Sri Lanka Institute of Information Technology

Hospital Management System

Project Report

PAF Project 2020

Project ID -: S104.6

Submitted by:

1. Kotapolage D.G - IT17129336
2. Jayasuriya K.E - IT17179386
3. Gamage P.K.S.L.C - IT17142960
4. L.N Kodithuwakku - IT16038660

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Submitted to

Table of Content

Members' details

Student ID	Student Name	Contribution
IT17129336	Kotapolage D.G	Hospital Management ● Input hospital details ● View hospital details ● Update hospital details ● Delete hospital details
IT17179386	Jayasuriya K.E	Doctor Management ● Input doctor details. ● View doctors list. ● Update doctor details ● Delete doctor details
IT17142960	Gamage P.K.S.L.C	Appointments Register ● Input appointment details. ● View appointment list. ● Update appointment details ● Delete appointment details
IT16038660	L.N Kodithuwakku	User Management ● Input user/patient details ● View user/patient details ● Update user/patient details ● Delete user/patient details

Public VCS Repo(Click here to see repository)

SE Methodology

Incremental model

The incremental model is an evolution of the waterfall model. The incremental model prioritizes requirements of the system and then implements them into the groups. When we collaborate using github, then it is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance. We got basic functionality to the system early. most of the requirements are known up-front but are

expected to evolve over time. We used this project with new technology like Maven, Jersey and etc. we found basic software functionality before the implementation.

Advantages of incremental model with our system:

- More flexible and less costly to change scope and requirements.
- Risks of changing requirements are reduced.
- Easily to test and debug using postman during a small iteration.
- Generates working software quickly and early during the software life cycle.

Time schedule (Gantt chart)

Requirements

Stakeholder analysis

- Onion Diagram

Requirements analysis

- **Functional Requirements**
    1. **Patients**
        - Register to the system.
        - Can login to the system if already registered.
        - View hospital list.
        - View doctors list.
        - Update personal information.
        - Delete their own account.
        - Make an appointment.
        - Select payment method(postpaid or debit/credit card)
        - Add payment details(If credit/debit card selected)
        - View their payments
        - Delete their own appointment.
        - Logout from the system.
    2. **Staff**
        - Login to the system.
        - View hospital list.
        - View doctors list.
        - Delete appointments(outdated ones).
        - Logout from the system.
    3. **Admin**
        - Login to the system.
        - Add, update, view, delete hospitals.
        - Add, update, view, delete doctors.

- Remove users(If any fraud detected while payment)
- View Payments.
- Logout from the system.

- **Functional Requirements**
  1. **Security**
     - Passwords should be encrypted.
     - Can't use the same username.
     - Login details can be restored when forgotten.
     - Staff and users cannot insert, update and delete hospital and doctors data.
     - Staff cannot delete user accounts.
     - Session timeout after 15min if any users are inactive that period.
  2. **Performance**
     - Response time should be between 1 to 3 seconds.
     - System needs to support 5000 users at once.
     - Payment validation time should be at least 5 seconds.
     - After first logging some of appointment details should be auto-filled according to user information
  3. **Reliability**
     - System should be available for all the time.
     - System should be able to be used multiple times.

  4. **Correctness**
     - System should be bug free.

- **Technical requirements**
  1. **Developing End**
     - JDK 1.8
     - Eclipse Oxygen Release (4.7.0).
     - Tomcat 9.0
     - Java - JAX-RS(Jersey-Restful).
     - MySQL server.

  2. **Client End**
     - OS - Windows 7/8/8.1/10
     - Postman
     - JRE 1.8
     - MySQL server

  3. **Hardware**
     - Wifi-Router
     - Printer
     - PC/ Laptop -: Minimum - Core i3 processor, 4GB ram, 500GB HDD

Maximum - Core i5 processor, 8GB ram, 1TB HDD

Requirements modelling

- Use case diagram

System's overall design

Overall architecture

Overall DB design

- ER Diagram

Activity Diagrams

- Register and login to the system.
- User Payments.
- Doctors Management.
- Hospital Management.

Other Design Diagrams

- Class Diagram

Individual sections

Doctors Management -: IT17179386 (Jayasuriya K.E)

Service design

Design Rationale

Introduction

The Doctors Management section is always handled by the system admin. In this section, it's main task is, add new doctors to the system and view the doctors list on the main page. When a user makes an appointment he/she must have to select the doctor's name section using the drop down list in the appointment form. Therefore my main task is to send doctors names to the appointment form. And also System admin can update any of the doctors details when needed. On the other hand he can delete doctors from the list who are leaving the hospital permanently. And also any other who are using this site can view the doctors' list on the main page and they have read only permission for that.

Problems

- We are trying to implement an expanded full system. Most of the time we have to install additional software or libraries. When we do that we have to search for addons, plugins, jar files and etc. When we do that manually it takes a lot of time.
- Data must be stored in a secure way. Storing data directly to local databases did not provide much security. And also we need to prevent system overloading.
- If we use client server architecture we will face another problem: how to communicate between client and server.
- There can be a lot of requests coming from the same user. And also there can be many users. Then we need a way to keep our system performance steady.
- And also developers working with their individual projects. Then we need to connect these projects individually but they all must work as a single project.

For solving above all the problems I have chosen Restful API. In this situation Restful gives me a lot of advantages.

How Restful API solve my problems?

- Restful always takes advantage of HTTP. Therefore I don't want to bother about using frameworks and installing additional software or libraries. Restful, always take care of it by single handed.
- Most secure way to store client data in a local database is using Client-Server Architecture. It means there are two independent sides. One is Client. Other one is Server. Server can handle one or many users at once. Server side main tasks are application access, storage, file sharing etc. When multiple clients request resources or send processes, the server side knows which client sends which processes and which clients request which resources and send all those to relevant users. Restful API consists of this client-server architecture and it can communicate through the client and server.
- When users request resources or send processes to the server, it will take some few seconds. But if we are able to store the memory of client workflow we can make it fast when the user uses this system second time to all the time. Caching is the best way to store the memory of someone's workflow. This method always increases the system performance. This feature is also provided by Restful API.
- All of the developers in this project work on their own projects. Therefore when integrating the system we need to connect these individual projects and work as a single project. We can use microservices to do that. Restful will help us to implement this microservices easily.
- Restful always supports most types of platforms and languages. Therefore it's usability is at a high level.

Other reasons -: Why I choose Restful over other APIs

- It is easy to understand and easy to adapt without prior knowledge.
- It supports lot of files formats.(ex -: XML, Json)
- Provide better performance when caching.
- Most major sites use that.(ex -: Google, Amazon, Yahoo, Ebay)

Internal logic

- Doctors Management
- Class Diagram

Database for the service

- ER Diagram

Any other diagrams

- UseCase Diagram - Doctor Management System

Service development and testing

Tools used

- **Dependency management tool - Maven**
  - Very easy to use because of minimal configuration required.
  - Import internal and external libraries automatically and maintain it.
  - When I use framework as a dependency maven download jar files automatically and debug some library errors.(ex -: UnSupportedClassVersionError and NoClassDefFoundError)
  - It is easy to download maven to my eclipse. Only thing to do is download the M2E Eclipse plug-in eclipse marketplace.
- **Testing tool - Postman**
  - Restful API most comfortable with Postman than other tools.
  - It is easy to understand. It has great user interfaces and There are a lot of tutorials on the internet.
  - Support for Multiple formats.(ex -: Json, xml) Therefore it is easy to write codes.
  - We can easily move test cases from one system to another system and also from one environment to another
- **Framework - Jersey**
  - Open source framework
  - It can be used to develop RESTful Web Services and provides support for JAX-RS APIs.
  - Provide best tooling support with eclipse IDE.
- **Languages - Java**
  - Platform independent -: Can write once and use or reuse the code anywhere.
  - Highly secure.

- Most commonly used languages. Many applications created using java.
- Easy to understand.
- It is an Object Oriented Programming language.
- It has a lot of community support.(Ex -: stackoverflow, github, etc.)

- **IDE - Eclipse**
  - Cost less time and effort.
  - Open source.
  - Has one of the best UI. Therefore navigation is easier.
  - Has auto completion features.
  - .Error debugging is easy. you can easily navigate to the error line.
  - Support many languages, frameworks, APIs, plugins, Libraries, etc.

Testing methodologies and Results

- **Functionality Testing**
  - All form validations are working.
  - Forms have better readability and optimally formated.
- **Interface Testing**
  - Application -: client requests correctly sended to the database and display required details on the client side.
  - Web Server -: web server handled application requests without any service rejection.
  - Database Server -: queries which are sent to the database give expected results.
- **Database Testing**
  - All CRUD queries executed without any error or warning and given expected results.
  - Response time when running queries is at the best tune.
- **Performance Testing**
  - Cache enabled because of using API. That leads to reduced load times.
  - There is no visible Crash or bug, occurred.
- **Security Testing**
  - If a user inactive for given time sessions will be automatically killed.

Assumptions

- **Crowd Testing** -: This is a newly implemented project. Therefore this is not released to a crowd for testing. Therefore we don't know the performance level when using this system by a large crowd. But I believed this application was created using Client-Server Architecture. Therefore There won't be any low performance. Therefore Performance must be the same against any number of crowds.

