SC2002
SCMA Group3
Hospital Management System

Aanya, Jervis, Ke Jun, Maeko, Viona

TABLE OF CONTENTS

01 PROJECT INTRO

02 CODE IMPLEMENTATION

03 TEST CASES

04 CONCLUSION

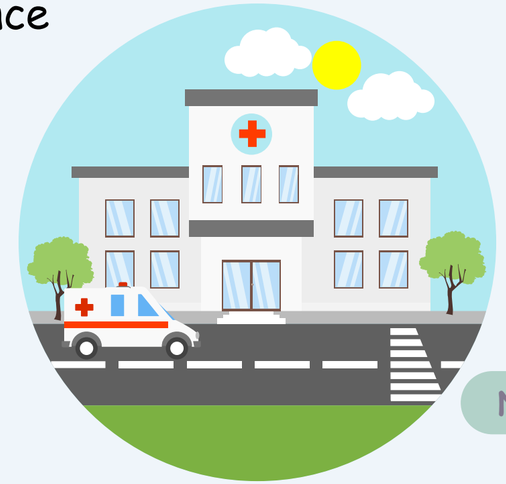
BACK

NEXT

WHAT IS HMS

Why we need **Hospital Management System**:

- Provides a user-friendly interface to streamline hospital operations
- Enables different roles to efficiently fulfill their responsibilities
- Enhances coordination among staff through a centralised HMS
- Creates a smooth, user-centered hospital experience



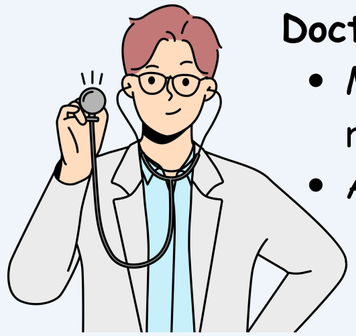
BACK

NEXT

ROLES

Patient

- Information access
- Manage own appointments



Doctor

- Manage patients' medical records
- Appointment scheduling

Pharmacist

- Prescription management
- Inventory management

Administrator

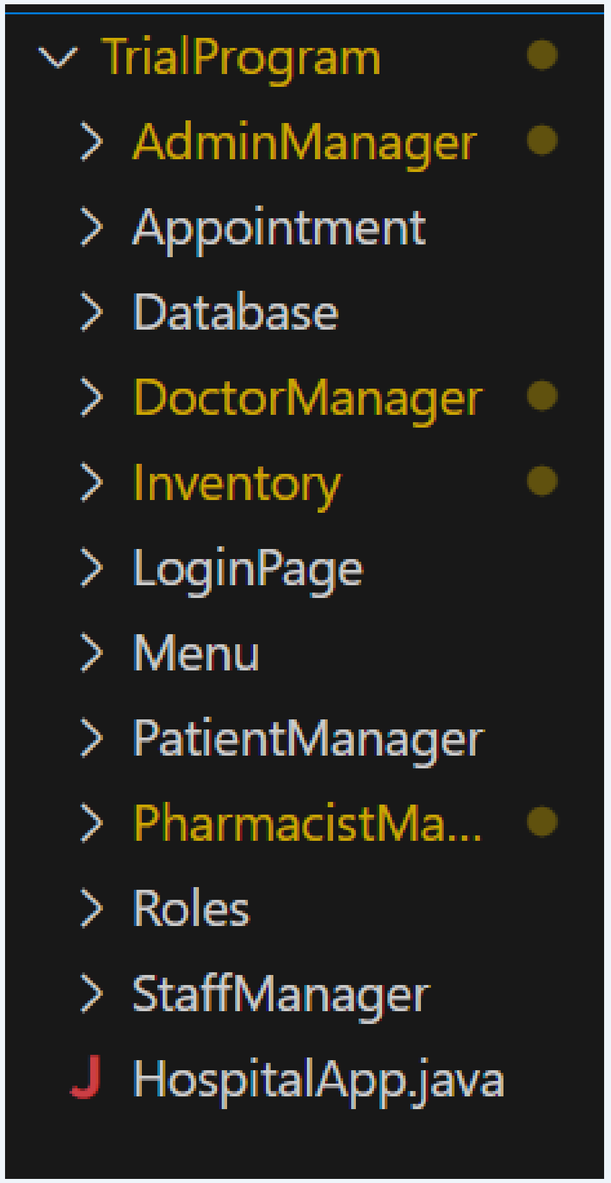
- Staff management
- View appointments
- Inventory management

