Bus management system

Documentation

Mazen Belal

Ahmed Hesham  
Ahmed Ayman

Ahmed Mohamed Shawky: 221004411

|  |
| --- |
| 1.User Management System |

A diagram of a computer

Description automatically generatedMost of the code regarding user management (**besides user authentication/authorization)** can be found in package “user”, defines different types of users, users’ repositories and users’services.   
*Package ‘user’ folder structure*

## User Class:

The **User** class is an **abstract base class** that provides a shared structure for all user types. This class implements the **UserDetails** interface, which is part of **Spring Security**.

## CLass annotations

* **@Slf4j**: Enables logging using the **Lombok** logging utility. It provides an easy way to log messages for debugging and tracking.
* **@NoArgsConstructor**: Generates a no-arguments constructor using **Lombok**.
* **@AllArgsConstructor**: Generates a constructor with arguments for all class fields using **Lombok**.
* **@Entity**: Marks this class as a JPA entity, making it a persistent object that maps to a table in the database.
* **@Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS)**: Indicates the inheritance strategy where each subclass will have its own table.

## properties of user Class

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Id | Integer | Unique identifier for each user (Primary Key). Automatically generated. |
| name | String | The name of the user. |
| Email | String | The email address used as the username for authentication. Must be unique. |
| password | String | Encrypted password for authentication. |
| Role | Role(enum) | Represent the Role that will be used to authorization of specific routes  *USER*, *ADMIN* |

## What is the USer Details class?

the **UserDetails** interface is a part of **Spring Security**. That allows the **User** class to be used for Spring Security's **Authentication** and **Authorization** processes.

*“Provides core user information.*

*Implementations are not used directly by Spring Security for security purposes. They simply store user information which is later encapsulated into* [*Authentication*](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/Authentication.html) *objects. This allows non-security related user information (such as email addresses, telephone numbers etc) to be stored in a convenient location.*

*Concrete implementations must take particular care to ensure the non-null contract detailed for each method is enforced. See* [*User*](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/User.html) *for a reference implementation (which you might like to extend or use in your code).”* Reference: [docs.spring.io](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html)

## User details methods

|  |  |  |
| --- | --- | --- |
| Method | Return Type | Description |
| getAuthorities() | Collection<? extends GrantedAuthority> | Returns the list of authorities (roles) assigned to the user. In this implementation, the role is converted into a **SimpleGrantedAuthority**. |
| getPassword() | String | Returns the encrypted password of the user. |
| getUsername() | String | Returns the email as the username for authentication purposes. |
| isAccountNonExpired() | boolean | Returns true, meaning the user account is always non-expired. This can be customized if needed. |
| isAccountNonLocked() | boolean | Returns true, meaning the user account is not locked. This can be customized if needed. |
| isCredentialsNonExpired() | boolean | Returns true, meaning the user’s credentials never expire. This can be customized if needed. |
| isEnabled() | boolean | Returns true, meaning the user is always active. This can be customized if needed. |

## User types:

{APP\_NAME} supports four account types that the user can be:

* **Student**
* **Parent**
* **Driver**
* **Admin**

Each of these user types inherits from the abstract **User** class, which defines the shared attributes and behaviors of all users. However, each user type also has its own **class**, **repository**, and **UserDetailsService** to handle account type specific logic and storage.

To manage user authentication, all these types are linked via a **ComposedDetailsService**, which aggregates the individual **UserDetailsService** for each user type. This architecture allows the system to handle login attempts for multiple user roles in a unified manner. We will talk about the behavoir of our user authentication and authorization in their section.

## Structure of User types

Each user type has the following components:

|  |  |
| --- | --- |
| Component | Description |
| User Class | a JPA entity class for each user type, inheriting from the base User class. |
| User Repository | A JPA repository that contains the basic API CRUD operations and also the API for pagination and sorting. To the SQL database |
| UserDetailsService | A service that **finds user details** (like username and password) to let users log in using the user’s repository to communicate with the SQL database |

## Componetns for each user type

**1.User Class**

Each user type has its own class that inherits from the base User class. These classes may have additional properties and relationships that are specific to that role.

***Note:We will be using the Student a type moving forward***



* The **Student** class inherits from the **User** class.
* The **@Entity** annotation indicates that this class is a JPA entity and will be mapped to a database table called **Student**.
* **@OneToOne** relationship links the student with their **parent**.
* The **Builder** pattern is used to create new instances of the **Student** class.

Similar classes are created for **Parent**, **Driver**, and **Admin**, each with specific properties and relationships as needed.

**2.Repository for Each User Type**Each user type has its own repository to interact with the database.

****

This is a **JPA Repository** for **Student** objects.

* It provides standard CRUD operations for **Student** entities.
* It includes a custom query method, **findByEmail**, to retrieve a **Student** by their **email**.

Repositories for **Parent**, **Driver**, and **Admin** are implemented similarly. Each of them extends **JpaRepository** and may have custom query methods as needed.

**3.UserDetailsService**

To support Spring Security, each user type has its own **UserDetailsService**.  
The **UserDetailsService** is responsible for finding user details (like username and password) during the login process.



The **StudentDetailsService** implements the **UserDetailsService** interface.

* It retrieves **Student** details from the **StudentRepository** using **findByEmail**.
* If no user is found, it throws a **UsernameNotFoundException**.
* Similar **UserDetailsService** classes exist for **Parent**, **Driver**, and **Admin**.

**4.ComposedDetailsService**

The **ComposedDetailsService** aggregates all the **UserDetailsService** implementations for each user type (Student, Parent, Driver, Admin) into a single service. This approach allows the system to attempt login for any user type using the same entry point.

**HOW DOES THE COMPOSEDDETAILSSERVICES BEHAVE?**  
**PostConstruct Initialization**:

* When the service is initialized, the **setServices()** method runs.
* It creates a list of **UserDetailsService** objects for **Student**, **Parent**, **Driver**, and **Admin**.

**Authentication Process**:

* When **loadUserByUsername()** is called, it loops through all the services in the list.
* It attempts to find a user with the provided username.
* If a user is found, it returns the user details.
* If no user is found in any of the services, a **UsernameNotFoundException** is thrown.



2.Authentication/Authorization

& Routing

3213