User API Design Documentation

1. Introduction

The User API will handle all User operations including administrative (Create, Read, Update, Delete) and end user operations (Login, Read, Update, Delete). The API will also handle the login process to only allow access to registered users with a valid account.

An admin will be able to utilise all CRUD operations to manage users, reset passwords, etc but will still need to use the login service to verify administrative privileges (Spring Security roles/granted authorities).

An end user will have limited access to CRUD operations involving their own account such as being able to reset their password, update their personal details, and delete their account. They will use the login service to verify their account using a username (email) and password.

1. Requirements

CRUD API to manage users stored in tbl\_users db; depending on the role of the end user (Admin or Customer) it will be able to perform a variety of CRUD operations outlined in section 3.

All input/output data will be in JSON format, example below:

{

“userId”: 10002,

“username”: “vfwood”,

“passwordHash”: “e443d94662sjems9”

“failedLoginAttempts”: 0

“dateAdded”: “2023-04-24 00:20:34”

“dateLastModified”: “2023-04-24 00:20:34”

“isDeleted”: false

}

The user API will need to be secure; no plaintext passwords, using a valid password encoder which utilises an additional ‘salt’ to mitigate brute force cracking.

The user API will also handle the login/authentication process, checking provided passwords (hashed) against those in the database. The API will also handle requests to an abstracted Token management API.

1. User API Endpoints

|  |  |  |  |
| --- | --- | --- | --- |
| METHOD | ENDPOINT | INPUT | OUTPUT |
| POST | /users | {  “username”: “user@email.com”,  “passwordHash”: “e443d94662sjems9”  “failedLoginAttempts”: 0  “dateAdded”: “2023-04-24 00:20:34”  “dateLastModified”: “2023-04-24 00:20:34”  “isDeleted”: false  } | **Success:**  HTTP 200 – OK  “Account Created Successfully”  **Error:**  HTTP 400 – Bad Request  Message dependent on reason for error. |
| GET | /users | na | {  {  “username”: “vfwood”,  “passwordHash”: “e443d94662sjems9”  “failedLoginAttempts”: 0  “dateAdded”: “2023-04-24 00:20:34”  “dateLastModified”: “2023-04-24 00:20:34”  “isDeleted”: false  },  {  “username”: “amyK123”,  “passwordHash”: “e44rrr662sjems9”  “failedLoginAttempts”: 1  “dateAdded”: “2023-04-24 00:20:34”  “dateLastModified”: “2023-04-24 00:20:34”  “isDeleted”: false  }  } |
| GET | /users/{id} | Integer: 10002 | {  “username”: “vfwood”,  “passwordHash”: “e443d94662sjems9”  “failedLoginAttempts”: 0  “dateAdded”: “2023-04-24 00:20:34”  “dateLastModified”: “2023-04-24 00:20:34”  “isDeleted”: false  } |
| PUT | /users/{id} | {  “username”: “vfwood”,  “passwordHash”: “e443d94662sjems9”  “failedLoginAttempts”: 0  “dateAdded”: “2023-04-24 00:20:34”  “dateLastModified”: “2023-04-24 00:20:34”  “isDeleted”: false  } | **Success:**  HTTP 204 – No Content  “Updated account details successfully”  **Error:**  HTTP 400 – Bad Request  Message dependent on reason for error. |

Users Table

* POST /users: Create a new user
* GET /users: Retrieve all users
* GET /users/{id}: Retrieve an existing user
* PUT /users/{id}: Update an existing user
* DELETE /users/{id}: Delete an existing user