BACK

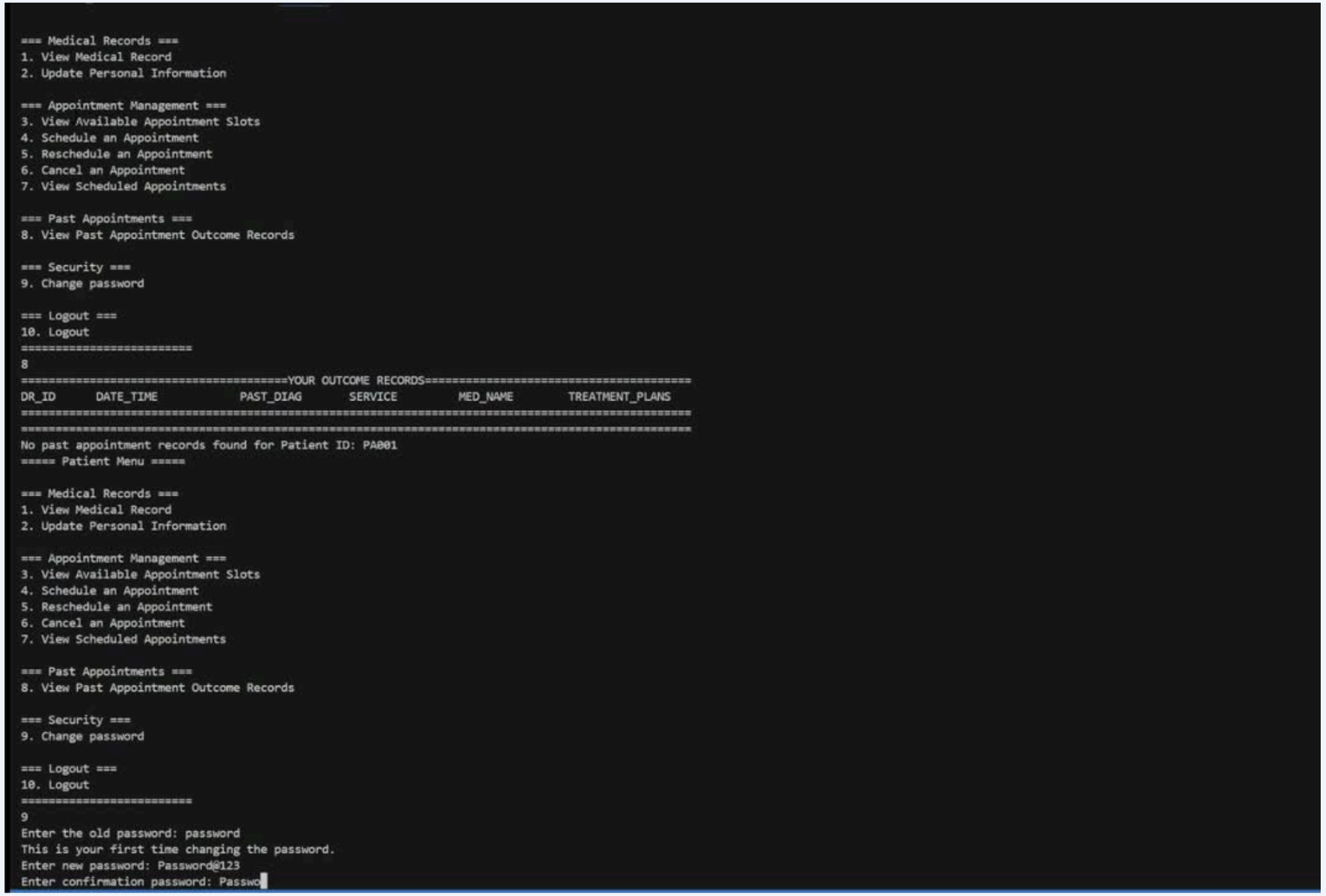
NEXT

CODE IMPLEMENTATION

We organised our code into the folowing folders:



