Final Project Requirements

1. **Project Setup (10 points)**
   1) **Choose a topic.** For example: a task manager, chat app, blog platform, habit tracking app, book tracking, finance app, fitness tracker, recipes or other.
   2) Use **Node.js** and **Express** to build the server.
   3) Follow a **modular structure** with separate files for routes, models, controllers, middleware, and configuration.
   4) Include a **README.md** file with:
      a) Set up instructions,
      b) Project overview,
      c) API documentation,
      d) Screenshots of features with a short description.

2. **Database (10 points)**
   1) Use **MongoDB** for data storage.
   2) Define at least **two collections** (e.g., User (e.g., username, email, password) and Task (e.g., title, description, status, due date), User and Habit (e.g., name, description, createdAt, weeklyStatus), User and Post (e.g., title, content, author), or another relevant pairing based on your chosen topic)

3. **API Endpoints (20 points)**
   1) **Authentication (Public Endpoints)**:
      a) POST /register: Register a new user with encrypted passwords.
      b) POST /login: Authenticate users and return a **JWT** (JSON Web Token).
   2) **User Management (Private Endpoints)**:
      a) GET /users/profile: Retrieve the logged-in user's profile.
      b) PUT /users/profile: Allow users to update their profile (e.g., change email or username).
   3) **Second Collection Management (Private Endpoints)**:
      a) POST /resource: Create a new resource.
      b) GET /resource: Retrieve all resources for the logged-in user.
      c) GET /resource/:id: Retrieve a specific resource by its ID.
      d) PUT /resource/:id: Update a specific resource(e.g., mark it as completed).
      e) DELETE /resource/:id: Delete a specific resource.

4. **Authentication and Security (15 points)**
   1) Use **JWT** for secure user authentication.
      a) Protect private endpoints using middleware to verify tokens.
      b) Use **bcrypt** or a similar library for password hashing.

5. **Validation and Error Handling (5 points)**
   1) Validate incoming data (e.g., email, password, title) using middleware or libraries like **Joi** or **Validator.js**.
   2) Implement appropriate error handling:
      a) Return meaningful error messages (e.g., 400 for bad requests, 401 for unauthorized access, 500 for internal server error, 404 for not found).
      b) Use a global error-handling middleware to manage errors.

6. **Deployment (10 points)**
   1) Deploy the project to a platform like **Render**, **Replit** or **Railway**
   2) Store sensitive information (e.g., database connection strings, JWT secret) in **environment variables**.

7. **Advanced Features**
   7.1) **Role-Based Access Control (RBAC):** Add roles like "admin", "user", “premium user”, “moderator” with different access levels (e.g., admin can delete tasks, users can only update their own tasks etc). **(5 points)**
   7.2) **SMTP Email Service Integration:** Use Nodemailer combined with a service like SendGrid, Mailgun, or Postmark. (Do not use a personal email account in production; use environment variables for API keys). **(5 points)**

8. **Defence (20 points)**
   1) **Work in groups**: Collaborate with team members to complete the project. All members should contribute equally and demonstrate an understanding of the entire system during the defence.
   2) **Project Defence**: Students clearly explain the functionality, structure, and decisions behind the code during the defense. They should also answer theoretical questions related to the project and the concepts used.

Submission Guidelines

Submit the project via **LMS** by the due date in the Final Exam. Include:

1) The complete codebase (all files in the project repository) in zip format.
2) Link to GitHub repository with code and **README.md** with setup instructions and API documentation, screenshots of all web app features.
3) A URL to the deployed project.