**SSDI_Team_One**
Arthur Carroll, Manju Jacob, Mitesh Peshave

# <Medical Information System>
# Software Design Document

**Version <1.0>**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 11/21/2009 | 1.0 | Working Copy | Arthur Carroll et. all |
| | | | |
| | | | |
| | | | |

# Table of Contents

# Software Design Document

## 1. Introduction

### 1.1 Purpose

This document describes the Medical Information System online application. This document is intended to serve as a guide for programmers during the implementation of said system. Once the system is developed, this document can help in testing the system for the required quality standards. Finally, this document can also be used by developers as a reference when maintaining or expanding the functionality of this software.

### 1.2 Scope

This document provides an overview of the Medical Information system. This includes the project's vision, scope and stakeholders. This document also gives an overview of the architectural design of the system. It lists the architectural goals and constraints to be considered. Finally, this document provides the different design views of the system such as Logical view, Process view and Deployment view. Quality standards for the system are also listed for quality assurance testing.

### 1.3 Definitions, Acronyms, and Abbreviations

**HTML** which stands for **Hyper Text Markup Language**, is the predominant markup language for web pages.

**HTTP** is short for **H**yper**T**ext **T**ransfer **P**rotocol, the underlying protocol used by the World Wide Web.

**JavaScript** is a scripting language used to enable programmatic access to objects within HTML DOM hierarchy and handle user events.

**JSON** short for **JavaScript Object Notation**, is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). It is basically a key value pair notation.

The **Apache HTTP Server**, commonly referred to as **Apache** is a web server notable for playing a key role in the initial growth of the World Wide Web.

**Servlets** are Java programming language objects that dynamically process requests and construct responses.

**MySQL** is a relational database management system (RDBMS). MySQL stands for "My Structured Query Language".

**Model-View-Controller (MVC)**: An architecture for programs that have graphical user interfaces. The programs are distinguished by a distinct separation in the User Interface, Business Logic, and Data.

**View model**: A View model is framework which provides the viewpoints on the system and its environment, to be used in the software development process. It is a graphical representation of the underlying semantics of a view.

**Thread:** In computer programming, a thread is placeholder information associated with a single use of a program that can handle multiple concurrent users.

**Process:** A process is an instance of a program running in a computer. It is close in meaning to task, a term used in some operating systems.

**J2EE: "**The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming."[2]

### 1.4 References

1.4.1 *Practical J2EE application architecture By Nadir Gulzar [1]*
1.4.2 *http://java.sun.com/j2ee/overview.html* [2]
1.4.3 *http://ootips.org/mvc-pattern.html* [3]
1.4.4 *Provided Template [4]*

### 1.5 Overview

This functional portion of the document consists of several distinct sections, which are described below:

1.5.1 Architectural Representation

This section gives a high level description of the architecture used in this system. [4]

1.5.2 Architectural Goals and Constraints

This section describes the software requirements and objectives that have some significant impact on the architecture. [4]

1.5.3 Use Case View

This section shows an updated version of the Use Case Diagram. [4]

1.5.4 Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. [4]

1.5.5 Process View

This view will display the processes that form the systems' mechanism. [4]

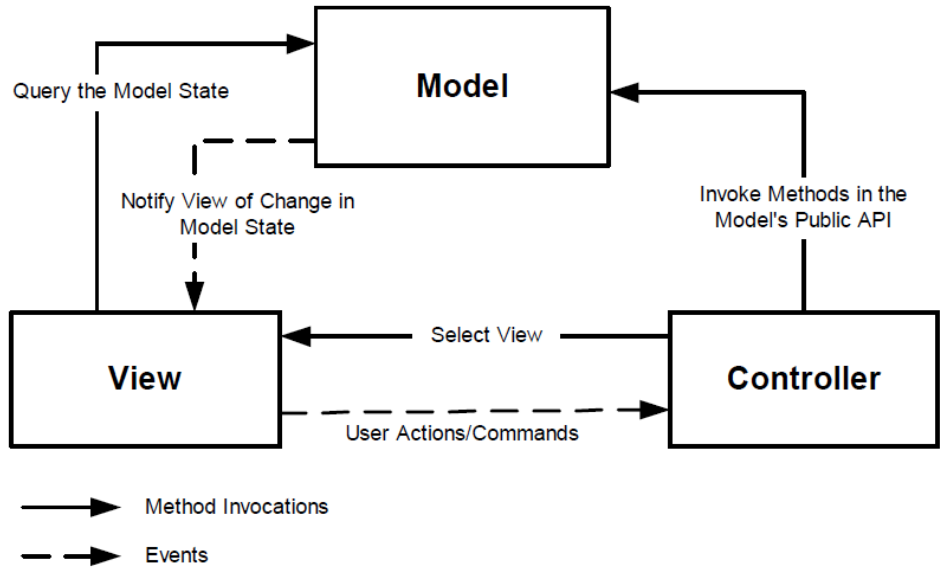
1.5.6 Deployment

This section describes one or more physical configurations on which the software is deployed and run. [4]