USER LOGIN & CHANGE PWD



BACK

NEXT

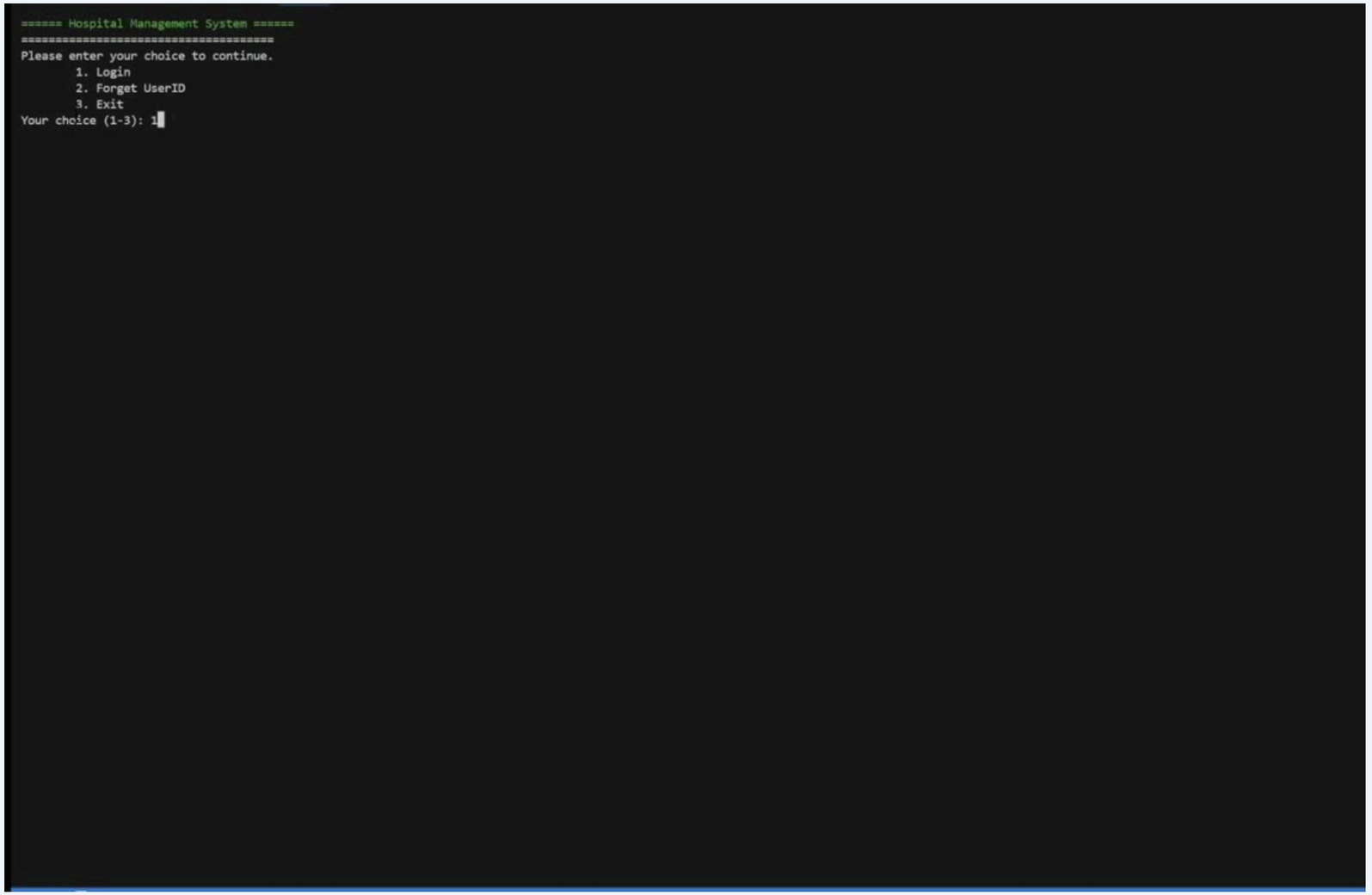
Code Implementation

PATIENT TEST CASES

BACK

NEXT

PATIENT



BACK

NEXT

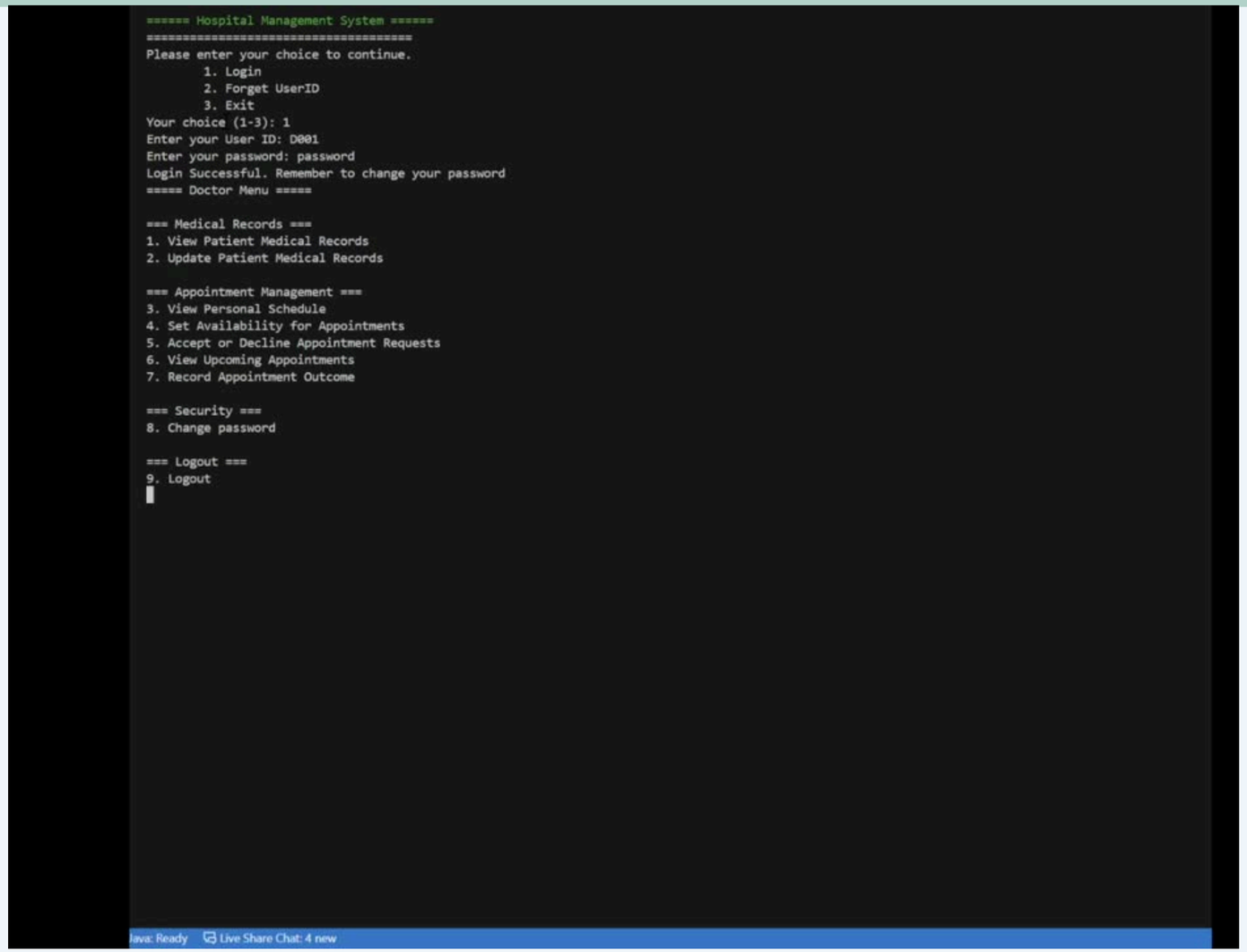
Code Implementation

DOCTOR TEST CASES

BACK

NEXT

DOCTOR



BACK

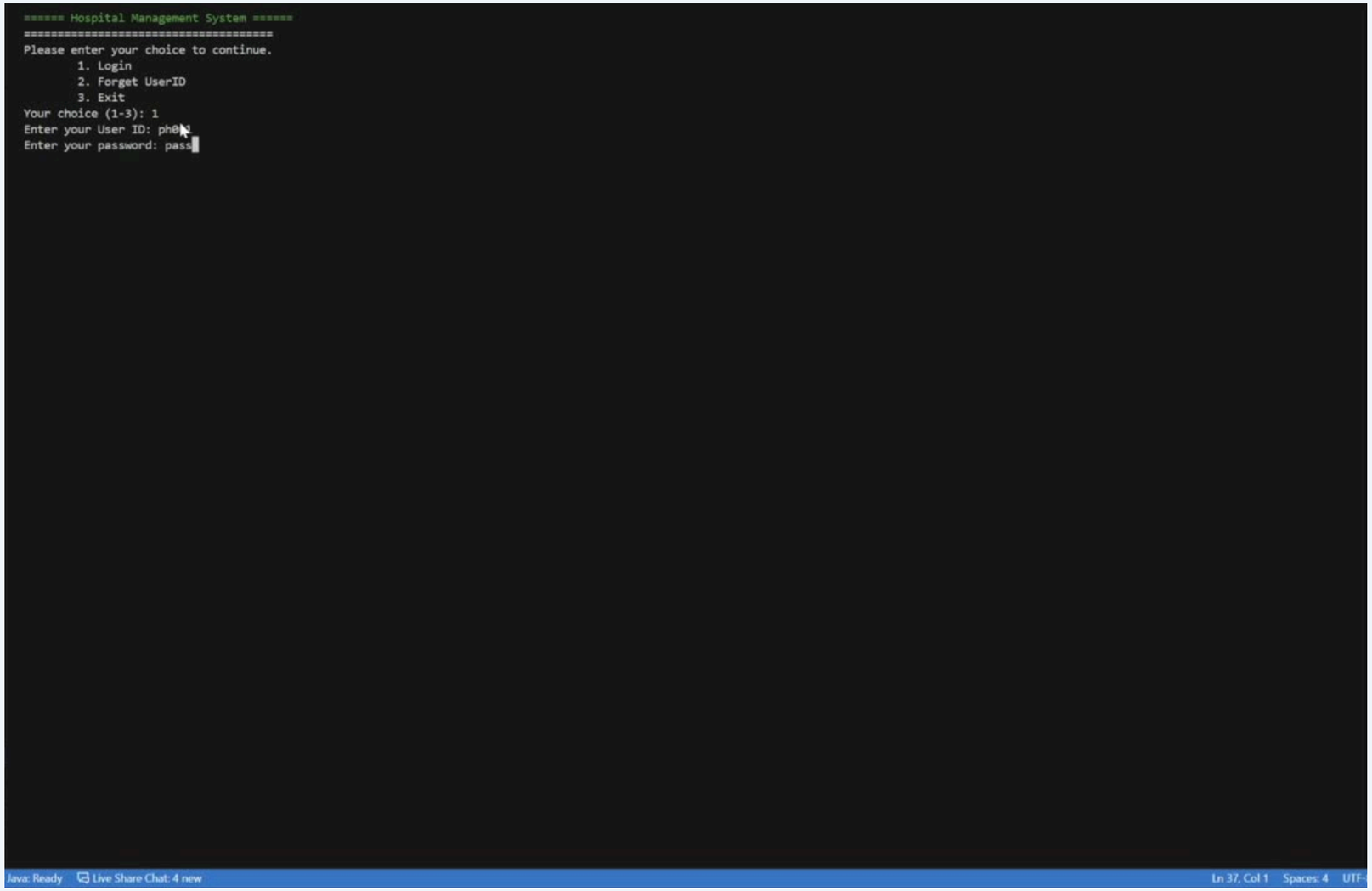
NEXT

