**TASK MANAGEMENT SYSTEM - TEAM ROLE PLAN**

**🧱 BACKEND DEVELOPERS (MEMBERS 1–4)**

**👤 MEMBER 1 — DATABASE & ENTITY DESIGNER**

**Main Goal:** Build the database structure and entity classes that represent your system.

**Responsibilities:**

* Design ER Diagram and MySQL schema
* Create 3 main entities: User, Task, Role
* Define relationships between them (One-to-Many, Many-to-One)
* Add JPA annotations: @Entity, @Id, @GeneratedValue, @OneToMany, etc.
* Create UserRepository and TaskRepository interfaces
* Create data.sql with sample users/tasks for testing
* Set up database relationships and constraints in schema

**Deliverable Files:**

* model/User.java
* model/Task.java
* model/Role.java
* repository/UserRepository.java
* repository/TaskRepository.java
* database/schema.sql
* database/data.sql

**Completion Criteria:** Backend can fetch and save users/tasks properly in the database with sample data.

**👤 MEMBER 2 — TASK MANAGEMENT & BUSINESS LOGIC**

**Main Goal:** Handle all core logic for task CRUD (Create, Read, Update, Delete).

**Responsibilities:**

* Create TaskService.java with CRUD functions
* Add filtering and sorting (priority, due date, status)
* Handle pagination in results
* Integrate methods into TaskController.java
* Implement proper error handling in service layer
* Create TaskRequest.java and TaskResponse.java DTOs
* Test APIs using Postman

**Deliverable Files:**

* service/TaskService.java
* controller/TaskController.java
* dto/TaskRequest.java
* dto/TaskResponse.java

**Completion Criteria:** All /api/tasks endpoints work in Postman with proper error handling.

**Connections:** Member 1’s entities → Member 5–7’s frontend

**👤 MEMBER 3 — AUTHENTICATION & SECURITY**

**Main Goal:** Build user authentication and authorization (login/register).

**Responsibilities:**

* Create AuthController.java for login/register endpoints
* Use Spring Security with password encryption (BCryptPasswordEncoder)
* Enable role-based access (ADMIN / USER)
* Implement JWT Token-based login (REQUIRED)
* Configure CORS for React frontend
* Write UserService.java methods for registration/login
* Modify SecurityConfig.java for access control

**Deliverable Files:**

* controller/AuthController.java
* service/UserService.java
* config/SecurityConfig.java
* config/JwtAuthenticationFilter.java
* security/JwtUtil.java

**Completion Criteria:** Users can register/login with JWT tokens and CORS works with frontend.

**Connections:** Member 1 (entities) → Member 6 (React login)

**👤 MEMBER 4 — SPECIAL FEATURES & REPORTS**

**Main Goal:** Add extra functionality and make backend more advanced.

**Responsibilities:**

* Implement Email Notifications with Spring Boot Mail
* Create email templates for notifications
* Add Task Scheduler with @Scheduled for overdue tasks
* Implement Report Export (PDF/CSV)
* Implement PDF/CSV export endpoints
* Add AdminController.java for stats
* Create global exception handler

**Deliverable Files:**

* service/EmailService.java
* service/ReportService.java
* config/SchedulerConfig.java
* controller/AdminController.java
* exception/GlobalExceptionHandler.java
* util/PdfGenerator.java
* util/CsvGenerator.java
* resources/templates/email-reminder.html

**Completion Criteria:** Backend can send emails, auto-check overdue tasks, and export reports.

**Connections:** Member 1–3 → Member 8 (Admin Dashboard)

**🎨 FRONTEND DEVELOPERS (MEMBERS 5–8)**

**👤 MEMBER 5 — REACT SETUP & ROUTING**

**Main Goal:** Build the app structure and routing system.

**Responsibilities:**

* Initialize React app with create-react-app
* Setup Routing using React Router
* Create global navigation (Navbar.js)
* Create and manage AuthContext.js for login sessions
* Set up Axios interceptors for automatic token attachment
* Create protected route components
* Define all application routes

**Deliverable Files:**

* App.js
* Navbar.js
* context/AuthContext.js
* services/api.js (Axios setup)
* components/ProtectedRoute.js

**Completion Criteria:** Navigation works and Axios automatically attaches tokens to requests.

**👤 MEMBER 6 — AUTHENTICATION PAGES**

**Main Goal:** Build and connect Login and Register pages.

**Responsibilities:**

* Create Login.js and Register.js
* Add form validation with error message display
* Implement loading states during API calls
* Use Axios to call login/register endpoints
* Save JWT token in AuthContext
* Handle authentication errors gracefully

**Deliverable Files:**

* components/auth/Login.js
* components/auth/Register.js
* utils/validators.js

**Completion Criteria:** Users can successfully register & login with proper validation and error handling.

**Connections:** Member 3’s backend → Member 5’s routing

**👤 MEMBER 7 — DASHBOARD & TASK MANAGEMENT PAGES**

**Main Goal:** Build main user task interface (CRUD).

**Responsibilities:**

* Create Dashboard.js, TaskList.js, TaskForm.js
* Implement real-time task counts on dashboard
* Add task filtering and search UI
* Connect with all task APIs (GET, POST, PUT, DELETE)
* Implement pagination and sorting in UI
* Handle task creation/editing forms

**Deliverable Files:**

* pages/Dashboard.js
* components/tasks/TaskList.js
* components/tasks/TaskForm.js
* components/tasks/TaskFilters.js
* hooks/useTasks.js

**Completion Criteria:** Full task CRUD operations work from frontend with search/filter.

**Connections:** Member 2’s APIs → Member 5’s context

**👤 MEMBER 8 — ADMIN DASHBOARD & UI ENHANCEMENTS**

**Main Goal:** Create admin panel and polish UI.

**Responsibilities:**

* Create AdminDashboard.js with charts and stats
* Use Chart.js or Recharts for visualizations
* Implement responsive design throughout app
* Add export functionality in admin panel
* Apply Bootstrap/Tailwind styling
* Add SweetAlert2 for pop-ups and confirmations
* Create reusable UI components

**Deliverable Files:**

* pages/AdminPage.js
* components/admin/AdminDashboard.js
* components/admin/charts/\*
* components/common/\*
* styles/\* (all CSS files)

**Completion Criteria:** Admin dashboard shows charts and entire app is styled responsively.

**Connections:** Member 4’s admin APIs → Member 5’s routing

**🔗 CRITICAL INTEGRATION CONTRACTS**

**API CONTRACTS (BACKEND → FRONTEND)**

**1. Authentication API (Member 3 → Member 6)**

* Login: POST /api/auth/login
* Register: POST /api/auth/register

**2. Task API (Member 2 → Member 7)**

* Get Tasks: GET /api/tasks
* Create Task: POST /api/tasks
* Update Task: PUT /api/tasks/{id}
* Delete Task: DELETE /api/tasks/{id}

**3. Admin API (Member 4 → Member 8)**

* Stats: GET /api/admin/stats
* Export: GET /api/admin/export
* Users: GET /api/admin/users

**SHARED STANDARDS**

* **Error Response Format:** { "error": "message", "timestamp": "..." }
* **Date Format:** ISO format (YYYY-MM-DDTHH:mm:ss)
* **JWT Storage:** localStorage with automatic refresh
* **API Versioning:** All endpoints under /api/
* **HTTP Status Codes:** Consistent use (200, 400, 401, 404, 500)