horizontal line

|  | **AlphaTribe** |
| --- | --- |

Assignment: Stock Discussion Platform Backend

**5th September 2024**

# OVERVIEW

You are tasked with building the backend of a community platform where users can discuss various stocks in the market. The focus will be on creating key backend functionality such as user authentication, posts related to stocks, and the ability for users to engage with posts through comments and likes. The backend should be built using the MERN stack (MongoDB, Express.js, Node.js).

# REQUIREMENTS

Tech Stack - **MERN (MongoDB, Expres.js, Node.js)**

1. **User Authentication (JWT-based):**
   1. Users should be able to update their profile (username, bio, and profile picture).
2. **Stock Post Management:**
   1. Users can create posts related to specific stocks. A post should contain:
      1. Stock symbol (e.g., AAPL, TSLA)
      2. Title and description
      3. Tags (optional)
      4. Created date
   2. Posts should be linked to the user who created them.
3. **Commenting System:**
   1. Users can comment on any post.
   2. Comments should be linked to both the post and the user who made the comment.
4. **Like System:**
   1. Users can like or unlike posts.
   2. A post should display the total number of likes.
5. **Filtering and Sorting:**
   1. Implement an API endpoint to fetch posts filtered by stock symbol or tags.
   2. Posts should be sortable by creation date or number of likes.
6. **API Documentation:**
   1. Use Swagger or Postman to document the API endpoints (optional but preferred)
7. **Database Schema:**
   1. Use MongoDB for data storage, and design the schema to handle users, posts, comments, and likes.
8. **Bonus Features (Optional):**
   1. Implement basic pagination when fetching posts.
   2. Include user authentication in the comment and like endpoints, ensuring only authenticated users can interact with posts.
   3. Use Socket.io for real-time updates on new comments or likes.

# 

# API ENDPOINTS

#### **User Authentication and Management:**

1. **User Registration - POST** /api/auth/register
   * Request Body: { username, email, password }
   * Response: { success: true, message: 'User registered successfully', userId }
2. **User Login - POST** /api/auth/login
   * Request Body: { email, password }
   * Response: { token, user: { id, username, email } }
3. **Get User Profile - GET** /api/user/profile/:userId
   * Headers: { Authorization: Bearer <token> }
   * Response: { id, username, bio, profilePicture }
4. **Update User Profile - PUT** /api/user/profile
   * Headers: { Authorization: Bearer <token> }
   * Request Body: { username, bio, profilePicture }
   * Response: { success: true, message: 'Profile updated' }

#### 

#### **Stock Posts Management:**

1. **Create a Stock Post - POST** /api/posts
   * Headers: { Authorization: Bearer <token> }
   * Request Body: { stockSymbol, title, description, tags }
   * Response: { success: true, postId, message: 'Post created successfully' }
2. **Get All Stock Posts (with filters and sorting) - GET** /api/posts
   * Query Parameters:
     + stockSymbol (optional)
     + tags (optional)
     + sortBy (date or likes, optional)
   * Response: [ { postId, stockSymbol, title, description, likesCount, createdAt } ]
3. **Get a Single Stock Post (with comments) - GET** /api/posts/:postId
   * Response: { postId, stockSymbol, title, description, likesCount, comments: [ { commentId, userId, comment, createdAt } ] }
4. **Delete a Stock Post - DELETE** /api/posts/:postId
   * Headers: { Authorization: Bearer <token> }
   * Response: { success: true, message: 'Post deleted successfully' }

#### **Comments Management:**

1. **Add a Comment to a Post - POST** /api/posts/:postId/comments
   * Headers: { Authorization: Bearer <token> }
   * Request Body: { comment }
   * Response: { success: true, commentId, message: 'Comment added successfully' }
2. **Delete a Comment - DELETE** /api/posts/:postId/comments/:commentId
   * Headers: { Authorization: Bearer <token> }
   * Response: { success: true, message: 'Comment deleted successfully' }

#### **Like System:**

1. **Like a Post - POST** /api/posts/:postId/like
   * Headers: { Authorization: Bearer <token> }
   * Response: { success: true, message: 'Post liked' }
2. **Unlike a Post - DELETE** /api/posts/:postId/like
   * Headers: { Authorization: Bearer <token> }
   * Response: { success: true, message: 'Post unliked' }

#### **Bonus (Optional) Endpoints:**

1. **Paginated Posts Retrieval - GET** /api/posts
   * Query Parameters:
     + page (optional, default: 1)
     + limit (optional, default: 10)
   * Response: [ { postId, stockSymbol, title, description, likesCount, createdAt } ] with pagination metadata.
2. **Real-time Updates (Socket.io) - WebSocket Endpoint**
   * Updates on new comments or likes for subscribed users.

# 

# DELIVERABLES

## Code Repository

1. Code repository (preferably GitHub) with proper README explaining how to set up and run the project.
2. Clear API documentation (Swagger/Postman collection).
3. Basic testing with tools like Jest or Mocha (optional but preferred).

## Evaluation Criteria

**Code Quality:** Clean, maintainable, and well-structured code.

**Backend Design:** Efficiency of database schema, API design, and performance considerations.

**Security:** Implementation of authentication, password hashing, and JWT-based user management.

**Completion:** How well the requirements were met, including bonus features.

**Documentation:** Clarity of instructions in README, quality of API documentation.