PHARMACIST TEST CASES

BACK

NEXT

PHARMACIST



BACK

NEXT

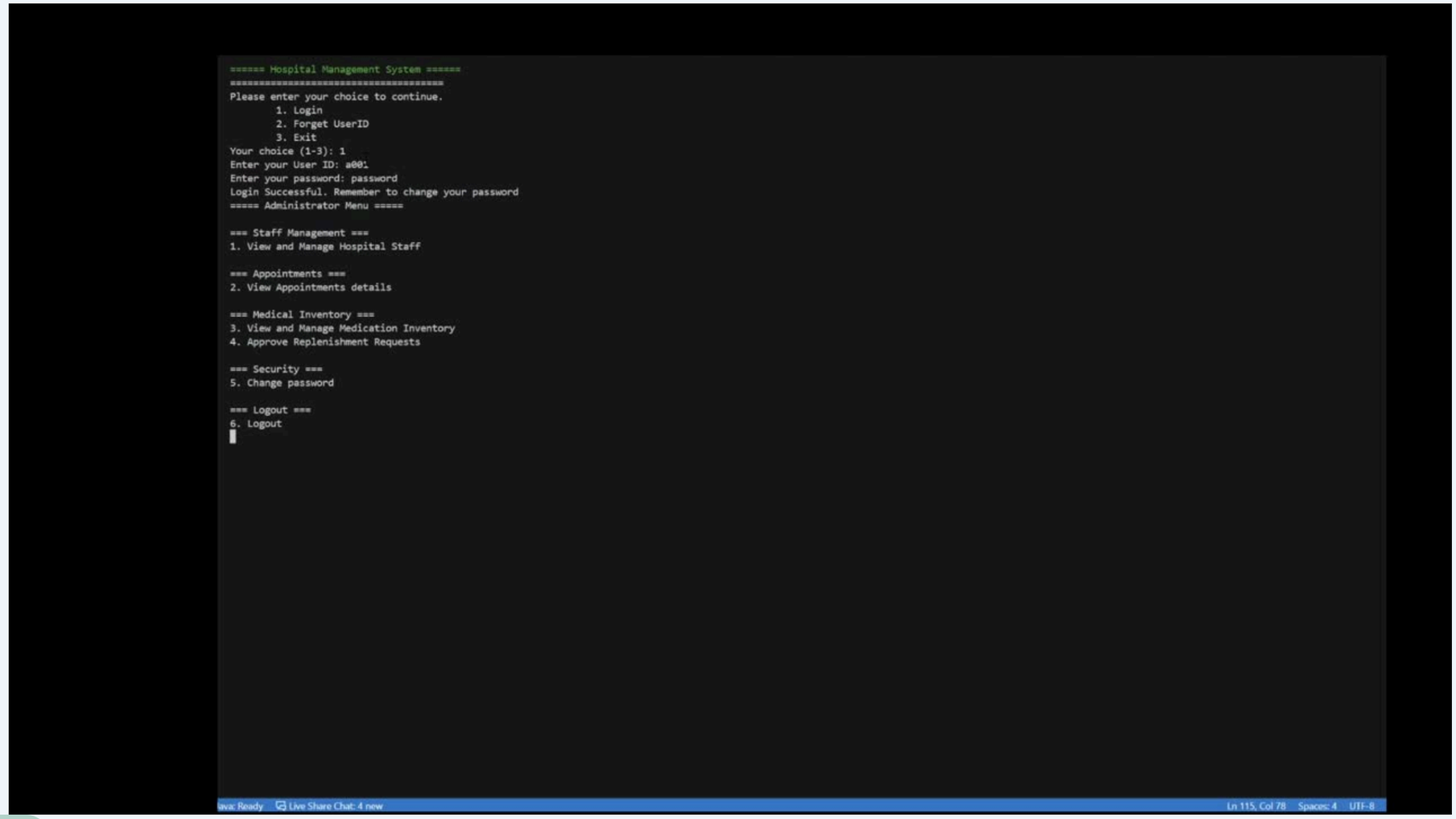
Code Implementation

ADMIN
TEST CASES

BACK

NEXT

ADMIN



BACK

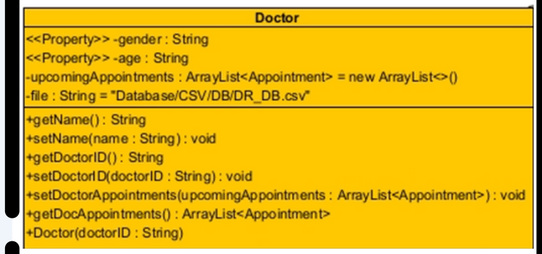
NEXT

DESIGN PRINCIPLES USED

Model-View-Controller (MVC)

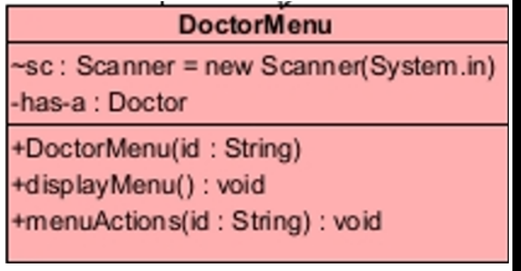
Model

