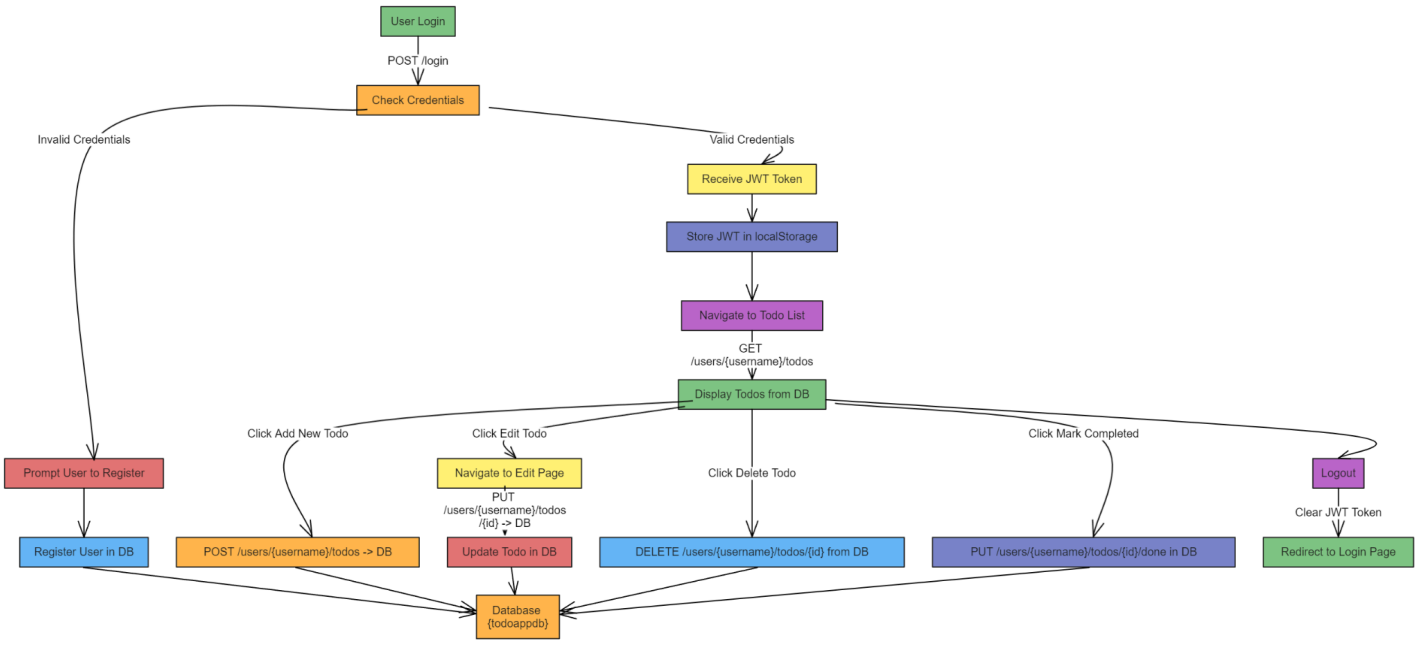
Software Documentation

1. High-Level Design (HLD)



a. Frontend Design

The frontend is a **React-based single-page application (SPA)** that communicates with the backend via **RESTful API** endpoints. The design focuses on simplicity, responsiveness, and a user-friendly interface.

- **Tech Stack**:
  - ReactJS for UI
  - MDBootstrap for styling
  - Axios for making HTTP requests
  - React Router for navigation
  - React Toastify for toast notifications
- **User Flow**:
  - **Login/Registration**: The user can log in or register to access the todo management system.
  - **Todo Management**: The user can view, create, edit, mark as completed, and delete todos.
  - **Logout**: The user can log out, which clears the session and token.
- **Components**:

    - `LoginPage`: Handles user authentication (login and registration).
    - `TodoList`: Displays the list of todos.
    - `EditTodo`: Allows the user to edit or mark a todo as completed.
    - `AddTodo`: Allows the user to add a new todo.
    - `Header`: Displays the title and logout button.
    - `Footer`: Displays credits at the bottom of the page.

b. Backend Design

The backend is a **RESTful API** built with **Spring Boot**, following a standard service-repository architecture. The API is protected by **JWT-based authentication**.