### 1.5.7 Design Rationale

This describes the motivation and reasoning for decisions made during the design process.

## 2. Architectural Representation




The architecture chosen for this project is the Model View Controller architecture. In this architecture the application is decomposed into three parts, the model, view and controller. The Model part will handle the business logic for accessing and manipulating data. The controller uses the services of the model for querying purposes and making a change in the state of the model. The model will notify the view whenever the state of the model is changed. The view part is responsible for interpreting and presenting the state of the model to the user. When the model communicates a change of state, the view modifies itself to reflect the change. The view also accepts inputs from the user and passes it to the controller. The job of the controller is to intercept and transform user input into actions to be performed by the model. Based on the user input and model operations, the controller will determine what view needs to be shown to the user.

We have followed the "4+1" view model of software architecture to depict the architectural design of our system. The following are the views we have used in our model.

**2.1 Logical view:** This view will support the functional requirements of the application and show an object model of the system. The overall structure of the design is represented using a block diagram. Class diagrams are also used to give a static view of classes and their logical relationships such as association, usage, composition, inheritance etc. The dynamic views of the important modules of the system are represented by sequence diagrams.

**2.2 Process view**: We have a multiple-client/ single server style of process view for our system. It shows a single server process that handles multiple client requests by spawning a new thread for each request.

**2.3 Implementation view**: (Development view): This view shows the files that are needed to run this software.

**2.4 Deployment view** (Physical view): This view shows how the software is mapped onto the hardware.

**2.5 Use case view** (Scenarios): This view has a small set of important scenarios in the form of a use case diagram and corresponding descriptions.

## 3. Project Overview

### 3.1 Project Introduction

In any field redundant paperwork is a drain on time and resources. Redundant paperwork also results in outdated information due to a lack of backward propagation. These two effects are especially detrimental in medical settings because of the scarcity of skilled personnel and the nature of the information. This project seeks to address these concerns by providing a unified system for managing patient information. This system will allow for healthcare professionals to make more efficient use of their time and ensure that all parties have the most current information.

### 3.2 Project Vision and Scope

The Medical Information System is designed for use by medical institutions in developing nations which could not otherwise afford such a system. The goal of this project is to provide an extensible platform which can be tailored to the specific needs of the institution using it. This project will provide the core functionality of this system and leave it to each institution to develop additional software modules suited to their specific needs.

### 3.3 Stakeholders

Stakeholders include any party that interacts with the medical information system directly or indirectly and have a stake in the system. This includes patients, doctors, nurses, receptionists, lab technicians, pharmacists, administrators and insurance companies.

3.3.1 **Patient**: Almost everything that happens in a medical information system revolves around the patient and patients are indirect stakeholders in this system. The patient

interacts with the receptionist and other medical personnel to provide inputs to the system and also receives medical reports, prescriptions and bills from the system.

3.3.2 **Doctors:** Whenever a doctor sees a patient he enters a new encounter into the system and provides the data to update patient records. Doctors also use the system to view their schedule and send notifications or messages to other medical personnel.

3.3.3 **Nurses:** Whenever a patient is admitted in a hospital ward, the nurse uses the system to view patient information and also updates patient information after doctor visits to include any new prescriptions, tests or dietary changes**.** Nurses can also view their schedule and use the system to send messages to doctors and the receptionist.

3.3.4 **Receptionist:** A new patient is registered and his demographic details are entered into the system by the receptionist. Appointment scheduling is handled by the receptionist whenever a patient needs to schedule or cancel doctor appointments. The receptionist also does ward management and is involved in allotting beds to patients admitted to the hospital. Medical bills are generated by the system and the receptionist collects payments from patients and enters payment details into the system. Insurance claims are generated by the system and the receptionist sends it out to the appropriate insurance companies. Notifications can be sent by the receptionist to doctors and nurses as needed.