- I haven't implemented a frontend but backend takes client requests and performs CRUD operations as I expected. Therefore I assume if I have implemented the frontend, the same result can be expected.

- We have implemented our system using Jersey Restful web services. There are limited resources in the internet to create microservices using that technology. But if we used spring boot there are a lot of resources on the internet.

- We have used a single Database with multiple tables. This may create some traffic when data exchanging. If we use multiple databases I think we will be able to reduce our data traffic.

- We didn't implement microservices when we were informed earlier. This occurred because of over poor time management. If we implemented microservices, our system maintenance, performance, flexibility, etc and a lot of features would be improved than the current system.

Hospital Management -: IT17129336 (Kotapolage D.G)

Service design

Design Rationale

Introduction

The Hospital Management part is always handled by the system administrator.in this section,it's major task is add new hospitals to the system and view the hospital list with details on the main page.when registered user makes an appointment they should be select the hospital name where among registered hospitals.therefore my one of major task is to send hospital names to the appointment form and doctor registration form.As well as system administrator can update hospital details anytime.other major task is delete some hospitals.then admin decide the some doctors are register using it hospital ID,then owners cannot delete their hospital permanently.if not,then hospital owners can remove their hospital without issues.And who are using this site,their can view hospitals details and decide to channel location.as well as any others can view hospitals details with read only permission.

Problems

- We hope to implement a full hospital management system with all the major functions.we searched a lot of articles and youtube videos related to this topic.it takes a lot of time.
- Before starting this project, we installed some of softwares, libraries,jar files, etc. as well as updating current softwares and tools.Then it took a lot of time.
- When during the implementation, we changes some software versions.otherwise projects is not run properly.(change tomcat 8.0 to 9.0 and change jdk 1.6 to 1.8)
- In client-server architecture,a major problem is how to handle multiple requests without server failure at simultaneously.

- This hospital management system should be worked with 24 hours without any server failure.because this is one of compulsory service.
- As well as we try to provide this application for many users.therefore we found what is most suitable for our purpose.
- As well as developers come up with their individual projects.but it should be integrated and work as a single project .
- Finally I found the most suitable solution to solve these all problems.it is Restful API and came up with a lot of benefits to our project.

Solving problems using Restful API

- Restful API gets all advantages of HTTP(get,post,put,delete).I want routes to be designed with HTTP recommendations called Restful.as a result of this, I don't want any other tools,frameworks or software.

- In client-server architecture, client data are stored in local database.it is secure store method within client-server architecture and having two independent sides called client and server.but this case, client side has less process and server side has huge process.major task of server side are give access for some resources and processes. When multiple clients request resources or send processes, the server side knows which client sends which processes and which clients request which resources and send all those to relevant users. Restful API uses this client-server architecture and it can communicate through the client and server.

- Restful API is independent of the type of platforms and languages
- As well as give freedom changing or testing any environment during development .
- Rest API allows various types of data formats.(JSON,XML)

- When we use microservices for this project,we don’t want to integrate projects as a single project.We can implement projects as individually and using microservices we can work multiple projects as single project.Restful helpful for implementing microservices.

- Restful API is designed to encourage the storage of catchable data.
- With the initial URL the client does not require route information.
- REST provides best performance and catching for information that is not altered.(faster and use less bandwidth)

Internal logic

- Activity diagram
- Class Diagram

Database for the service

- ER Diagram

Any other diagrams

- Use case diagram - hospital manage system

Service development and testing

Tools used

- **Dependency management tool - Maven**
  - Very easy to handle this. because of minimal configuration required.
  - Import internal and external libraries automatically and maintain it.
  - Manage project dependencies and update the classpath of the project dependencies in the Eclipse IDE.
  - Maven is come up best performance with Eclipse
  - As well as it provides an editor for tha pom.xml maven configuration file and edits the xml data directly.
  - Maven downloads into eclipse and sets the environmental variable.after we can run maven projects in PCs.
  - When I use Maven with my project,download jar files automatically and debug some library errors. (ex -: UnSupportedClassVersionError and NoClassDefFoundError)
  - Easy to manage multi module projects with Maven.
- **Testing tool – Postman**
  - Sending request---- > running In few seconds------>inspect response
  - Restful API most comfortable with Postman than other tools.
  - Postman provides some authentication protocols(0AUTH 2.0,AWS signature, Hawk authentication)
  - Debugging - postman console helps to check what data have been retrieved making it process easy to debug tests.
  - Collection of environments can be imported or exported making easy to shared files.(collaboration)
  - Support for Multiple formats.(ex -: Json, xml) Therefore it is easy to request and pass data.
  - Backend developers can easily check API without frontend app.
- **Framework - Jersey**
  - Open source framework
  - It can be used to develop RESTful Web Services and provides support for JAX-RS APIs.
  - Provide best tooling support with eclipse IDE.
- **Languages - Java**
  - Platform independent -: Can write once and use or reuse the code anywhere.
  - Highly secure.

- Most commonly used languages.
- Used for many applications(desktop,web applications etc).
- Easy to understand and implement
- It is an Object Oriented Programming language.
- It has a lot of community support.(Ex -: stackoverflow, github, etc.)

- **IDE - Eclipse**
  - Cost less time and effort.
  - Open source.
  - Industrial level of development.
  - Easy to framework integration like things.
  - All the files can be viewed and managed at the same screen.
  - Error debugging is easy. you can easily navigate to the error line.
  - Support many languages, frameworks, APIs, plugins, Libraries, etc.

Testing methodology and Results

- **Functional Testing**
  - Aim to well prepared input and output data.
  - Also helps to define verification approaches and improve readability.
- **Interface Testing**
  - Application -: client requests correctly sended(POST) to the database and display required details on the client side.
  - Web Server -: web server handled application requests without any server error.
  - Database Server -:we got expected results without any bugs.
- **Database Testing**
  - All CRUD queries executed without any bugs.
  - Response time when running queries is at the best tune.
- **Performance Testing**
  - Cache enabled because of using API. That leads to reduced load times.
  - There is no visible Crash or bug, occurred.
  - No take any long time for a response.it gives results quickly.
- **Security Testing**
  - When the user session expires,then automatically logout.

Assumptions

- I didn't implement the interface.but I test clients using postman.therefore I assume when postman gives expected results the same as the frontend result.and assume I have implemented frontend.

- I assume this application testing using an audience,we found some bugs and we can correct those bugs and develop a system with minimum bugs and more user friendly.

- When we use ASP.NET api I think we can use the routeprefix attribute and then control class implement very easily.

- I think restful API implementations easily use spring boot like frameworks.then we can implement this project in less time period.

- We didn't implement microservices when we were informed earlier. This occurred because of over poor time management. If we implemented microservices, our system maintenance, performance, flexibility, etc and a lot of features would be improved than the current system.

Appointments Register - IT17142960 (Gamage P.K.S.L.C)

Service design

Design Rationale

Introduction

The user logs in to the system using a username and password. Then the user can enter his location and preferred date for the appointment. System will show the hospital list according to the user location. Then the user can select any hospital and doctor category(Dentist, ENT etc.). After that system will show a list of doctors who visit selected hospital and in that category. Then user can select their doctor and payment type. System will issue a token number for the appointment. All the appointment details are saved in a separate table in the database. Users can see their all appointment details from the user dashboard. If a user needs to change something, he can update the appointment details. Even users can cancel their previously booked appointments from the dashboard.

API for get all the appointment details in the database (GET Request)

Resource : appointments.

Request: GET Hospital/HospitalService/appointments

Response : Html table with all the appointment details

API for insert a new appointment to the database (POST Request)

Resource: appointments

Request: POST /Hospital/HospitalService/appointments

Media: Form data - URL encoded

Data:

“username” = “<username>”
“doctor_name” = “<doctor_name>”
“hospital_name” = “< hospital_name>”
“date” = “< date>”
“payment_type” = “< payment_type>”

Response: String status message

API for update an appointment which is existing in database (PUT Request)

Resource: appointments
Request: PUT Hospital/HospitalService/appointments
Media: Application JSON

Data:

```
{
        "Token_number" : "<token_number>",
         "Username" : "<username>",
         "Doctor_name" : "<doctor_name>",
         "Hospital_name" : "<hospital_name>",
         "Date" : "<date>",
         "Payment_type" : "<payment_type>"
}
```

Response: String status message

API for delete an appointment which is existing in database (DELETE Request)

Resource: appointments
Request: DELETE Hospital/HospitalService/appointments
Media: Application XML

Data:

```
<appData>
        <token_number>”token_number”</token_number>
</appData>
```

Response: String status message

Internal logic

- Activity Diagram
- Class Diagram

Database for the service

- ER Diagram

Any other diagrams

- Use Case Diagram - Appointments & Payments

Service development and testing

Tools used

- **Dependency management tool - Maven**
  - Very easy to use because of minimal configuration required.
  - Import internal and external libraries automatically and maintain it.
  - When I use framework as a dependency maven download jar files automatically and debug some library errors.(ex -: UnSupportedClassVersionError and NoClassDefFoundError)
  - It is easy to download maven to my eclipse. Only thing to do is download the M2E Eclipse plug-in eclipse marketplace.
- **Testing tool - Postman**
  - Restful API most comfortable with Postman than other tools.
  - It is easy to understand. It has great user interfaces and There are a lot of tutorials on the internet.
  - Support for Multiple formats.(ex -: Json, xml) Therefore it is easy to write codes.
  - We can easily move test cases from one system to another system and also from one environment to another
- **Framework - Jersey**
  - Open source framework
  - It can be used to develop RESTful Web Services and provides support for JAX-RS APIs.
  - Provide best tooling support with eclipse IDE.
- **Languages - Java**
  - Platform independent -: Can write once and use or reuse the code anywhere.
  - Highly secure.
  - Most commonly used languages. Many applications created using java.
  - Easy to understand.
  - It is an Object Oriented Programming language.
  - It has a lot of community support.(Ex -: stackoverflow, github, etc.)
- **IDE - Eclipse**
  - Cost less time and effort.
  - Open source.
  - Has one of the best UI. Therefore navigation is easier.
  - Has auto completion features.
  - .Error debugging is easy. you can easily navigate to the error line.