- **Tech Stack**:
    - Spring Boot for the backend framework
    - MySQL for the database
    - JWT for authentication
    - Hibernate/JPA for ORM
- **Key Functionalities**:
    - **Authentication**: Users must authenticate using JWT for accessing todo operations.
    - **Todo Management**: Provides CRUD operations (Create, Read, Update, Delete) for todos.
    - **Security**: JWT-based security ensures that only authenticated users can perform operations on their todos.
- **Services**:
    - **UserService**: Handles user authentication and registration.
    - **TodoService**: Handles business logic for managing todos.
    - **JwtService**: Generates and validates JWT tokens for secure access.

---

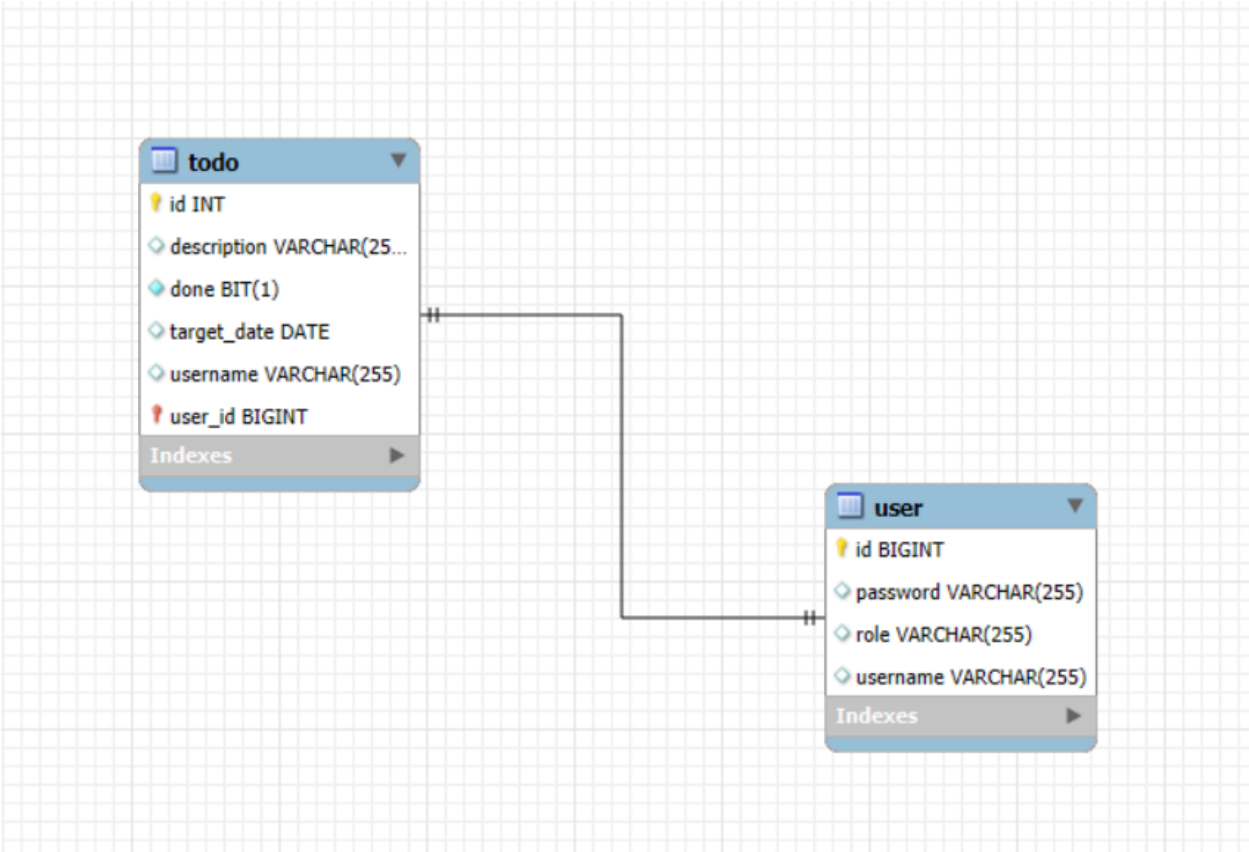
2. Entity-Relationship (ER) Diagram

Below is an **ER diagram** representing the structure of the database, specifically focusing on users and todos.