3.3.5 **Lab technicians:** The results of lab tests can be entered into the system by lab technicians. They can also use the system to see the schedule for various lab tests to be done daily or weekly.

3.3.6 **Pharmacists:** Medicines can be allotted to patients by pharmacists using the system. The pharmacist can check the patient's prescription history and provide the prescribed medicines to patients. An invoice is generated and the patient's bill is updated. They can also use the system to check the expiry date of the medicines and monitor the reorder level of the medicines stock.

3.3.7 **Administrators:** Users of the medical system are added by administrators. They assign appropriate privileges to different kinds of users and also can modify or delete user profiles. Administrators can send messages to medical personnel to notify them about system related matters like system outages or maintenance.

3.3.8 **Insurance companies:** Claims are sent out on behalf of patients to collect payments from insurance companies and so they are indirect stakeholders in the system.

## 4. Architectural Goals and Constraints

The architectural goals of this system are as follows:

**4.1** The separate modules in this system must be distinct and separated by well defined interfaces.

**4.2** Any of the given modules can be individually replaced with no affect to the system.

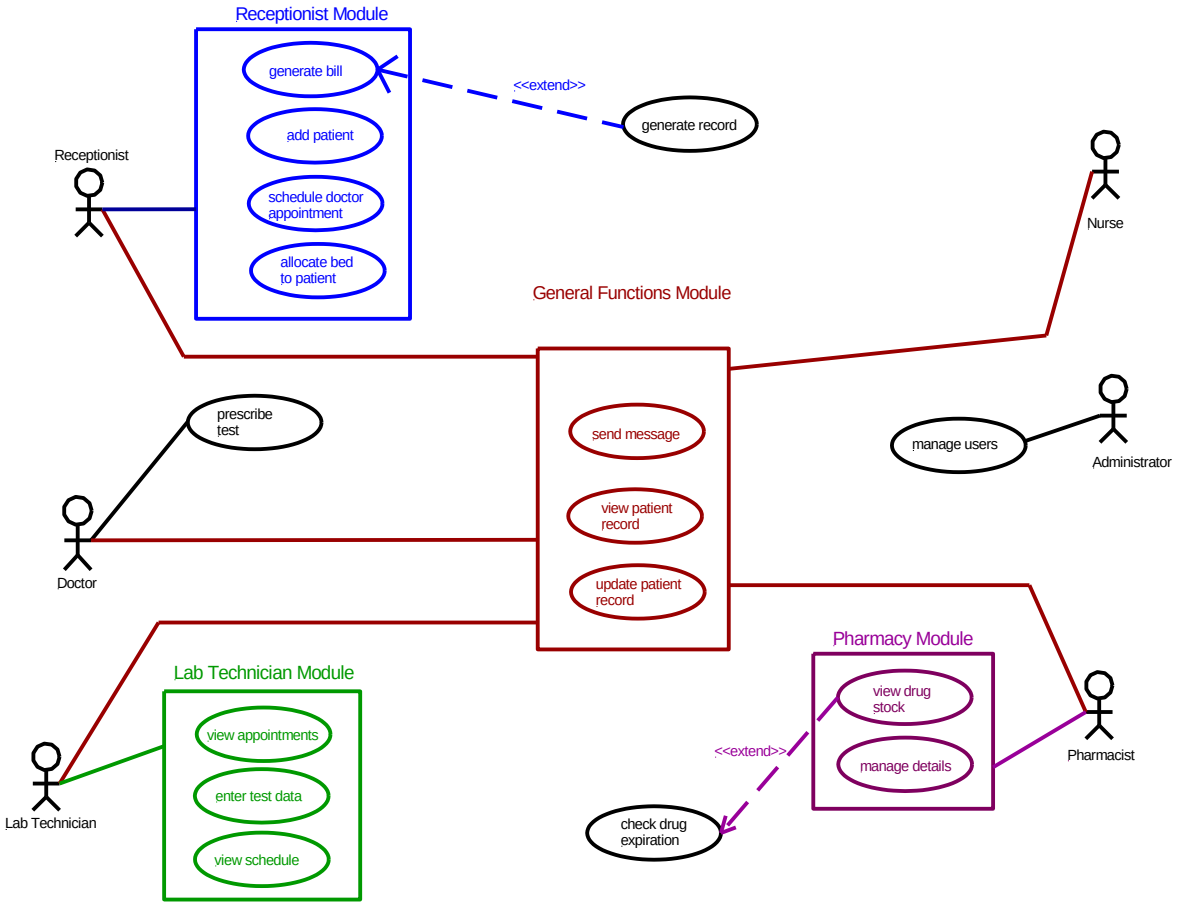
**4.3** A developer working on one module should not be required to have any knowledge of the internal workings of the other modules in the system.

