To-Do List Application Documentation

Table of Contents

1. Introduction
2. API Endpoints
   - User Endpoints
   - ToDo Endpoints
3. Codebase Overview
   - Project Structure
   - Key Classes and Services
4. Database Schema
5. Security Considerations

Introduction

The To-Do List Application is a full-stack web application designed to manage tasks efficiently. The backend is built with Java and Spring Boot, while the frontend can interact with the backend through a set of RESTful API endpoints. The backend handles CRUD operations, user authentication, and authorization using JWT tokens.

API Endpoints

User Endpoints

`POST /usertodo/signup`

Description: Registers a new user in the system.

Request Body:

```
{
"name": "John Doe",
"email": "john.doe@example.com",
"password": "securepassword"
}
```

Response:

`200 OK`: Returns the created user object.

`400 Bad Request`: If the request data is invalid.

`POST /usertodo/login`

Description: Authenticates a user and returns a JWT token.

Request Body:

```
{
  "email": "john.doe@example.com",
  "password": "securepassword"
}
```

Response:

- `200 OK`: Returns a JWT token and a success message.
- `401 Unauthorized`: If the email or password is incorrect.

ToDo Endpoints

`GET /todoapi/getAllToDoList`

Description: Retrieves all ToDo items for the authenticated user.
Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Response:

- `200 OK`: Returns a list of ToDo items.
- `401 Unauthorized`: If the token is invalid or expired.

`GET /todoapi/getToDoById/{id}`

Description: Retrieves a specific ToDo item by its ID.
Path Parameters:

- `id` (Long): The ID of the ToDo item.

Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Response:

- `200 OK`: Returns the ToDo item.
- `404 Not Found`: If the ToDo item with the specified ID does not exist.
- `401 Unauthorized`: If the token is invalid or expired.

`POST /todoapi/saveToDo`

Description: Creates a new ToDo item.
Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Request Body:

```
{
  "title": "New Task",
  "description": "Description of the new task",
  "dueDate": "2024-08-30",
  "status": "Pending"
}
```

Response:

- `200 OK`: Returns the ID of the newly created ToDo item.
- `500 Internal Server Error`: If an error occurs during the creation.
- `401 Unauthorized`: If the token is invalid or expired.

`PUT /todoapi/updateToDoById/{id}`

Description: Updates an existing ToDo item by its ID.

Path Parameters:

- `id` (Long): The ID of the ToDo item to update.

Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Request Body:

```
{
  "title": "Updated Task",
  "description": "Updated description",
  "dueDate": "2024-09-01",
  "status": "Completed"
}
```

Response:

- `200 OK`: Returns the updated ToDo item.
- `404 Not Found`: If the ToDo item with the specified ID does not exist.
- `401 Unauthorized`: If the token is invalid or expired.

`DELETE /todoapi/deleteToDoById/{id}`

Description: Deletes a specific ToDo item by its ID.

Path Parameters:

- `id` (Long): The ID of the ToDo item to delete.

Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Response:

- `200 OK`: Returns a success message if the item is deleted.
- `404 Not Found`: If the ToDo item with the specified ID does not exist.
- `401 Unauthorized`: If the token is invalid or expired.

`GET /todoapi/search`

Description: Searches for ToDo items based on a search key.
Headers:

- `Authorization: Bearer <JWT_TOKEN>`

Query Parameters:

- `key` (String): The search key to filter ToDo items by title.

Response:

- `200 OK`: Returns a list of ToDo items that match the search key.
- `401 Unauthorized`: If the token is invalid or expired.

Codebase Overview

Project Structure

The project is structured as follows:

```
/src
  /main
    /java/com/TodoList/todo
      ├── controller          // Controllers for handling API requests
      ├── model               // Entity classes representing the database table
      ├── repository          // Repository interfaces for database operations
      ├── service             // Business logic and service classes
      ├── util                // Utility classes like JWTTokenUtil
    /resources
      ├── application.properties // Spring Boot application configuration
  /test
    /java/com/TodoList/todo
```

Key Classes and Services

- **ToDoController**: Handles all CRUD operations related to ToDo items. It interacts with the ToDoService to perform business logic and return appropriate HTTP responses.
- **UserController**: Manages user signup and login. It interacts with UserService to handle user authentication and authorization.
- **ToDoService**: Contains business logic for managing ToDo tasks, such as retrieving, saving, updating, and deleting tasks. It also handles token validation and user-specific queries.
- **UserService**: Responsible for user-related operations, such as saving new users and validating login credentials.
- **JWTTokenUtil**: Utility class for generating, parsing, and validating JWT tokens.
- **TaskRepository**: Extends JpaRepository and provides methods for interacting with the tasks table in the database.
- **UserRepository**: Extends JpaRepository and provides methods for interacting with the users table in the database.

Database Schema

The application uses a MySQL database with the following schema:

todotable

- id (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each ToDo item.
- title (VARCHAR(255)): The title of the ToDo item.
- description (VARCHAR(255)): The description of the ToDo item.
- dueDate (VARCHAR(255)): The due date of the ToDo item.
- isCompleted (BOOLEAN): The completion status of the ToDo item.
- created_at (TIMESTAMP): The timestamp when the ToDo item was created.

user

- id (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for each user.
- name (VARCHAR(45)): The name of the user.
- email (VARCHAR(45)): The email address of the user.
- password (VARCHAR(255)): The hashed password of the user.
- modified_date (TIMESTAMP): The timestamp when the user was last modified.

Relationships

- A task is linked to a user via user_id in the tasks table, establishing a foreign key relationship.

Security Considerations

- **Password Storage**: Passwords are stored using hashing techniques (e.g. JWT) to ensure that even if the database is compromised, raw passwords are not exposed.
- **JWT Authentication**: JWT tokens are used for authenticating API requests. These tokens are generated upon user login and must be included in the `Authorization` header for subsequent requests.
- **Token Validation**: Each token has an expiration time and is validated on each request to ensure it's still valid and corresponds to the correct user.
- **Cross-Origin Resource Sharing (CORS)**: The application uses CORS to restrict requests from untrusted sources. Ensure that only trusted domains are allowed in production environments.