- Support many languages, frameworks, APIs, plugins, Libraries, etc.

Testing methodology and Results

- **Functionality Testing**
  - All form validations are working.
  - Forms have better readability and optimally formated.
- **Interface Testing**
  - Application -: client requests correctly sended to the database and display required details on the client side.
  - Web Server -: web server handled application requests without any service rejection.
  - Database Server -: queries which are sent to the database give expected results.
- **Database Testing**
  - All CRUD queries executed without any error or warning and given expected results.
  - Response time when running queries is at the best tune.
- **Performance Testing**
  - Cache enabled because of using API. That leads to reduced load times.

Test ID	Test Description / Test Steps	Test input(s)	Expected Output(s)	Actual Output(s)	Result (Pass/Fail)
01	Get all the appointment details	-	Display html table with all the details	Display html table with all the details	Pass
02	Create new appointment	[{"key":"username","value":"Chathurange"}, {"key":"doctor_name","value":Dr.silva"}, {"key":"hospital_name","value":"Asiri"}, {"key":"date","value":"020-04-27"}, {"key":"payment_type","value":"cash"}]	Display Message as "*Successfully inserted*"	Display Message as "*Successfully inserted*"	Pass

03	Update appointment details	{
		"Token_number":"3",
		"username":"Gamage",
		"doctor_name":"Dr. Shashi",
		"hospital_name":"Suwasewana",
		"date":"2020-05-03",
		"payment_type":"online"
		}
			Display Message as "*Successfully updated*"	Display Message as "*Successfully updated*"	Pass

04	Remove appointment	<appData> <token_number>2</token_number> </appData>	Display Message as "*Successfully deleted"*	Display Message as "*Successfully deleted"*	Pass

Users Management - IT16038660 (L.N. Kodithuwakku)

Service design

Design Rationale

Introduction

First if any person wants to get an appointment and meet a doctor they should register to the hospital system. That means they must create an account for them. For the registration they should provide their details to the system. Those areName, age, address, email, password, phone number etc. Then the user will be uniquely identified by his/her email. Also there will be a patient id for every user which is unique and auto incremented by the database. After creating the account they can login to the system using email and password. Then the user can access the other functionality of the hospital system. Then the user can select any hospital and doctor in the appointment section.

Internal logic -: User Management, Class Diagram

Database for the service -: ER Diagram

Service development and testing

Tools used -: Dependency management tool - Maven, **IDE** - Eclipse, **Languages** - Java, **Server** - Tomcat, **Database** - MySQL

Testing methodology and Results -: Testing tool – Postman

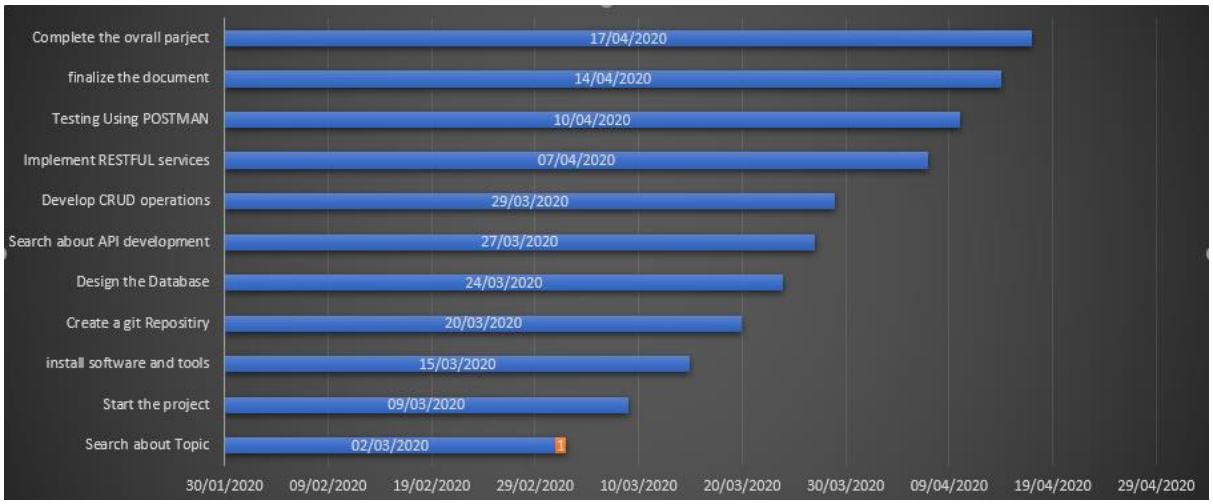
Systems integration

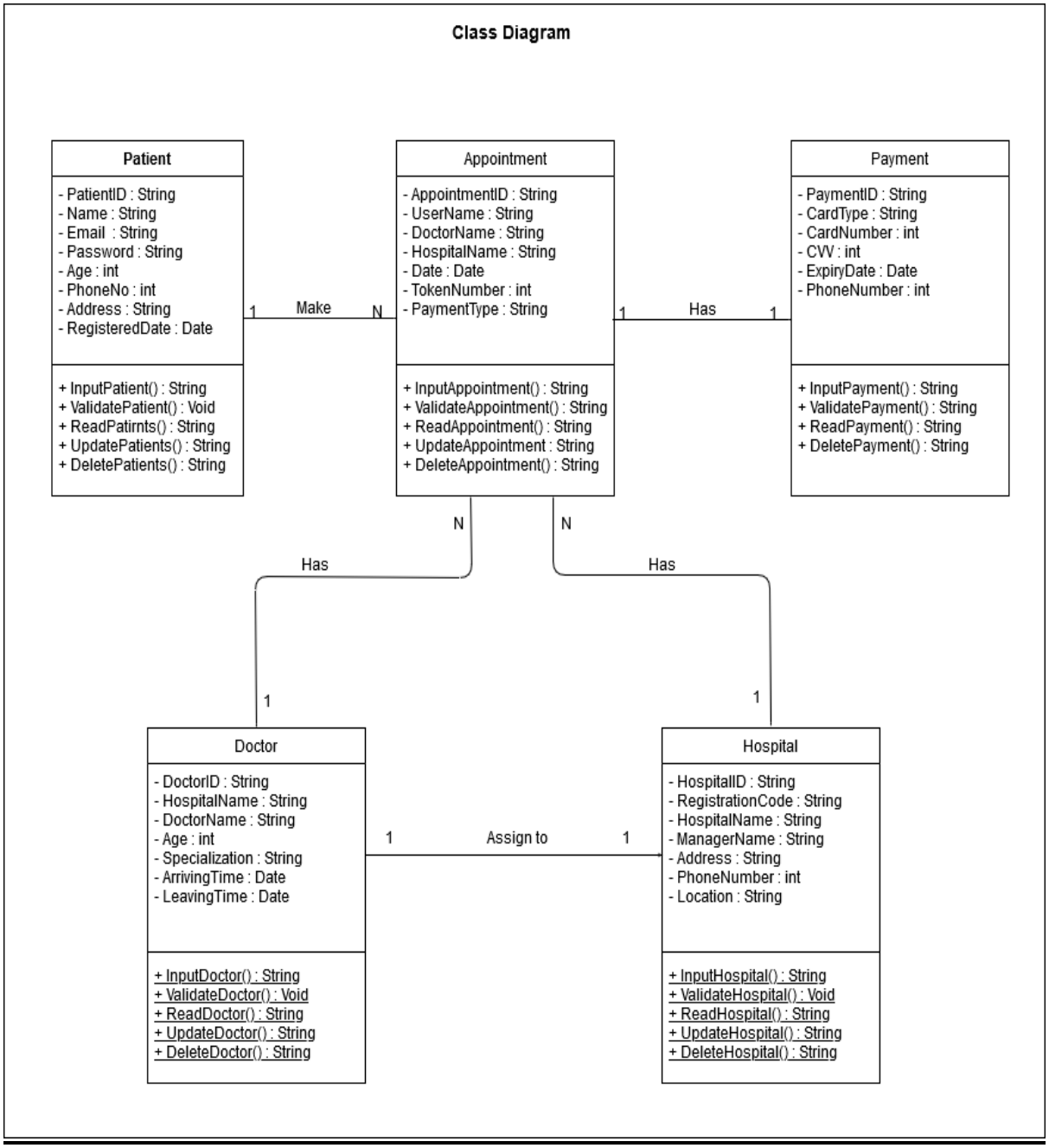
- For System integration we have used Github(https://github.com/).
- First of all, one of our member has created a public Git repository and given access to three other members.
- Then we created our own projects in our eclipse and pushed that project to the github.
- By using pull and commit & push methods we have implemented our project.
- After the final commit, every member pulls the full project to their IDE and check is it working in all Laps.
- In every commit all members gave meaningful commit messages.
- We clone the project using the git repositories option which exists in our Eclipse IDE.
- And also the final document was created using google drive with a shared option enabled. That option provides us to edit the document as a group at the same time.