**4.4** The individual modules in this must use said interfaces to interact with each other.

**4.5** Each module should employ the latest security standards for its given technology to defend against known attacks.

## 5. Use-Case View

In the use case diagram below, Medical Personnel has been broken down into lower level actors, such as doctors, nurses, pharmacists, etc. At this point, the team has not made a final decision on how many access control levels there will be. For example, a doctor and nurse may have the same access levels in a system, but this is yet to be determined.
The choice of showing all the individual actors in the diagram below was made because this would be the best case situation. In the final system the only distinguishing feature between these actors will be the screen which is being viewed at the time. Permissions set by the Administrator user will determine which user has rights to view each screen.


Receptionist Module
generate bill
add patient
schedule doctor appointment
allocate bed to patient
<<extend>>
generate record
Receptionist
Nurse
General Functions Module
send message
view patient record
update patient record
prescribe test
Doctor
manage users
Administrator
Lab Technician Module
view appointments
enter test data
view schedule
Lab Technician
Pharmacy Module
view drug stock
manage details
<<extend>>
check drug expiration
Pharmacist

### 5.1 Use-Cases

### 5.2 Actor Descriptions

5.2.1 Medical personnel – This actor class includes all hospital personnel who have access to the medical information system. This includes doctors, nurses, lab technicians, and receptionists. The system does not distinguish users by their title, only by which screen the user is viewing at the time.

5.2.1.1 Doctors: When a patient comes for a doctor appointment, the doctor creates a new encounter for the patient and updates patient record.

5.2.1.2 Nurses: The system allows nurses to view and modify their daily and weekly schedules. Nurses can also update the records of patients admitted to a ward.

5.2.1.3 Lab technicians: Results of investigative tests are entered into the system by Lab technicians.

5.2.1.4 Receptionists: Patient registration, appointment scheduling, bed allotment and bill payments are handled by receptionists.

5.2.2 Administrators – Users of the system are managed by Administrators. They are not concerned with the day to day aspects of the system such as patient records and test results.

### 5.3 Overview of Use Case Descriptions

5.3.1 Manage users: Administrator can add, modify and delete users and also assign appropriate privileges to different users.

5.3.2 Add patient: Receptionist registers a new patient and enters his demographic details into the system.

5.3.3 Schedule doctor appointments: Receptionist will schedule or cancel doctor appointments.

5.3.4 View patient record: Receptionists, doctors and nurses can search for a patient and view his details.

5.3.5 Generate bill: Bills are consolidated and generated by the system and given to patient. Receptionist enters bill payment information into the system after collecting payments from patient.

5.3.6 Allocate bed to patient: The Receptionist will allot a bed to a patient that needs to be admitted in the hospital ward.

5.3.7 View appointments: Lab technicians can see a list of appointments for the day.

5.3.8 View schedule: Lab technicians can view their daily/weekly schedule.

5.3.9 Enter test data: Results of investigative tests done are entered into the system by technicians.

5.3.10 Edit Patient record: Patient records can be edited by doctor, nurse and receptionist.

5.3.11 Prescribe tests: Doctor can prescribe tests needed for patients and system updates patient's record.

5.3.12 Update patient record: Patient records can be updated by doctors, receptionists, nurses and Lab technicians.

5.3.13 Send messages: Notifications can be sent to other medical personnel by Administrators, doctors, nurses and receptionists.