- **User Entity**:
    - `id`: Primary Key
    - `username`: Unique username
    - `password`: Hashed password
    - `email`: User email

- **Todo Entity**:
  - `id`: Primary Key
  - `description`: The description of the todo task
  - `targetDate`: Date by which the task should be completed
  - `done`: Boolean indicating if the task is completed or not
  - `user_id`: Foreign Key linking the todo to a specific user

ER Diagram Structure:



---

3. API Endpoints

The following are the **backend API endpoints** that the frontend interacts with.

a. Authentication Endpoints

1. **Login**
   - **Method**: `POST`
   - **Endpoint**: `/login`

Request Body:

```
{
      "username": "user123",
      "password": "pass123"
}
```

Response:

```
{
      "token": "jwt-token"
}
```

Description: Authenticates the user and returns a JWT token.

2. **Register**
    - **Method**: `POST`
    - **Endpoint**: `/register`

    Request Body:

```
{
"username": "user123",
"password": "pass123",

}
```

    Response:

```
{
      "message": "User registered successfully"
}
```

    Description: Registers a new user.

b. Todo Management Endpoints

3. **Get All Todos**
    - **Method**: `GET`

- **Endpoint**: `/users/{username}/todos`
- **Headers**:
  - `Authorization: Bearer <jwt-token>`

Response:

```
[
{
    "id": 1,
    "description": "Learn Spring Boot",
    "targetDate": "2024-12-31",
    "done": false
}
]
```

- **Description**: Retrieves all todos for the authenticated user.

4. **Get Todo by ID**
   - **Method**: `GET`
   - **Endpoint**: `/users/{username}/todos/{id}`
   - **Headers**:
     - `Authorization: Bearer <jwt-token>`

   Response:

```
{
    "id": 1,
    "description": "Learn Spring Boot",
    "targetDate": "2024-12-31",
    "done": false
}
```

   - **Description**: Retrieves a specific todo by its ID.

5. **Create New Todo**
   - **Method**: `POST`
   - **Endpoint**: `/users/{username}/todos`
   - **Headers**:
     - `Authorization: Bearer <jwt-token>`

   Request Body:

```
{
    "description": "New Todo",
```

```
    "targetDate": "2024-12-31",
    "done": false
}
```

Response:

```
{
    "id": 2,
    "description": "New Todo",
    "targetDate": "2024-12-31",
    "done": false
}
```

- **Description**: Creates a new todo for the user.

6. **Update Todo**
   - **Method**: `PUT`
   - **Endpoint**: `/users/{username}/todos/{id}`
   - **Headers**:
     - `Authorization: Bearer <jwt-token>`

Request Body:

```
{
    "description": "Updated Todo",
    "targetDate": "2024-12-31",
    "done": true
}
```

Response:

```
{
    "id": 1,
    "description": "Updated Todo",
    "targetDate": "2024-12-31",
    "done": true
}
```

   - **Description**: Updates an existing todo.

7. **Delete Todo**
   - **Method**: `DELETE`

- **Endpoint**: `/users/{username}/todos/{id}`
- **Headers**:
  - `Authorization: Bearer <jwt-token>`
- **Response**:
  - `204 No Content`
- **Description**: Deletes a specific todo by its ID.

8. **Mark Todo as Completed**
   - **Method**: `PUT`
   - **Endpoint**: `/users/{username}/todos/{id}/done`
   - **Headers**:
     - `Authorization: Bearer <jwt-token>`

   Response:

```
{
    "id": 1,
    "description": "Updated Todo",
    "targetDate": "2024-12-31",
    "done": true
}
```

   - **Description**: Marks a specific todo as completed.