Clinic Management System implemented using Spring

Authors: Farah Tarek, Farah Waleed, Mariam Kandeel, Rawan Waleed, & Omar Hany

[a]*Supervised by: Dr. Hisham Mansour and Dr. Moamen Zaher at MSA University Faculty of Computer Science*

Abstract

The Clinical Management System (CMS) is a critical software system that streamlines clinical operations for administrative, medical, and support staff. This paper provides a comprehensive overview of the CMS, including its key design patterns and functional requirements. The CMS incorporates features such as appointment scheduling, prescription administration, patient/doctor management, and authentication to simplify day-to-day clinical workflows. To demonstrate the system's capabilities and interconnections, the paper delves into the precise functions and sub-functions of each CMS component. Notably, the implementation leverages the Spring Framework and Dependency Injection (DI) to promote modularity, testability, and maintainability by decoupling object creation from usage, as well as, Dependency Injection Principle this principle depends on interfaces rather than concrete classes.

1. Purpose of the CMS

This document's main goal is to give a thorough overview of the Clinical Management System (CMS), including information on its design patterns and functional needs. By including features like appointment scheduling, prescription administration, patient or doctor management, and authentication, the CMS simplifies clinical operations for the administrative, medical, and support staff. To provide clarity in the system's capabilities and interconnections, each component is described with precise functions and sub-functions. The document also explains the use of the Spring Framework for implementing Dependency Injection (DI), which promotes modularity, testability, and maintainability by decoupling object creation from usage. This foundational guide aims to ensure a shared understanding among developers, stakeholders, and project managers, facilitating efficient development, deployment, and future scalability of the CMS.

2. Project problem

The current Clinical Management System (CMS) used by the healthcare organization is not working well. Patient information is scattered across different systems, making it hard for staff to access and update records. Scheduling appointments is a manual process that often leads to conflicts and miscommunications between patients and providers. Keeping track of patient prescriptions is also a challenge, with difficulties in coordinating refills and renewals. On top of that, the CMS has a complex and confusing user interface, making it difficult for the entire clinical staff to efficiently use the system. To solve these problems, the organization needs to develop a new CMS that can better manage patient information, appointments, and prescriptions in a more efficient and user-friendly way.

3. Project goals

- Create an efficient system for handling patient information, appointments, and prescriptions in order to improve clinical operations' efficiency.
- Improved Accessibility: Provide an intuitive user interface that medical staff, administrators, physicians, assistants, and other personnel can all utilise to easily navigate and utilise the system's features.
- Efficient Appointment Scheduling: Facilitate the easy planning and administration of patient appointments, minimising conflicts and enhancing communication between the doctor and the patient.

4. Functional Requirements

This document's main goal is to give a thorough overview of the Clinical Management System (CMS), including information on its design patterns and functional needs. By including features like appointment scheduling, prescription administration, patient or doctor management, and authentication, the CMS simplifies clinical operations for the administrative, medical, and support staff. To provide clarity in the system's capabilities and interconnections, each component is described with precise functions and sub-functions. The document also explains the use of the Spring Framework for implementing Dependency Injection (DI), which promotes modularity, testability, and maintainability by decoupling object creation from usage. This foundational guide aims to ensure a shared understanding among developers, stakeholders, and project managers, facilitating efficient development, deployment, and future scalability of the CMS.

Table 1: Admin Functionality

Requirement id	Name of requirements	Requirement description
ADM001	Admin Login	Allows the admin to securely log into the system using their credentials
ADM002	Manage Patients	Provides CRUD and view operations for patient records credentials
ADM002B	Update Patient	Allows the admin to update existing patient records, modifying details like contact information and medical history.
ADM002C	Delete Patient	Allows the admin to delete patient records from the system, removing all associated data.
ADM002D	View Patients	Allows the admin to view a list of all patients and their details, including the ability to filter and search for specific records.
ADM003	Manage Doctors	Provides CRUD and view operations for doctor records.
ADM003A	Add Doctor	Allows the admin to add new doctor records into the system, including entering details such as name, specialty, and contact information.
ADM003B	Update Doctor	Allows the admin to update existing doctor records, modifying details like specialty and contact information.

