**Routes**

1. **Register Route**
   * Endpoint: **POST /api/v1/users/register**
   * Description: Register a new user.
2. **Login Route**
   * Endpoint: **POST /api/v1/users/login**
   * Description: Authenticate a user and provide a session or token.
3. **Logout Route**
   * Endpoint: **POST /api/v1/users/logout**
   * Description: Clear user session or token.
4. **Create Profile**
   * Endpoint: **POST /api/v1/users/createProfile**
   * Description: A mandatory profile completion step post-registration to ensure users provide essential details before accessing core platform functionalities.

**For each of these routes, I'll need a corresponding controller function:**

1. **Register Controller**

* Input: User details (username, password, email, etc.)
* Action: Save the user to the database (hashed password, not plain text).
* Output: Confirmation message (and possibly a login token/session).

1. **Login Controller**

* Input: User credentials (username/email and password).
* Action: Authenticate the user, start a session or provide a token.
* Output: User details and a token or session ID.

1. **Logout Controller**

* Input: Session ID or token.
* Action: Invalidate the session or token.
* Output: Confirmation message.

1. **Create Profile Controller**

* Input: User profile details
* Action:
  + First, verify that the user is authenticated.
  + Check if the user already has a profile created. If one exists, return an error or prompt to update instead.
  + Save the profile details to the database.
* Output:
  + Successful creation message.
  + The created profile details.

**Let’s get started...!**

* **authorizationController.js** will deal primarily with authentication and related concerns (login, logout, JWTs, password resets, etc).
* **userController.js** will handle CRUD operations for users, profile updates, viewing user profiles, etc.

Here’s the structure:

**1. authorizationController.js**

* **Signup**
  + Check if user exists
  + Hash password
  + Create user
  + Return user
* **Login**
  + Check if user exists
  + Check if password matches
  + Generate a JWT
  + Return JWT and user
* **Logout**
  + For JWTs: Handled client-side by removing token. Optionally, maintain a server-side blacklist.
  + For sessions: Clear the user session.
* **Forgot Password**
  + Generate a unique token.
  + Send an email with a link containing that token.
* **Reset Password**
  + Check if token is valid.
  + Hash and update password.
* **Update Password**
  + Check current password.
  + Hash and update to new password.

**2. userController.js**

* **Get User Profile**
  + Return the user's data.
* **Update User Profile**
  + Update the user's details.
* **Delete User**
  + Delete the user from the database.
* **List of Online Users**
  + I can maintain online status in the user model.
* **Other User operations**
  + Any other operations specific to the user entity.

**Dependencies:**

1. **bcryptjs**: For hashing and verifying passwords.
2. **jsonwebtoken (JWT)**: For generating and verifying JSON web tokens.
3. **crypto**: Built-in Node.js package to generate random tokens, for use in features like "forgot password".
4. **nodemailer**: For sending emails (e.g., for password resets).
5. **validator**: For validating emails, passwords, and other input.

**User registration flow:**

1. **Account Registration** (username, email, password, and password confirmation)
2. **Profile Completion** (all the other details, e.g., first name, last name, languages, about, photos, location, etc.)

Here's a step-by-step breakdown of the implementation:

**1. Account Registration:**

1. **Frontend**:
   * Display a registration form with fields: username, email, password, and password confirmation.
   * On successful submission, the backend will respond, and the frontend should then redirect the user to the profile creation page.
2. **Backend**:
   * Accept only username, email, password, and password confirmation for this step.
   * Validate these fields based on the Mongoose schema.
   * If all validation passes, save this user in the database.
   * Generate a JWT token to keep the user authenticated.
   * Respond to the frontend, signaling a successful registration.

**2. Profile Completion:**

1. **Frontend**:
   * Display the profile completion form with fields like first name, last name, languages, about, etc.
   * Ensure the user can't navigate away from this page without completing the profile or they'll be directed back to it.
   * On successful submission, update the user's profile in the backend.
2. **Backend**:
   * Receive the data from the profile completion form.
   * Validate the data based on the Mongoose schema.
   * If all validation passes, update the previously created user entry in the database with the additional details.
   * Respond to the frontend.

**3. Post-Profile Completion:**

* Once the profile is successfully completed, the user can be redirected to the ‘discover’ page.
* The user should now have full access to the website's features.

**Note:** If a registered user exits the profile creation page without completing it and later logs in, they'll be redirected there again, which must be completed before accessing the rest of the website.