### **Project Planning Document**

#### **Title: Todo List Application Project Plan**

### **1. Introduction**

The Todo List Application is designed to help users manage their tasks effectively. It allows users to create, read, update, and delete todos, manage groups, and assign tasks within groups. The application will support user authentication and role-based access control to ensure data security.

### **2. Functional Requirements**

1. **User Management**
   * User registration, login, and logout.
   * Send and otp email for signin, resetPassword and login(if user opted for it)
   * View and update user profile.
   * Reset Password.
2. **Todo Management**
   * Create, read, update, and delete todos.
   * Mark todos as completed.
   * Scheduled tasks should send email/notifications about it starting.
   * **Automatic Task Deletion:** Tasks will be automatically deleted from the database after three months from the date of creation/scheduled.
3. **Group Management**
   * Create, join, and leave groups.
   * Assign todos to groups.
   * Send email/notifications to the user a about tasks whoes it’s assigned too.
   * Send email/notifications to newly added member.
   * Manage group members (admin and members).
4. **Role-Based Access Control**
   * Admin and user roles.
   * Admin can manage groups and group members.
   * Users can manage their own todos and view group todos.

### **3. Non-Functional Requirements**

* **Performance**: The application should handle up to 1000 concurrent users.(not an immediate requirement)
* **Security**: User data should be encrypted, and secure communication protocols should be used.
* **Usability**: The application should be user-friendly and responsive, accessible from various devices.

### **4. High-Level Architecture**

* **Backend**: Spring Boot application with RESTful APIs.
* **Frontend**: React application.
* **Database**: MySQL and postgres data storage.
* **In-Memory Caching Database**: Redis
* **Authentication**: OAuth2 for secure user authentication.

### **Data Models and Database Design**

#### **Title: Todo List Application Database Design**

### **1. ERD Diagrams**

(Placeholder for your actual ERD diagram)

### **2. Table Structures**

* **User**
  + id: Integer (Primary Key)
  + username: String (Unique)
  + password: String
  + roles: String
* **Role (Optional , if requirement occurs)**
  + id: Integer (Primary Key)
  + name: String (Unique)
* **Group**
  + id: Integer (Primary Key)
  + name: String
  + admin\_id: Integer (Foreign Key to User)
* **Todo**
  + id: Integer (Primary Key)
  + title: String
  + completed: Boolean
  + startTime: LocalDateTime
  + startDate: LocalDate
  + endTime: LocalDateTime
  + endDate: LocalDate
  + frequency: Frequency (Enum)
  + userId: Integer (Foreign Key to Group)
* Frequency (Enum)
  + DAILY**,**
  + MONDAY**,**
  + TUESDAY**,**
  + WEDNESDAY**,**
  + THURSDAY**,**
  + FRIDAY**,**
  + SATURDAY**,**
  + SUNDAY**,**
  + NONE
* Provider(Enum)
  + LOCAL,
  + GOOGLE,
  + GITHUB
* **User\_Role (Optional , if requirement occurs)**
  + user\_id: Integer (Foreign Key to User)
  + role\_id: Integer (Foreign Key to Role)
* **Group\_Members**
  + group\_id: Integer (Foreign Key to Group)
  + user\_id: Integer (Foreign Key to User)

### **3. Relationships**

* One-to-many relationship between User and Todo.
* Many-to-many relationship between User and Role.**(Optional , if requirement occurs)**
* One-to-many relationship between Group and Todo.
* Many-to-many relationship between Group and User through Group\_Members.

### **API Design**

#### **Title: Todo List Application API Documentation**

### **1. Introduction**

This document provides details of the RESTful APIs for the Todo List Application, including endpoints, request and response formats, and authentication requirements.

### **2. Authentication**

The application uses OAuth2 for secure authentication. Users must obtain an access token to access protected endpoints.

### **3. Endpoints**

* Authentication Enpoints
  + POST /auth/login – User login via application.
  + POST /auth/sendOtp – Sends a mail consisting otp for signin, login and resetPassword.
* **User Endpoints**
  + POST /api/users - Create a new user.
  + POST /api/auth/login - User login.
  + GET /api/user/profile - Get user details.
* **Todo Endpoints**
  + GET /api/todos - Get all todos.
  + POST /api/todos - Create a new todo.
  + PUT /api/todos/{id} - Update a todo.
  + DELETE /api/todos/{id} - Delete a todo.
* **Group Endpoints**
  + POST /api/groups - Create a new group.
  + GET /api/groups - Get all groups.
  + POST /api/groups/{id}/members - Add a member to a group.
  + GET /api/groups/{id}/todos - Get todos of a group.

### **Security Design**

#### **Title: Todo List Application Security Design**

### **1. OAuth2 Implementation**

* **Grant Types**: Authorization Code, Refresh Token
* **Token Endpoint**: /oauth/token
* **Authorization Endpoint**: /oauth/authorize

#### **2.** **JWT Integration**

**Purpose**: JWTs are used for stateless authentication and authorization. They allow the backend to verify the identity of users and provide access to resources based on the user’s role.

**JWT Generation and Storage**:

* **Generation**: Upon successful login, a JWT is generated and includes the user ID and other relevant claims.
* **Storage**: JWTs are stored in HTTP-only cookies to enhance security against XSS attacks. HTTP-only cookies are used because they cannot be accessed by JavaScript, reducing the risk of token theft.

**JWT Structure**:

* **Header**: Contains metadata about the token, such as the algorithm used for signing.
* **Payload**: Includes claims like user ID , token expiration and user emailId.
* **Signature**: Ensures the token has not been tampered with.

**Implementation Steps**:

1. **Generate JWT**: Use the JWT utility class to create tokens upon user authentication. Include the user ID as a claim in the token.
2. **Store JWT**: Store the JWT in an HTTP-only cookie.
3. **Send JWT**: Include the JWT in the Authorization header for protected API endpoints.
4. **Validate JWT**: On the server side, validate the JWT, extract the user ID, and perform authorization checks.

**Handling JWTs in Requests**:

* **Frontend**: The JWT is sent with each request to protected endpoints via the Authorization header.
* **Backend**: Extract and validate the JWT from the Authorization header. Use the user ID extracted from the JWT to perform authorization checks and retrieve user-specific data.

**Security Measures**:

* **XSS Protection**: HTTP-only cookies prevent JavaScript from accessing the token.
* **CSRF Protection**: Implement CSRF tokens to prevent CSRF attacks when using cookies.
* **Token Expiry**: Use short-lived tokens to limit the impact of token theft. (Implement refresh tokens for maintaining sessions.)optional



### **3. Role-Based Access Control (RBAC)**

* **Roles**:
  + **Admin**: Can manage groups, members, and todos within groups.
  + **User**: Can manage personal todos and view group todos.
* **Permissions**:
  + **Admin**:
    - Create, update, delete groups.
    - Add and remove group members.
    - Manage todos within groups.
  + **User**:
    - Create, update, delete personal todos.
    - View group todos.

### 

### **4. Security Protocols**

* **Data Encryption**: Use HTTPS for secure data transmission.
* **Password Storage**: Store passwords using a strong hashing algorithm (e.g., bcrypt).
* **Access Control**: Implement access control checks in the backend to ensure users can only access their data or data they have permissions for.