Table 2: Admin Functionality Continued

ADM003C	Delete Doctor	Allows the admin to delete doctor records from the system, removing all associated data.
ADM003B	Update Doctor	Allows the admin to update existing doctor records, modifying details like specialty and contact information.
ADM003C	Delete Doctor	Allows the admin to delete doctor records from the system, removing all associated data.
ADM003D	View Doctors	Allows the admin to view a list of all doctors and their details, credentials including the ability to filter and search for specific records
ADM003C	Delete Doctor	Allows the admin to delete doctor records from the system, removing all associated data.
ADM003D	View Doctors	Allows the admin to view a list of all doctors and their details, credentials including the ability to filter and search for specific records

Table 3: Doctor Functionality

Requirement id	Name of requirements	Requirement description
DOC001	Doctor Login	Allows the doctor to securely log into the system using their credentials.
DOC002	Manage Prescriptions	Allows the doctor to add, delete, and edit prescriptions by patient name.
DOC002A	Add Prescription	Allows the doctor to add a new prescription for a patient, including details like medication, dosage, and duration.
DOC002B	Edit Prescription	Allows the doctor to edit an existing prescription, modifying details like medication, dosage, and duration.
DOC002C	Delete Prescription	Allows the doctor to delete a prescription from a patient's record.
DOC003	Manage Patients	Provides the ability to view and delete patient records.
DOC003A	View Patient	Allows the doctor to view detailed information about a patient, including name, age, gender, email, and prescription history.
DOC003B	Delete Patient	Allows the doctor to delete patient records from the system, removing all associated data.

Table 4: Assistant Functionality

Requirement id	Name of requirements	Requirement description
ASST001	Assistant Login	Allows the assistant to securely log into the system using their credentials.
ASST002	Manage Appointments	Provides CRUD and view operations for managing appointments.
ASST002A	Add Appointment	Allows the assistant to schedule a new appointment, including details like patient name, doctor name, date, and time
ASST002B	Update Appointment	Allows the assistant to update an existing appointment, modifying details like date and time.
ASST002C	Delete Appointment	Allows the assistant to to delete an appointment from the system.
ASST002D	View Appointments	Allows the assistant to view a list of all appointments, including the ability to filter and search for specific records.
ASST003	Update Patient Email	Allows the assistant to update a patient's email address by inserting the patient ID and new email address by inserting the

5. Non-Functional Requirements

5.1. Performance

- Response Time: The system should respond to user queries within 2 seconds during peak hours.
- Scalability: The system should handle user traffic and operate properly under various conditions.

5.2. Reliability

- Fault Tolerance: The system should continue to operate in the event of a failure of one or more components.

5.3. Security

- Access Control: The system should only authorized personnel should have access to patient records
- Data Encryption: The system should guarantee encryption of all patient data both at rest and during processing

5.4. Usability

- User Interface: The system should feature an intuitive, user-friendly interface that simplifies navigation for healthcare professionals and enhances patient data accessibility.

5.5. Maintainability

- Documentation: Comprehensive system documentation should be maintained and updated with each new release.

5.6. Compliance

- Data Retention: The system should implement data retention policies that adhere to legal requirements for medical records.

6. Operational Scenarios

6.1. Admin Adds New Patient Record

Description:
This scenario involves an admin adding a new patient record to the system.
steps:

- The admin logs into the CMS using their credentials.
- They navigate to the patient management section.
- The admin selects the option to add a new patient record.
- They enter the patient's details such as name, age, gender, and contact information.
- After confirming the information, the admin submits the new patient record.

Expected Outcome:

The new patient record is successfully added to the system, and the details are stored in the database for future reference.

6.2. Doctor Updates Patient Prescription

Description:
In this scenario, a doctor updates an existing patient's prescription.
steps:

- The doctor logs into the CMS and accesses the patient management section.
- They search for the patient by name or ID and select the patient's record.
- The doctor views the patient's prescription details and decides to update it.
- They modify the prescription information, such as medication, dosage, and duration.
- After confirming the changes, the doctor saves the updated prescription.

Expected Outcome:
The patient's prescription is successfully updated in the system, reflecting the changes made by the doctor.

6.3. Assistant Schedules Patient Appointment

Description:
This scenario involves an assistant scheduling a new appointment for a patient.
steps:

- The assistant logs into the CMS and navigates to the appointment management section.
- They choose to schedule a new appointment and select the patient and doctor for the appointment.
- The assistant enters the date and time for the appointment and any additional notes.
- After confirming the details, the assistant submits the appointment request.