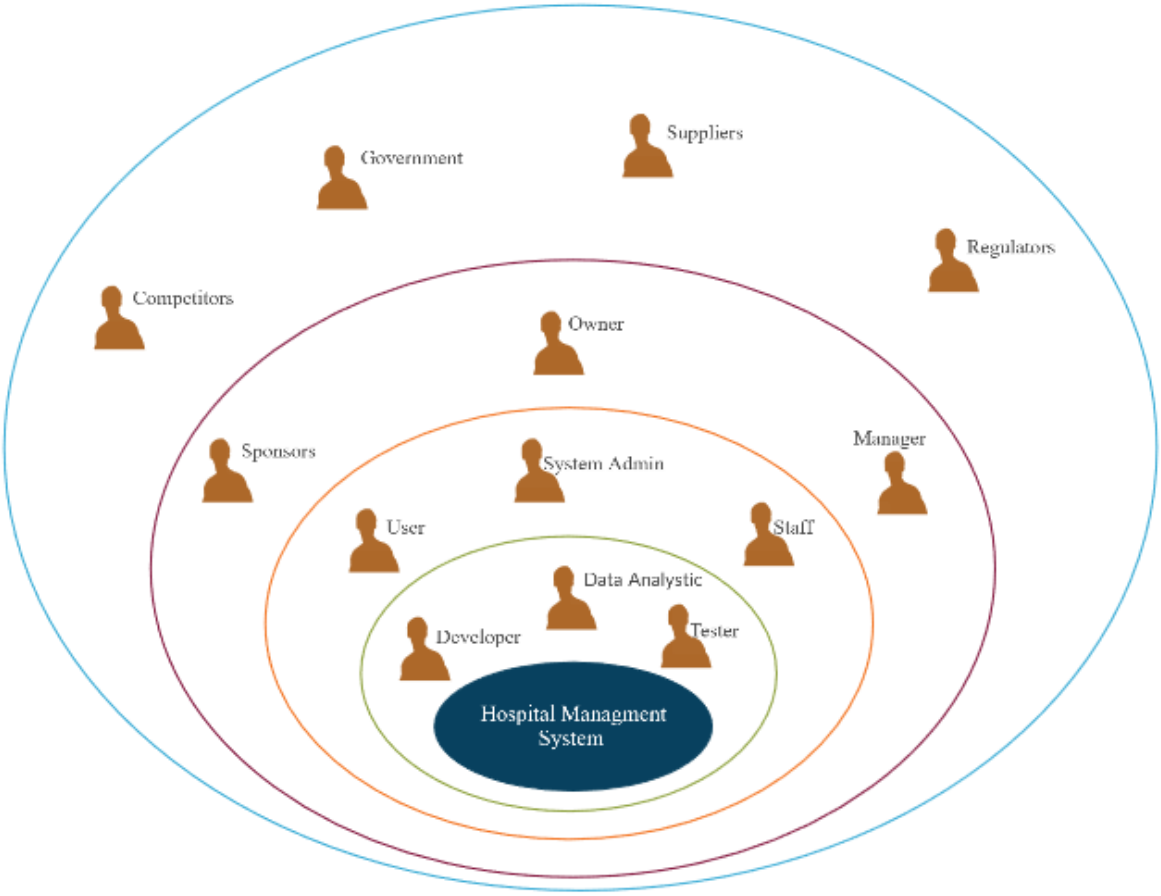
References

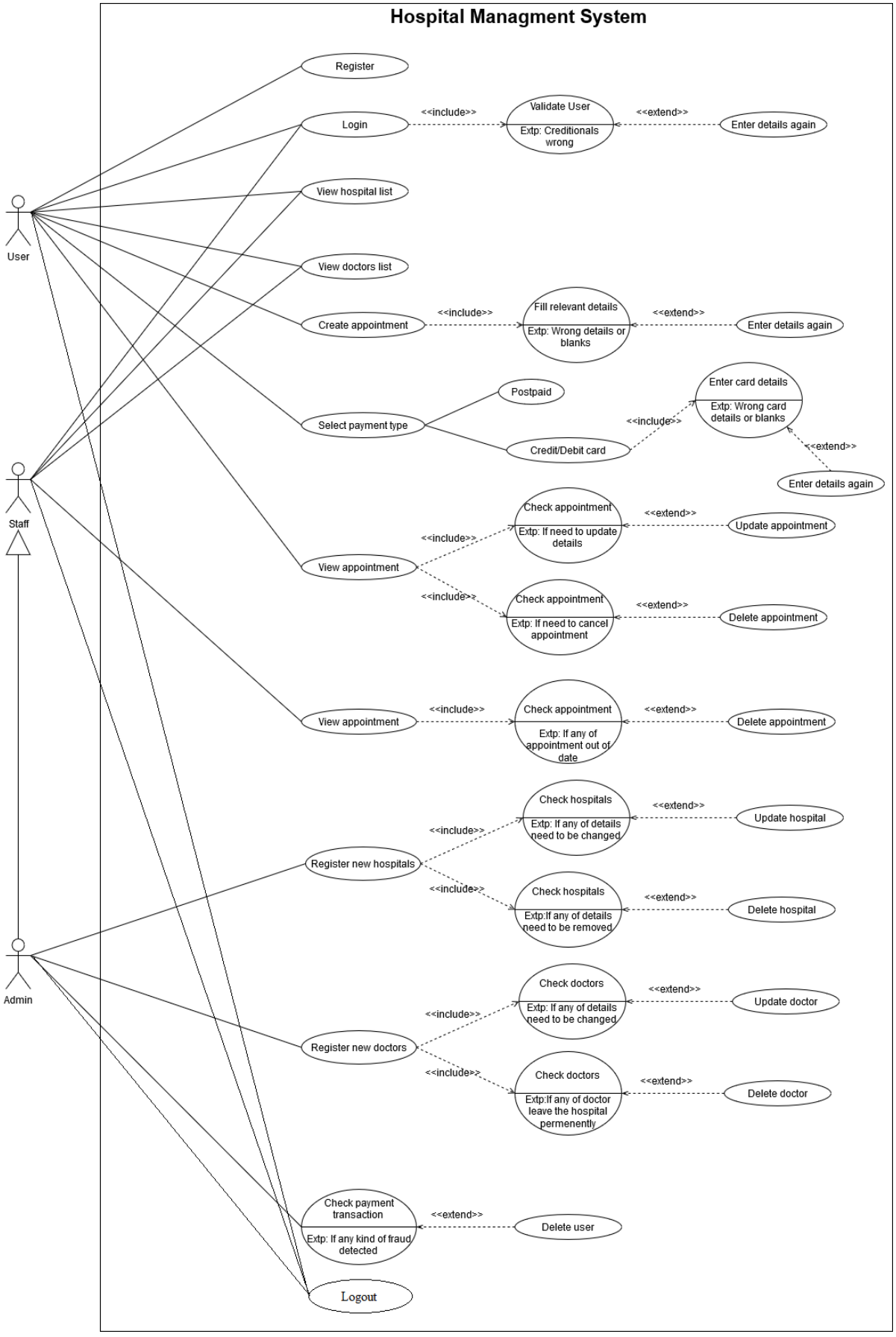
- **Information Gathering**
  - **What are Dependency Managers? - Prodsters - Medium**
  - **10 reasons why we chose Maven over Gradle**
  - **Benefits of Maven for java developers - lenin stalin - Medium**
  - **API Testing using Postman - Aubergine Solutions - Medium**
  - **What is Postman, and Why Should I Use It? | DigitalCrafts**
  - **rest - Why use JAX-RS / Jersey? - Stack Overflow**
  - **9 Best Reasons To Choose Java For Web Development | Xicom**
  - **(2) What are the advantages of Eclipse IDE? - Quora**
  - **Web Application Testing: 8 Step Guide to Website Testing**
  - **Functional vs Non-functional Requirements: List & Examples**
  - **What are Technical Requirements in Project Management?**
  - **Systems architecture - Wikipedia**
  - **RESTFul APIs! - Vikas Shivpuriya - Medium**
  - **How to Write a Design Rationale - Jennifer Cederstam**
  - **Uniform Interface - an overview | ScienceDirect Topics**
  - **CourseWeb | Sri Lanka Institute of Information Technology**
- **Diagram designing**
  - https://www.draw.io/
  - https://app.creately.com/

