**Project Documentation - Snapsy**

**Snapsy** is a modern, minimalist **social media platform** built for creators, influencers, and casual users to share visual content like photos, moments, and memories. It offers a familiar Instagram-like experience with features like posts, likes, comments, followers, and notifications

**Key Features**

**User & Auth**

* Register/login with email or username
* JWT token-based authentication
* Edit profile (name, bio, avatar)
* Follow/unfollow other users

**Posts**

* Create posts with multiple images, hashtags, captions
* Like, comment, and save posts
* Delete own posts with confirmation
* Explore grid of posts with hover zoom
* View full post with details and comments

**Comments**

* Add and delete your own comments
* Comments show username, avatar, timestamp
* Your own comments float to the top

**Notifications**

* Receive alerts for new followers, likes, and comments
* Click to open related post or profile

**Explore & Search**

* Public explore feed
* Search for users

**Tech Stack**

**Frontend**

* **React (Vite)** – lightning-fast SPA development
* **Tailwind CSS** – utility-first styling
* **Formik + Yup** – robust forms and validation
* **React Icons** – consistent iconography
* **Axios** – API communication

**Backend**

* **Node.js + Express** – RESTful API backend
* **MongoDB + Mongoose** – flexible NoSQL database
* **Zod** – backend request validation
* **JWT** – authentication and route protection

**Other Integrations**

* **Appwrite** – image storage (for profile pictures and post images)

**MongoDB Schemas**

**User**

* email, username, password, name, bio, avatar
* isPrivate, followers, following
* savedPosts, likedPosts

**Post**

* author, username, images, caption, hashtags, likes

**Comment**

* post, user, username, text, timestamp

**Notification**

* recipient, type (like, comment, follow)
* from, post, read

**API Routes**

**1. Authentication**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** | **Errors** |
| --- | --- | --- | --- | --- | --- |
| POST | /auth/register | — | { email, username, password, name? } | { message: 'User created', user: { id, email, username, name } } | 400, 409, 500 |
| POST | /auth/login | — | { usernameOrEmail, password } | { token, user: { id, username, email } } | 400, 401, 500 |

**2. User & Profile**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| GET | /user/myprofile | Authorization: Bearer ¤ | — | { id, username, name, bio, avatar, followersCount, followingCount, totalPosts, ... } |
| PUT | /user/edit-profile | Authorization: Bearer ¤ | multipart/form-data: name, bio, avatar? (URL) | { message: 'Profile updated', user: { name, bio, avatar } } |
| GET | /public/get-user/:id | Authorization: Bearer ¤ | — | { username, name, avatar, bio, followersCount, followingCount, totalPosts, posts: [ { id, images, likeCount } ] } |

**3. Posts**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| POST | /user/create-post | Authorization: Bearer ¤ | { caption, images: [file...], hashtags: [string] } | { message: 'Post created', post: { id, author, ... } } |
| GET | /public/get-post/:postId | Authorization: Bearer ¤ | — | { \_id, author:{id,username,avatar}, images, caption, hashtags, createdAt, likeCount, isLiked, isSaved } |
| DELETE | /user/delete-post/:id | Authorization: Bearer ¤ | — | { message: 'Post deleted' } |
| POST | /public/like-unlike/:id | Authorization: Bearer ¤ | — | { likeCount, isLiked } |
| POST | /public/save-unsave/:id | Authorization: Bearer ¤ | — | { saved } |
| GET | /public/explore | Authorization: Bearer ¤ | — | [{ \_id, images:[string], likeCount }] |

**4. Comments**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| GET | /public/get-comments/:postId | Authorization: Bearer ¤ | — | [ { \_id, post, user:{id,username,avatar}, text, timestamp } ] |
| POST | /public/add-comment/:postId | Authorization: Bearer ¤ | { text } | { message: 'Comment added', comment: { ... } } |
| DELETE | /public/delete-comment/:id | Authorization: Bearer ¤ | — | { message: 'Comment deleted' } |

**5. Follow / Unfollow**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| POST | /public/follow/:id | Authorization: Bearer ¤ | — | { following: true } or { following: false } |

**6. Search & Explore**

| **Method** | **Endpoint** | **Headers** | **Query Params** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| GET | /public/search | Authorization: Bearer ¤ | ?username=string | [ { \_id, username, avatar } ] |

**7. Notifications**

| **Method** | **Endpoint** | **Headers** | **Body** | **Success Response (200)** |
| --- | --- | --- | --- | --- |
| GET | /user/notifications | Authorization: Bearer ¤ | — | [ { \_id, type, from:{id,username,avatar}, post:{id,images}, read, createdAt } ] |

**Validation Middleware**

We use **Zod** schemas to validate incoming payloads for routes such as registration, login, profile updates, post creation, and comments.

**Why?**

1. **Data Integrity**: Ensures requests contain required fields and correct types.
2. **Security**: Prevents malicious or malformed inputs before any database operation.
3. **Developer Experience**: Returns clear, structured error messages back to clients.

Example Zod schema for creating a comment:

const addCommentSchema = z.object({

text: z.string().min(1, 'Comment cannot be empty')

});

This runs as middleware before your controller logic to catch invalid data early.