Expected Outcome:
The appointment is successfully scheduled, and both the patient and doctor are notified of the new appointment details.

6.4. Admin Deletes Doctor Record

Description:
In this scenario, an admin removes a doctor record from the system
steps:

- The admin accesses the doctor management section and views the list of doctors.
- They identify the doctor record that needs to be deleted.

- The admin selects the option to delete the doctor record.
- The system prompts for confirmation before permanently removing the doctor record.
- Upon confirmation, the admin confirms the deletion.

Expected Outcome:
The selected doctor record is successfully deleted from the system, and all associated data is removed from the database.

6.5. Doctor Views Patient Details

Description:
This scenario involves a doctor viewing detailed information about a specific patient.
steps:

- The doctor logs into the CMS and searches for the patient by name or ID.
- They select the patient's record to view detailed information.
- The doctor reviews the patient's medical history and current prescriptions.
- They may also view other appointments as well.
- Upon confirmation, the admin confirms the deletion.

Expected Outcome:
The doctor successfully accesses and reviews the detailed information about the selected patient, aiding in their treatment and care decisions.

7. Components used to create the CMS

- Authentication Component : Manages the login functionality for all users (admin, doctor, assistant). Ensures secure access and role-based authentication.
  - Admin Login
  - Doctor Login
  - Assistant Login
- Patient Management Component: Handles CRUD operations and viewing of patient records. Used by both admin and doctor roles for various patient-related tasks.
  - Add Patient
  - Update Patient
  - Delete Patient
  - View Patients
  - View Patient Details (for doctors)
- Doctor Management Component: Manages CRUD operations and viewing of doctor records. Used by the admin to maintain the list of doctors.
  - Add Doctor
  - Update Doctor
  - Delete Doctor
  - View Doctors
- Prescription Management Component: Allows doctors to manage patient prescriptions, including adding, editing, and deleting prescriptions.
  - Add Prescription
  - Edit Prescription
  - Delete Prescription
- Appointment Management Component: Handles CRUD operations and viewing of appointments. Used by the assistant to schedule and manage patient appointments.
  - Add Appointment
  - Update Appointment
  - Delete Appointment
  - View Appointments

8. Design Pattern Used

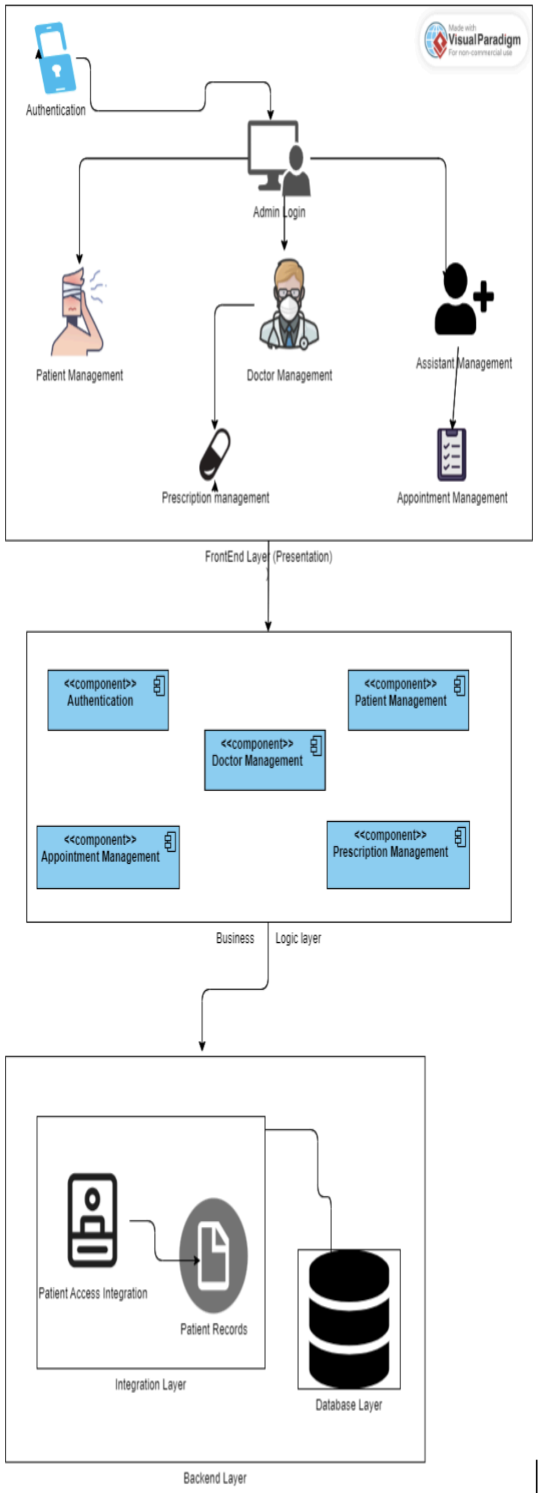
In this project, we utilize the Spring Framework to implement Dependency Injection (DI),following the IoC principle.Which allows us to manage dependencies between various components and services efficiently. DI promotes loose coupling, making the system more modular, testable, and maintainable. As well as we used Dependency Injection Principle by this principle its meant that the our code implemented relies on interfaces and not classes such as (IDBPatient).