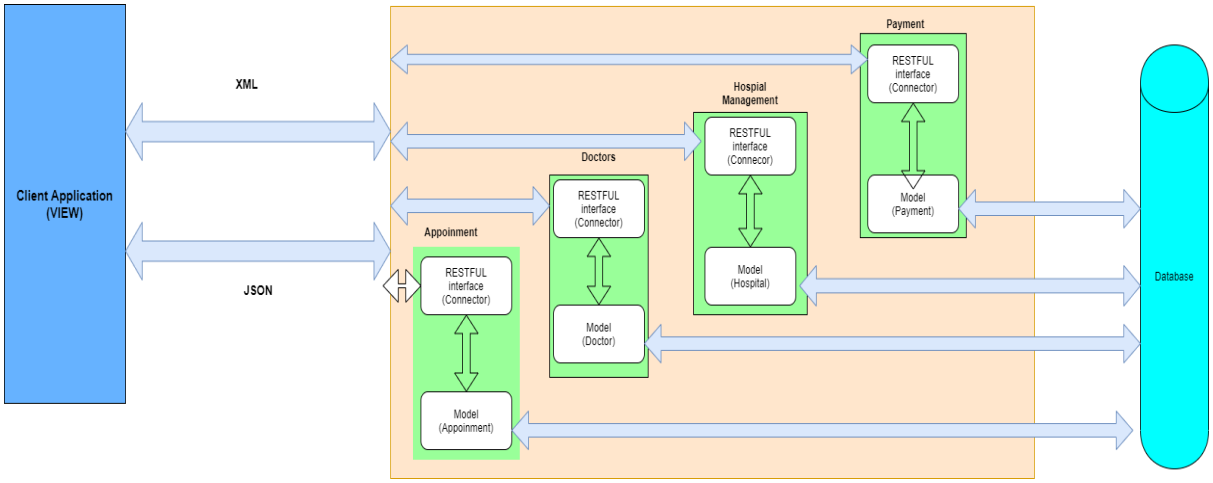
Appendix

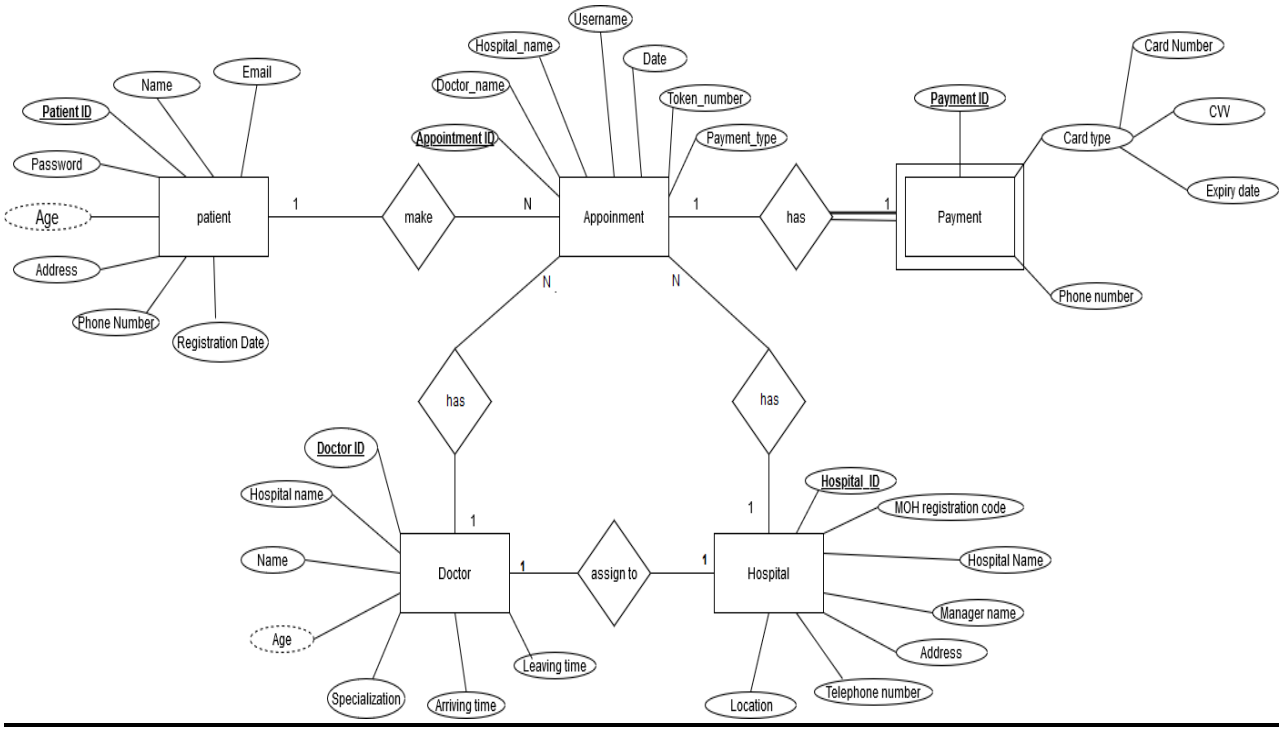

Complete the ovrall parject
finalize the document
Testing Using POSTMAN
Implement RESTFUL services
Develop CRUD operations
Search about API development
Design the Database
Create a git Repositiry
install software and tools
Start the project
Search about Topic
17/04/2020
14/04/2020
10/04/2020
07/04/2020
29/03/2020
27/03/2020
24/03/2020
20/03/2020
15/03/2020
09/03/2020
02/03/2020
30/01/2020
09/02/2020
19/02/2020
29/02/2020
10/03/2020
20/03/2020
30/03/2020
09/04/2020
19/04/2020
29/04/2020

