **Description:** Checks whether a user with the given UUID exists.

### **10. Login with Password**

**Route:** POST /user/login  
 **Input:**

* JSON with fields:  
  + email (string)
  + password (string)  
     **Output:**
* 200 OK with JSON { "token": "<auth-token>" } on success
* 401 Unauthorized with message "Invalid credentials" on failure  
   **Description:** Authenticates user with email and password using the PasswordLoginStrategy. Returns an authentication token if successful.

### **11. Login with OTP**

**Route:** POST /user/otpLogin  
 **Input:**

* JSON with fields:  
  + email (string)
  + otp (string)  
     **Output:**
* 200 OK with JSON { "token": "<auth-token>" } on success
* 401 Unauthorized with message "Invalid OTP" on failure  
   **Description:** Authenticates user with email and one-time-password (OTP) using the OtpLoginStrategy. Returns an authentication token if successful.

### 

### **12. Logout**

**Route:** POST /user/logout  
 **Headers:**

* Authorization with token  
   **Input:** None  
   **Output:**
* 200 OK with message "User logged out successfully." on success
* 401 Unauthorized with message "Missing or invalid token." on failure  
   **Description:** Invalidates the user's authentication token to log them out.

### **13. Block User**

**Route:** POST /user/{userId}/block/{targetId}  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable): User who is blocking
* targetId (UUID path variable): User to be blocked  
   **Output:**
* 200 OK with message "User blocked successfully."
* 401 Unauthorized if unauthorized  
   **Description:** Blocks the target user for the given user.

### **14. Unblock User**

**Route:** DELETE /user/{userId}/unblock/{targetId}  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable): User who is unblocking
* targetId (UUID path variable): User to be unblocked  
   **Output:**
* 200 OK with message "User unblocked successfully."
* 401 Unauthorized if unauthorized  
   **Description:** Unblocks the target user for the given user.

### **15. Get Blocked Users**

**Route:** GET /user/{userId}/blocked-users  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable)  
   **Output:**
* 200 OK with a list of UUIDs representing blocked users
* 401 Unauthorized if unauthorized  
   **Description:** Retrieves a list of users blocked by the given user.

### **16. Add Friend**

**Route:** POST /user/{userId}/friend/{friendId}  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable): User adding a friend
* friendId (UUID path variable): User to add as friend  
   **Output:**
* 200 OK with updated friend list or confirmation
* 401 Unauthorized if unauthorized  
   **Description:** Adds the specified user as a friend to the current user.

### **17. Remove Friend**

**Route:** DELETE /user/{userId}/unfriend/{friendId}  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable): User removing a friend
* friendId (UUID path variable): User to remove from friend list  
   **Output:**
* 200 OK with message "Friend removed."
* 401 Unauthorized if unauthorized  
   **Description:** Removes a friend from the user's friend list.

### **18. Get Friends**

**Route:** GET /user/{userId}/friends  
 **Headers:**

* Authorization with token  
   **Input:**
* userId (UUID path variable)  
   **Output:**
* 200 OK with list of friends (likely User objects or UUIDs)
* 401 Unauthorized if unauthorized  
   **Description:** Retrieves the friend list of the specified user.

### **19. Validate Token**

**Route:** GET /user/validate-token/{token}  
 **Input:**

* token (path variable, string)  
   **Output:**

200 OK with JSON:  
  
 json  
CopyEdit  
{

"valid": true/false,

"message": "optional message if invalid"

}

* 400 Bad Request if token is missing or blank  
   **Description:** Checks whether a given authentication token is valid.

### **20. Get User Token**

**Route:** GET /user/{userId}/token  
 **Input:**

* userId (UUID path variable)  
   **Output:**
* 200 OK with JSON { "token": "<token>" } if token exists
* 404 Not Found with message "Token not found for the specified user." otherwise  
   **Description:** Retrieves the current authentication token of a user (if any).

## 

## **WallApp: PostController Routes**

**Authentication Required:** All routes listed below require an Authorization header with a valid authentication token unless explicitly stated otherwise.

### **Create Post**

**Route:** POST /wallApp/posts  
**Inputs:**

* PostRequest (JSON body)  
  **Outputs:**
* Created Post object

### **Populate Dummy Posts**

**Route:** POST /wallApp/posts/dummy  
**Inputs:** None  
**Outputs:**

* Success message (e.g. "Dummy posts created")

### **Get All Posts**

**Route:** GET /wallApp/posts/all  
**Inputs:** None  
**Outputs:**

* List of all Post objects

### **Get My Posts**

**Route:** GET /wallApp/posts/{userId}/myposts  
**Inputs:**

* userId (UUID path variable)  
  **Outputs:**
* List of posts by the specified user

### **Get Post by ID**

**Route:** GET /wallApp/posts/{id}  
**Inputs:**

* id (UUID path variable)  
  **Outputs:**
* Post object if found

### **Update Post**

**Route:** PUT /wallApp/posts/{id}  
**Inputs:**

* id (UUID path variable)
* PostRequest (JSON body with updated content)  
  **Outputs:**
* Updated Post object

### **Delete Post**

**Route:** DELETE /wallApp/posts/{id}  
**Inputs:**

* id (UUID path variable)  
  **Outputs:**
* Success message or void

### 

### 

### 

### **Delete All Posts**

**Route:** DELETE /wallApp/posts/deleteAll  
**Inputs:** None  
**Outputs:**

* Success message or void

### **Like Post**

**Route:** PUT /wallApp/posts/{userId}/like/{id}  
**Inputs:**

* userId (UUID path variable)
* id (UUID of the post)  
  **Outputs:**
* Updated Post object with like added

### **Share Post**

**Route:** POST /wallApp/posts/{userId}/sharepost/{id}  
**Inputs:**

* userId (UUID path variable)
* id (UUID of the post to share)  
  **Outputs:**
* New Post object representing the shared post

### **Add Friend**

**Route:** POST /wallApp/posts/{userId}/friend/{friendId}  
**Inputs:**

* userId (UUID path variable)
* friendId (UUID path variable)  
  **Outputs:**
* Success message or updated friends list

### **Get Friends**

**Route:** GET /wallApp/posts/{userId}/friends  
**Inputs:**

* userId (UUID path variable)  
  **Outputs:**
* List of friend UUIDs

### **Remove Friend**

**Route:** DELETE /wallApp/posts/{userId}/unfriend/{friendId}  
**Inputs:**

* userId (UUID path variable)
* friendId (UUID path variable)  
  **Outputs:**
* Success message or updated friends list

## **WallApp: NotificationController Routes**

**Authentication Required: All routes listed below require an Authorization header with a valid authentication token unless explicitly stated otherwise.**

### **Get Notifications**

**Route: GET /wallApp/notifications/{userId}  
Inputs:**

* **userId (UUID path variable)  
  Outputs:**
* **List of Notification objects for the specified user**