- Handles data and rules
- Manages data for different roles (e.g. Doctor, Patient, etc.)



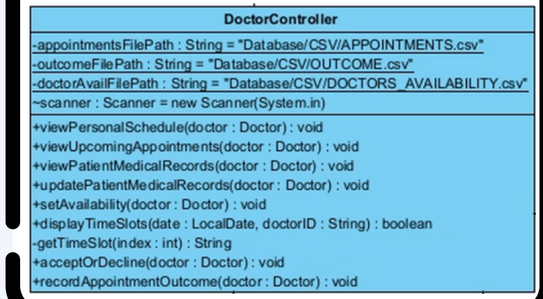
View

- Handles user interface rendering
- Implemented in DoctorMenu, PatientMenu etc.



Controller

- Connects Model & View
- Handles method implementation (DoctorController, PharmacistController etc.)



BACK

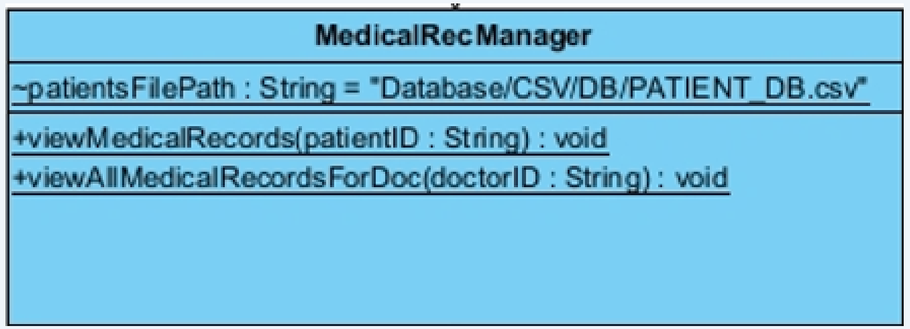
NEXT

SOLID PRINCIPLES

SINGLE RESPONSIBILITY PRINCIPLE

Every class should have one and only **one** responsibility

- Ensure every class has a **distinct** role
- **DoctorSchedule.java's** role is to manage/ update doctor's schedule