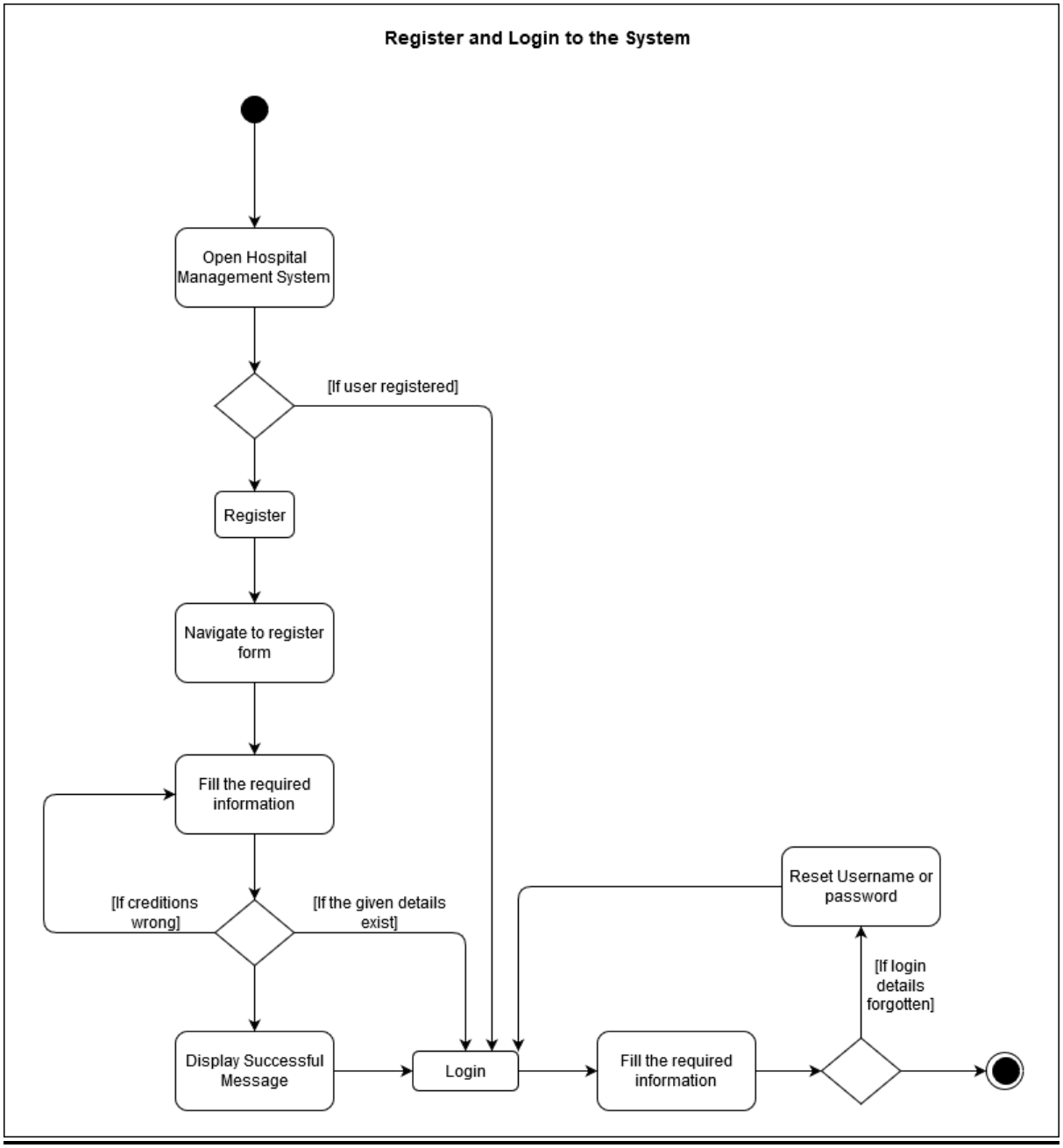

Class Diagram
Patient
- PatientID : String
- Name : String
- Email : String
- Password : String
- Age : int
- PhoneNo : int
- Address : String
- RegisteredDate : Date
+ InputPatient() : String
+ ValidatePatient() : Void
+ ReadPatirnts() : String
+ UpdatePatients() : String
+ DeletePatients() : String
1
Make
N
Appointment
- AppointmentID : String
- UserName : String
- DoctorName : String
- HospitalName : String
- Date : Date
- TokenNumber : int
- PaymentType : String
+ InputAppointment() : String
+ ValidateAppointment() : String
+ ReadAppointment() : String
+ UpdateAppointment : String
+ DeleteAppointment() : String
1
Has
1
Payment
- PaymentID : String
- CardType : String
- CardNumber : int
- CVV : int
- ExpiryDate : Date
- PhoneNumber : int
+ InputPayment() : String
+ ValidatePayment() : String
+ ReadPayment() : String
+ DeletePayment() : String
N
Has
1
N
Has
1
Doctor
- DoctorID : String
- HospitalName : String
- DoctorName : String
- Age : int
- Specialization : String
- ArrivingTime : Date
- LeavingTime : Date
+ InputDoctor() : String
+ ValidateDoctor() : Void
+ ReadDoctor() : String
+ UpdateDoctor() : String
+ DeleteDoctor() : String
1
Assign to
1
Hospital
- HospitalID : String
- RegistrationCode : String
- HospitalName : String
- ManagerName : String
- Address : String
- PhoneNumber : int
- Location : String
+ InputHospital() : String
+ ValidateHospital() : Void
+ ReadHospital() : String
+ UpdateHospital() : String
+ DeleteHospital() : String

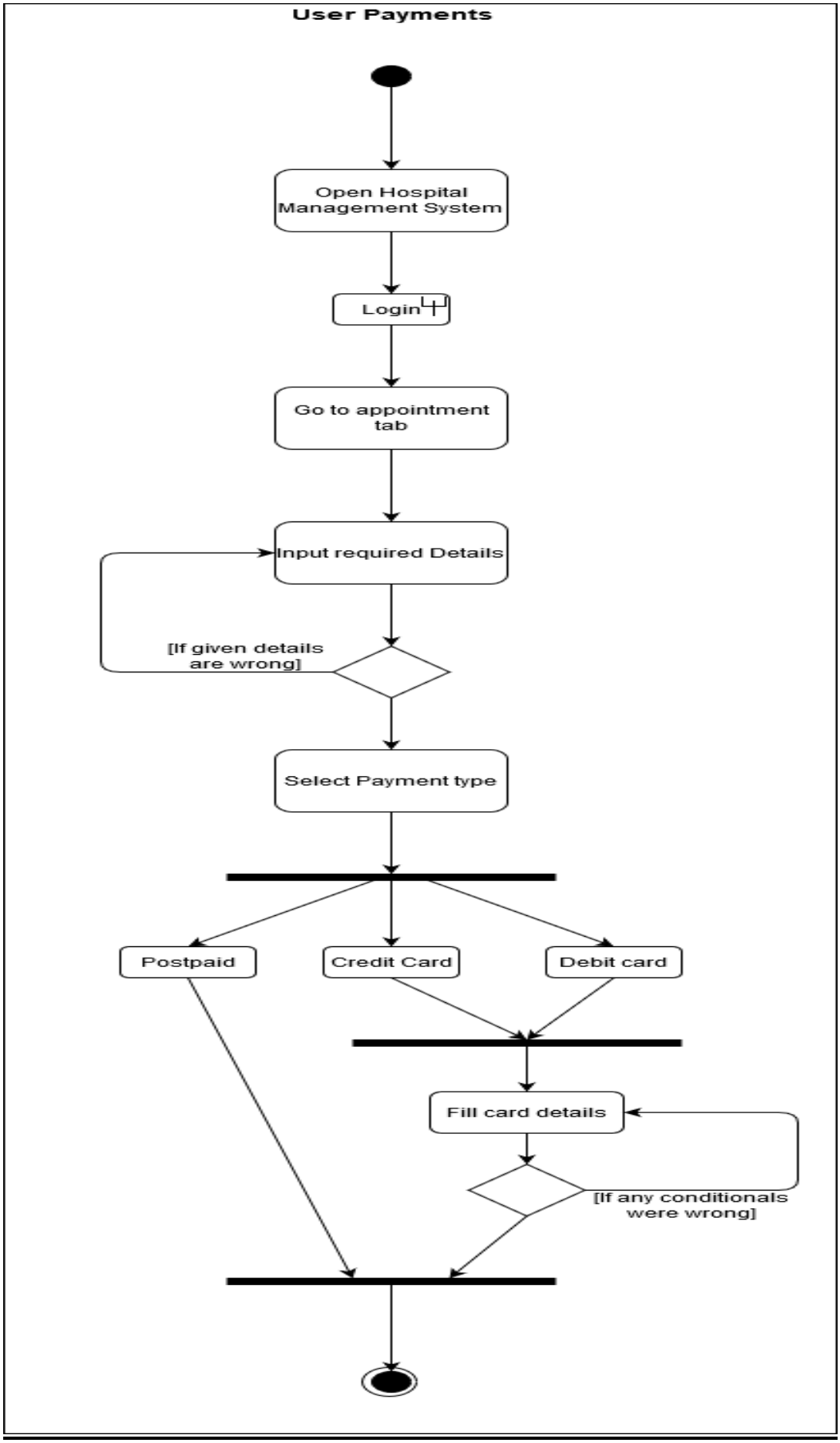

Government
Suppliers
Regulators
Competitors
Owner
Sponsors
System Admin
Manager
User
Staff
Data Analystic
Developer
Tester
Hospital Managment System