9. Benefits we got from using the Spring framework Dependency Injection method

- 1. Modularity: Each component can be developed and tested independently.
- 2. Testability: Dependencies can be easily mocked or stubbed in unit tests, enabling effective isolation of components during testing.
- 3. Maintainability: Updates to a dependency require minimal changes to the classes using it, thanks to the loose coupling promoted by DI.

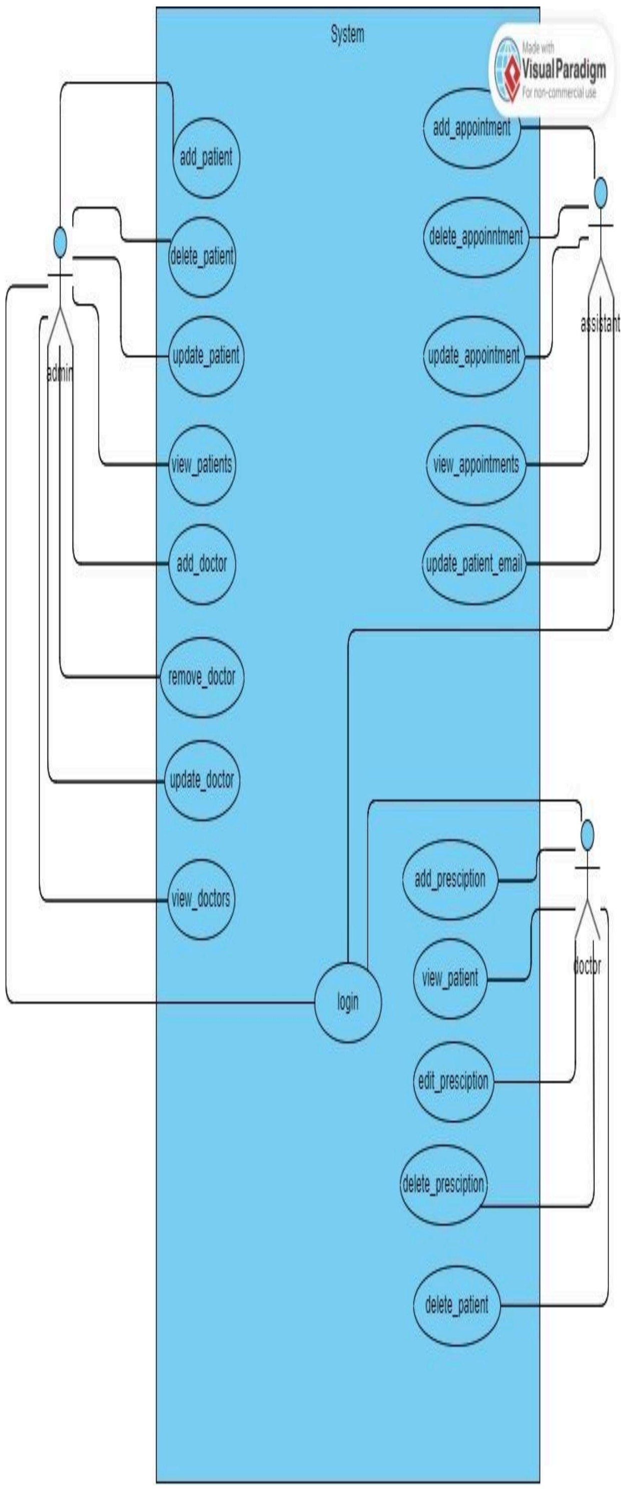
10. Diagrams

10.1 Design architecture

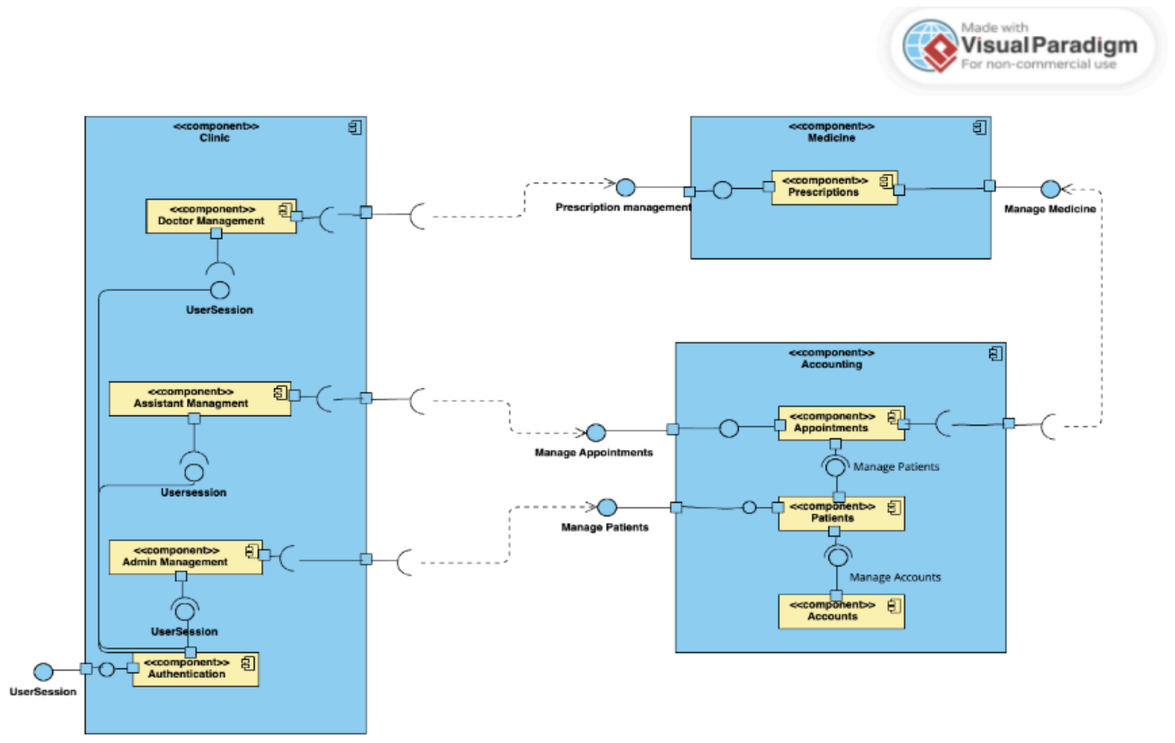


Made with Visual Paradigm For non-commercial use
Authentication
Admin Login
Patient Management
Doctor Management
Assistant Management
Prescription management
Appointment Management
FrontEnd Layer (Presentation)
<<component>> Authentication
<<component>> Patient Management
<<component>> Doctor Management
<<component>> Appointment Management
<<component>> Prescription Management
Business Logic layer
Patient Access Integration
Patient Records
Integration Layer
Database Layer
Backend Layer

10.2 Usecase diagram



10.3 Component Diagram


Made with Visual Paradigm For non-commercial use
<<component>> Clinic
<<component>> Doctor Management
UserSession
<<component>> Assistant Managment
Usersession
<<component>> Admin Management
UserSession
<<component>> Authentication
UserSession
Prescription management
<<component>> Medicine
<<component>> Prescriptions
Manage Medicine
<<component>> Accounting
Manage Appointments
<<component>> Appointments
Manage Patients
Manage Patients
<<component>> Patients
Manage Accounts
<<component>> Accounts