- **MedicalRecManager.java's** role is to allow Patient/ Doctor to view Medical Records

BACK

NEXT

OPEN/CLOSED PRINCIPLE

Classes should be open for **extension** but closed to modification

```java
package Menu;

import LoginPage.ChangePassword;

public abstract class UserMenu {
    public UserMenu() {
    }

    public void displayMenu() {
        System.out.println("This ID has no assigned role.");
    }

    public void menuActions(String var1) {
        System.out.println("This ID has no assigned role.");
    }

    public void clearConsole() {
        System.out.print("\u001b[H\u001b[2J");
        System.out.flush();
    }

    public void changePassword(String var1) {
        ChangePassword.changePassword(var1);
    }
}
```

- Other classes extend UserMenu (abstract class)
- **Scalability**: New features can be added without disrupting the existing code
- **Stability**: Core code remains unchanged, reducing the risk of bugs

SOLID PRINCIPLES

LISKOV SUBSTITUTION PRINCIPLE

```
public void displayMenu(String id) {
    UserMenu userMenu;
    String role = findRoleByID(id);
    switch(role) {
            case "PATIENT":
                userMenu = new PatientMenu();
                break;
            case "DOCTOR":
                userMenu = new DoctorMenu();
                break;
            case "PHARMACIST":
                userMenu = new PharmacistMenu();
                break;
            case "ADMINISTRATOR":
                userMenu = new AdminMenu();
                break;
            default:
                System.out.println(x:"Invalid Role. ");
                return;
        }
    userMenu.menuActions(id);
}
```

- Objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
- The objects of our subclasses (E.g. Patient, Doctor) will behave in the same way as the objects of our superclass (E.g. User)
- Each role will override the menuActions() method to carry out the actions specific to their roles