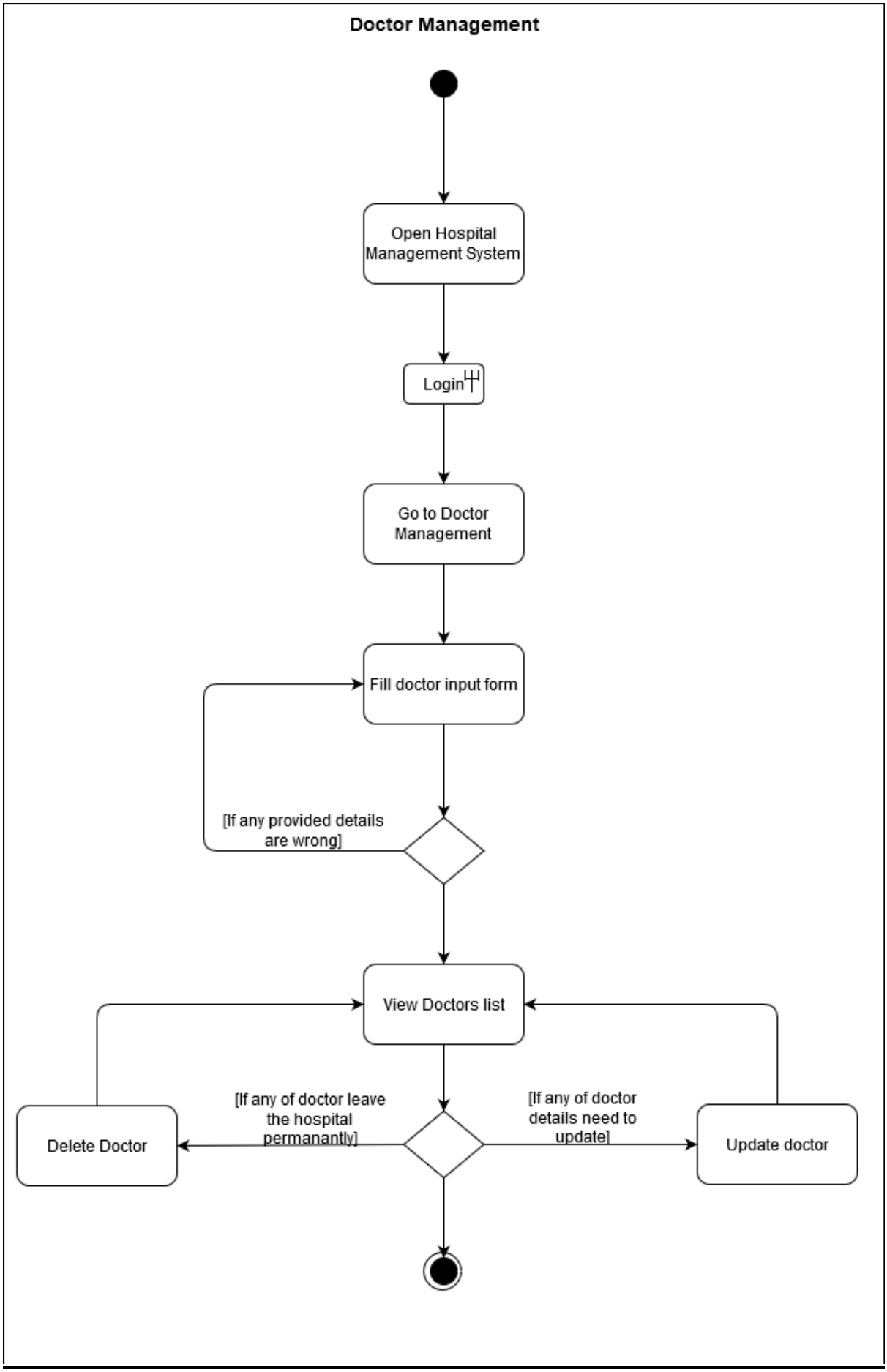

Hospital Managment System
User
Staff
Admin
Register
Login
Validate User
Extp: Creditionals wrong
Enter details again
View hospital list
View doctors list
Create appointment
Fill relevant details
Extp: Wrong details or blanks
Enter details again
Select payment type
Postpaid
Credit/Debit card
Enter card details
Extp: Wrong card details or blanks
Enter details again
View appointment
Check appointment
Extp: If need to update details
Update appointment
Check appointment
Extp: If need to cancel appointment
Delete appointment
View appointment
Check appointment
Extp: If any of appointment out of date
Delete appointment
Register new hospitals
Check hospitals
Extp: If any of details need to be changed
Update hospital
Check hospitals
Extp:If any of details need to be removed
Delete hospital
Register new doctors
Check doctors
Extp: If any of details need to be changed
Update doctor
Check doctors
Extp:If any of doctor leave the hospital permenently
Delete doctor
Check payment transaction
Extp: If any kind of fraud detected
Delete user
Logout
<<include>>
<<extend>>

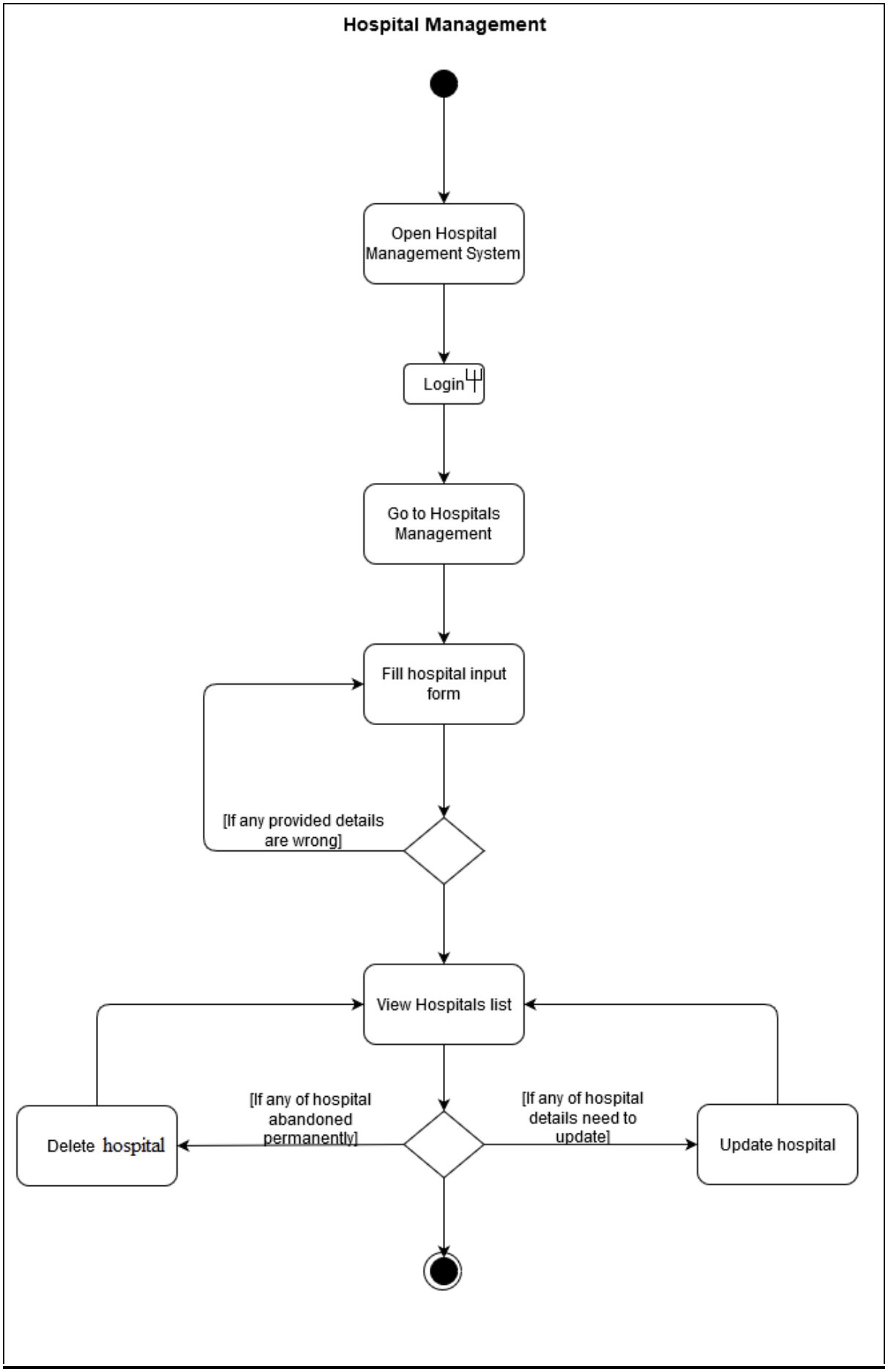

Client Application (VIEW)
XML
JSON
Payment
RESTFUL interface (Connector)
Model (Payment)
Hospial Management
RESTFUL interface (Connecor)
Model (Hospital)
Doctors
RESTFUL interface (Connector)
Model (Doctor)
Appoinment
RESTFUL interface (Connector)
Model (Appoinment)
Database

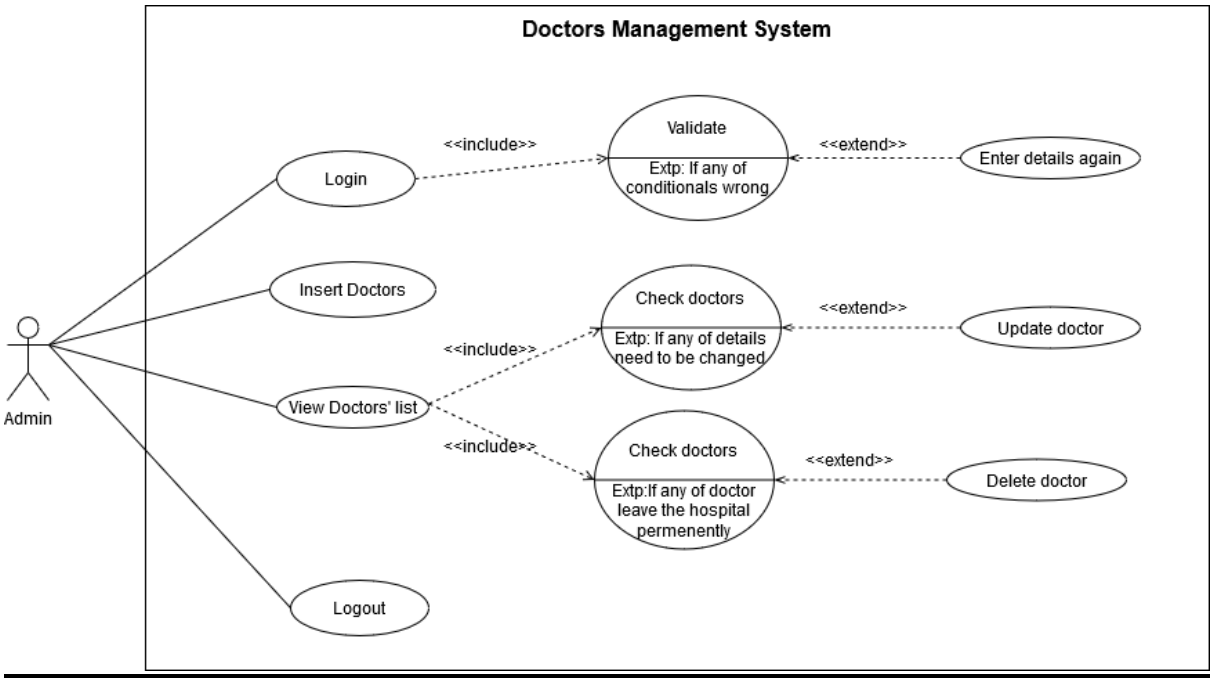

Username
Hospital_name
Date
Doctor_name
Token_number
Appointment ID
Payment_type
Name
Email
Patient ID
Password
Age
Address
Phone Number
Registration Date
patient
1
make
N
Appoinment
1
has
1
Payment
Payment ID
Card type
Card Number
CVV
Expiry date
Phone number
N
N
has
has
Doctor ID
Hospital name
Name
Age
Specialization
Arriving time
Leaving time
1
Doctor
1
assign to
1
Hospital
1
Hospital_ID
MOH registration code
Hospital Name
Manager name
Address
Telephone number
Location

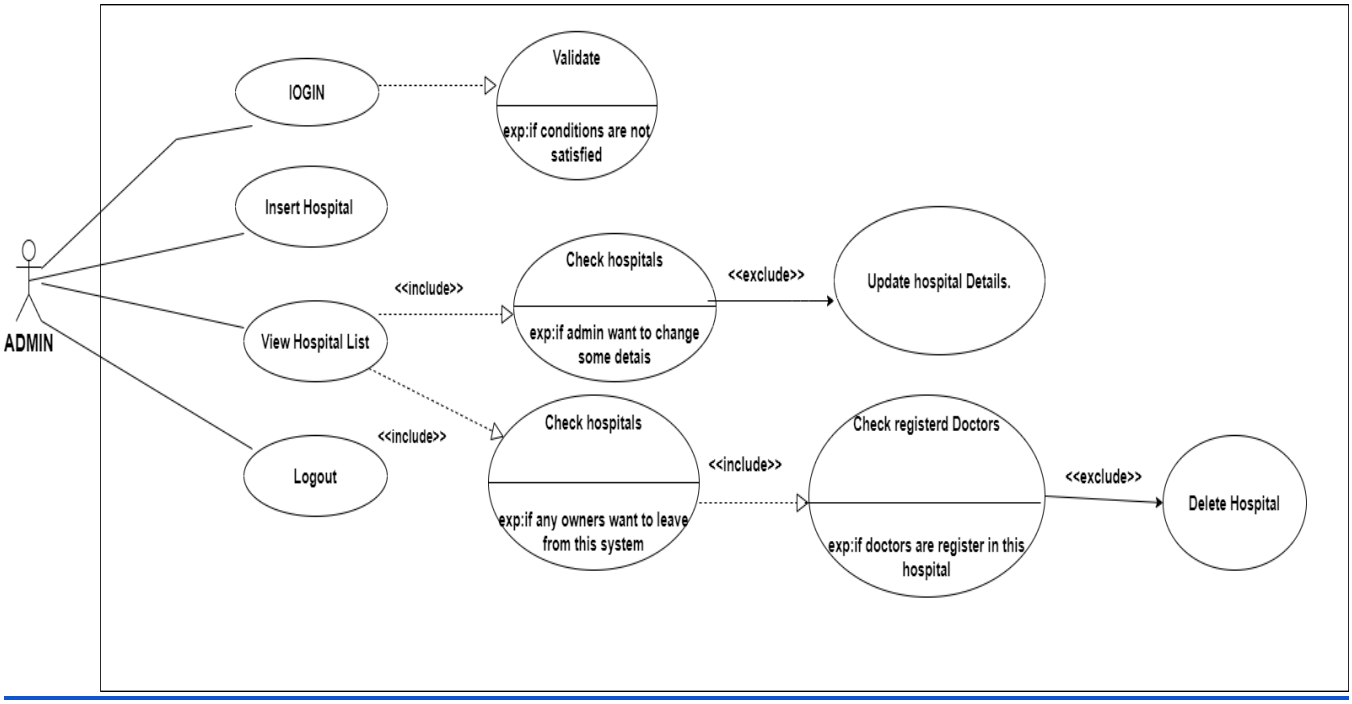

Register and Login to the System
Open Hospital Management System
[If user registered]
Register
Navigate to register form
Fill the required information
[If creditions wrong]
[If the given details exist]
Display Successful Message
Login
Fill the required information
Reset Username or password
[If login details forgotten]

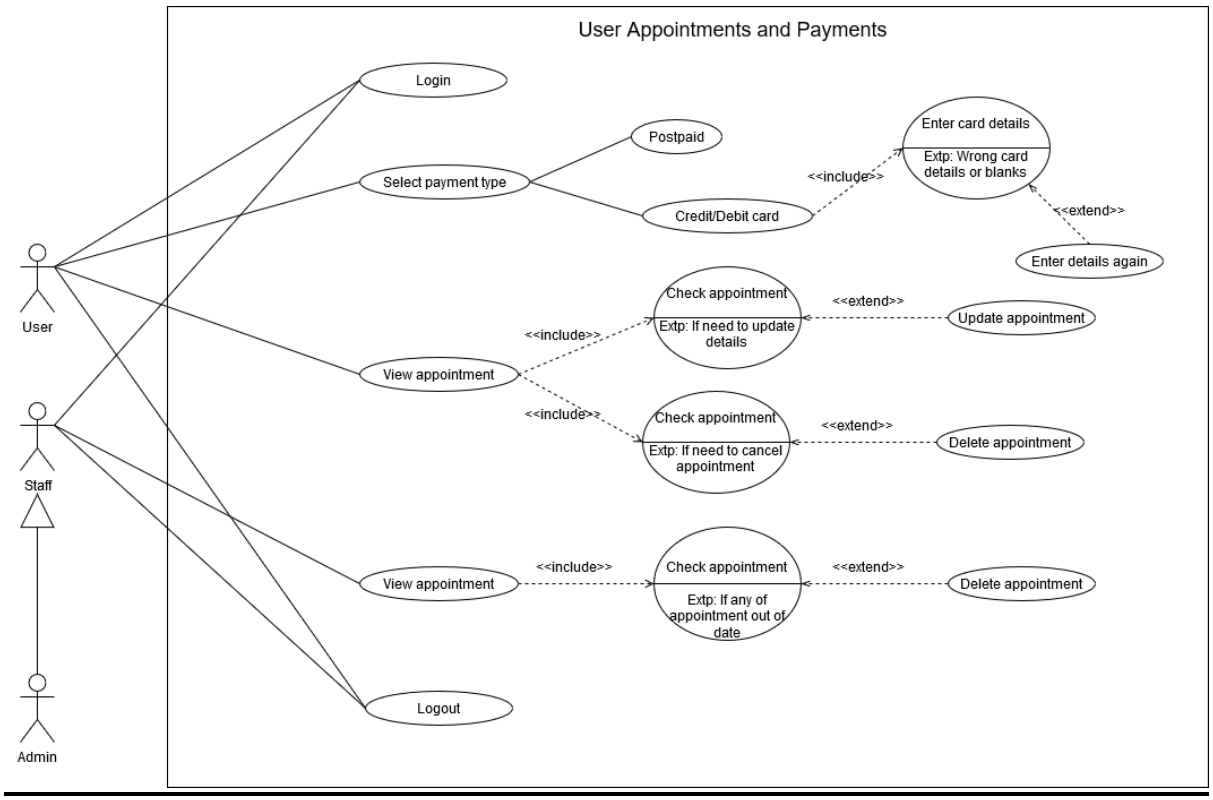

User Payments
Open Hospital Management System
Login
Go to appointment tab
Input required Details
[If given details are wrong]
Select Payment type
Postpaid
Credit Card
Debit card
Fill card details
[If any conditionals were wrong]


Doctor Management
Open Hospital Management System
Login
Go to Doctor Management
Fill doctor input form
[If any provided details are wrong]
View Doctors list
[If any of doctor leave the hospital permanantly]
[If any of doctor details need to update]
Delete Doctor
Update doctor


Hospital Management
Open Hospital Management System
Login
Go to Hospitals Management
Fill hospital input form
[If any provided details are wrong]
View Hospitals list
[If any of hospital abandoned permanently]
[If any of hospital details need to update]
Delete hospital
Update hospital


Doctors Management System
Admin
Login
Insert Doctors
View Doctors' list
Logout
<<include>>
Validate
Extp: If any of conditionals wrong
<<extend>>
Enter details again
<<include>>
Check doctors
Extp: If any of details need to be changed
<<extend>>
Update doctor
<<include>>
Check doctors
Extp:If any of doctor leave the hospital permenently
<<extend>>
Delete doctor


ADMIN
IOGIN
Insert Hospital
View Hospital List
Logout
Validate
exp:if conditions are not satisfied
<<include>>
Check hospitals
exp:if admin want to change some detais
<<exclude>>
Update hospital Details.
<<include>>
Check hospitals
exp:if any owners want to leave from this system
<<include>>
Check registerd Doctors
exp:if doctors are register in this hospital
<<exclude>>
Delete Hospital


User Appointments and Payments
Login
Select payment type
Postpaid
Credit/Debit card
Enter card details
Extp: Wrong card details or blanks
<<include>>
<<extend>>
Enter details again
User
Check appointment
Extp: If need to update details
<<extend>>
Update appointment
<<include>>
View appointment
<<include>>
Check appointment
Extp: If need to cancel appointment
<<extend>>
Delete appointment
Staff
View appointment
<<include>>
Check appointment
Extp: If any of appointment out of date
<<extend>>
Delete appointment
Logout
Admin