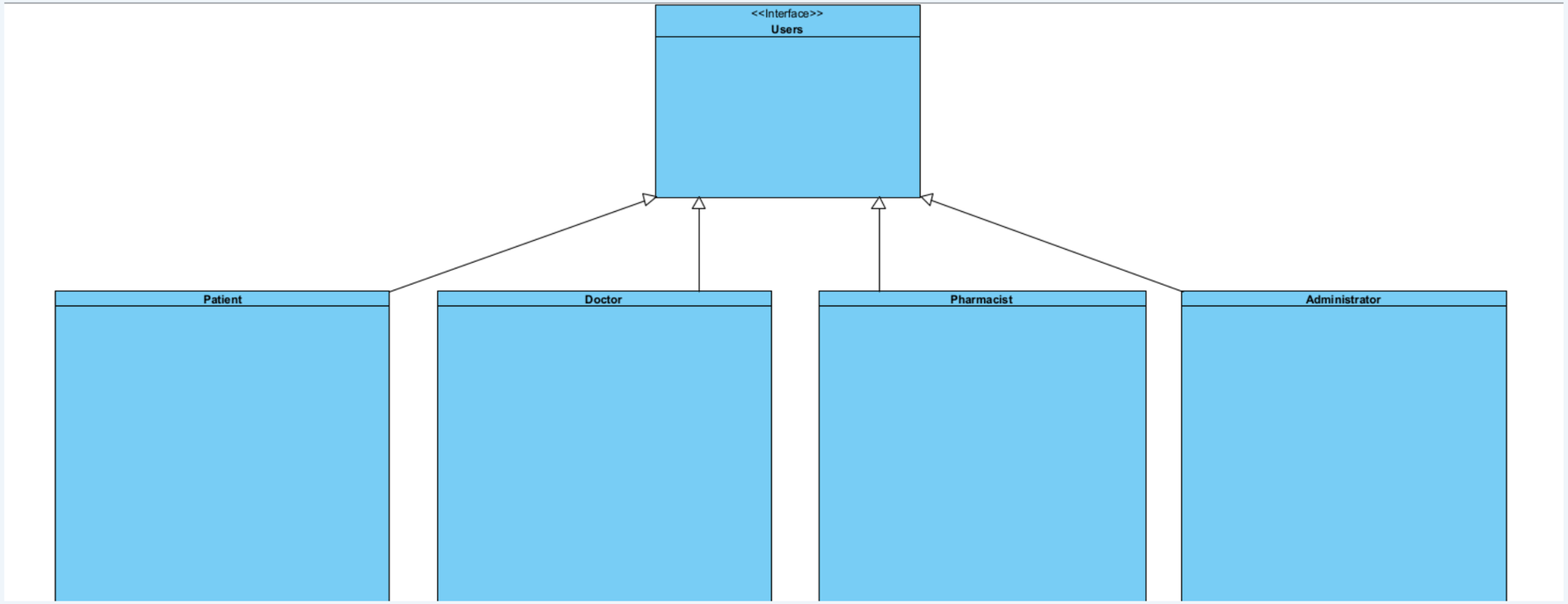
BACK

NEXT

INTEGRATION SEGREGATION PRINCIPLE

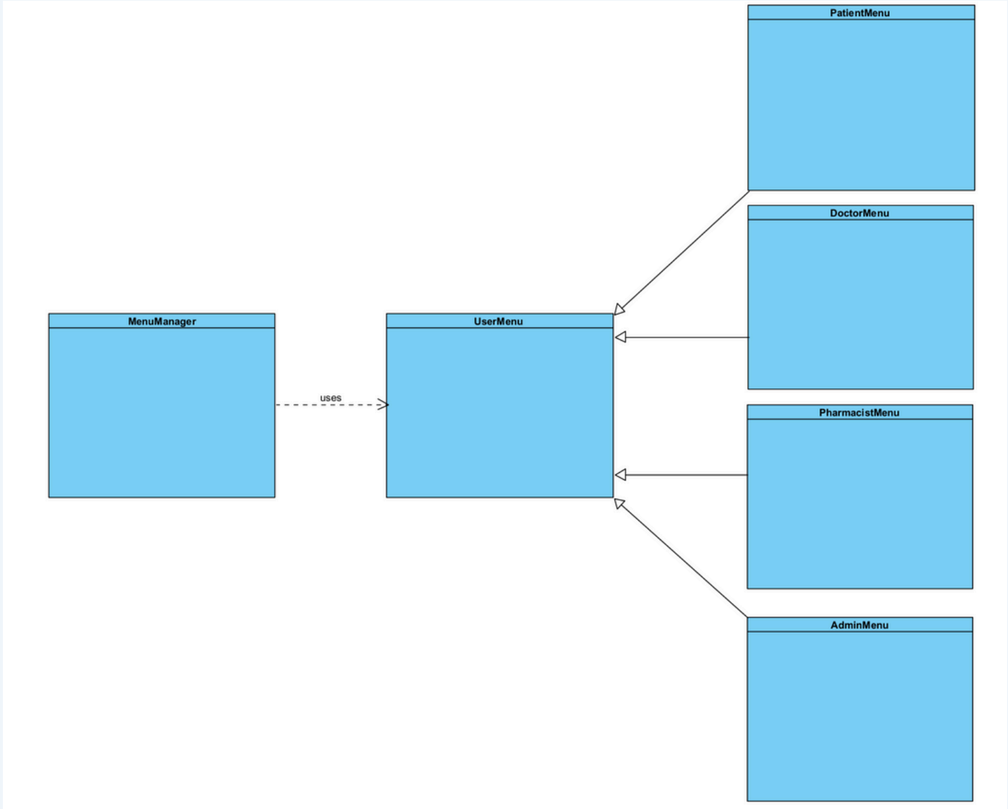
A class should not be forced to implement methods it doesn't need

Each role will only implement the methods that it needs and avoid unnecessary dependencies



SOLID PRINCIPLES

DEPENDENCY INVERSION PRINCIPLE



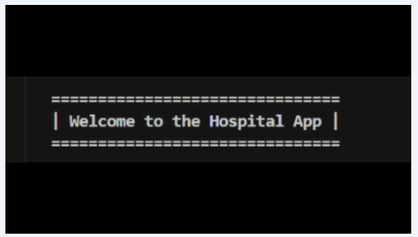
- High-level modules should not depend on low-level modules.
- Abstractions should not depend on details. Details should depend on abstractions.
- MenuManager (High-level module) is depending on abstraction class UserMenu
- Each role has a menu (Low-level module) that implements the UserMenu

BACK

NEXT

Additional Features

Password Hashing



Blinking Interface

Forget UserID

Clear Console

BACK

NEXT

THANK YOU