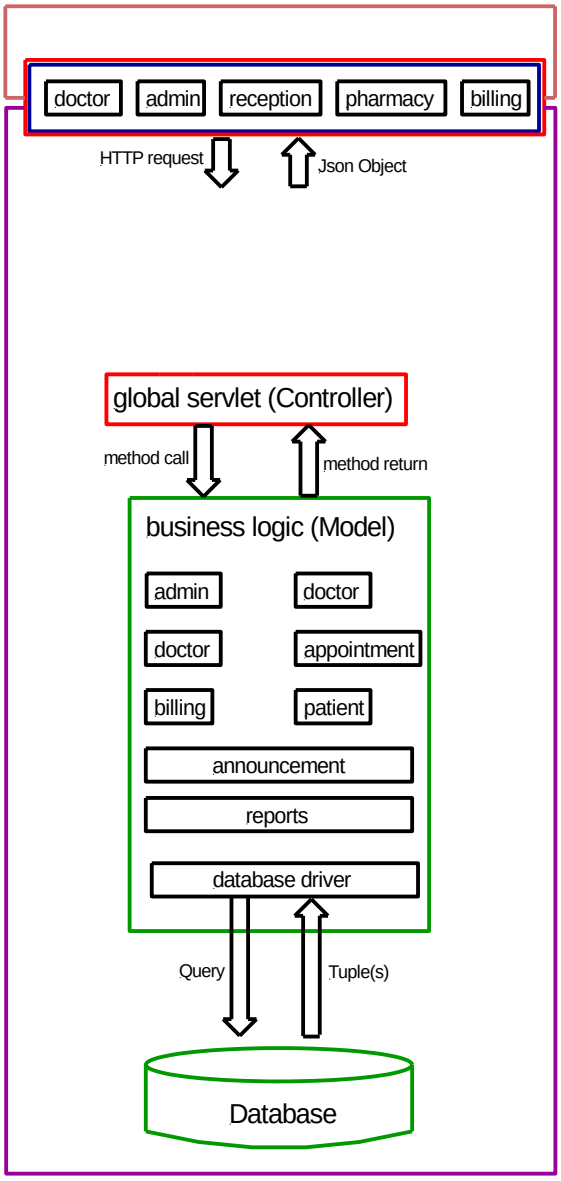
5.3.14 Manage details: Pharmacist will allot medicine to patients as per doctor prescription after checking the expiration date. The system updates the patient's prescription history and billing information.

5.3.15 View medicine stock: Pharmacist can view stock of medicines to see if any drugs need to be reordered.

# 6. Logical View

## 6.1 Overview

For this application the architecture has been somewhat modified. In this modified MVC architecture, the model has various modules such as doctor, appointment, patient etc that contain the business logic for the system. For the view component there are multiple views that different kinds of users such as doctor, receptionist, pharmacist etc. see. The controller is divided in two parts; one part on the client which handles the events such as button clicked etc. and invokes proper action/request to be sent back to the server, the second part of the controller is on the server side which handles the requests. Depending on the request, the server side controller interprets and delegates the request to handler modules in the model.




Color Codes:
Model
Controller
View
Server
Client PC

The diagram above provides a high level overview of the system and calls in the system. The diagram represents three components as follows:

6.1.1 **View** - the system has different views depending on the credentials of the user logged into the system. The important views in our system are as follows:

6.1.1.1 Doctor - the system shows announcements and tabs relevant to the doctor when a doctor logs on. From this view a doctor can see his appointment schedule and view details of a particular patient. When a patient has an appointment the doctor can create an encounter for the patient to record the diagnosis, procedures, and any prescriptions of medicines or investigative tests that may be required.

6.1.1.2 Admin – the system shows announcements and tabs relevant to the Administrator when he logs on. From this view the Administrator can add, modify, delete or view user details.

6.1.1.3 Reception – the system shows announcements and tabs relevant to the Receptionist in this view. The Receptionist can navigate to the Scheduling screen whenever a new appointment needs to be scheduled. New patient details can also be entered into the system from this view by navigating to the Patient registration view. This view has a Billing tab that allows for viewing patient bills and updating bill payment details.

6.1.1.4 Pharmacy – the system shows announcements and tabs relevant to the Pharmacist in this view. The Pharmacist can see a list of available medications with details such as name, manufacturer, expiry date, stock quantity etc. When a prescription is received by the Pharmacist he can allot medicine to the patient, update the medicine stock and also update the cost of the medicines to the patient's bill.

6.1.2 **Global controller** - the controller is a component that provides functions to intercept incoming requests scan them for parameters, and depending on the parameters, invoke the correct model module in the systems business logic layer.

6.1.3 **Model** (business logic layer) - This component consists of the modules that handle the business logic for the different requests coming from the client. The important modules in this component are as follows:

6.1.3.1 Doctor: This module has methods that interact with the database to retrieve and update patient records. The database is also accessed by another method in this module to retrieve list of appointments for the doctor.

6.1.3.2 Appointment: This module has methods that interact with the database to retrieve or update appointment schedules.

6.1.3.3 Patient: This module has methods that interact with the database to add, view or update patient records.

6.1.3.4 Admin: This module has methods that interact with the database to add, view or update user records.

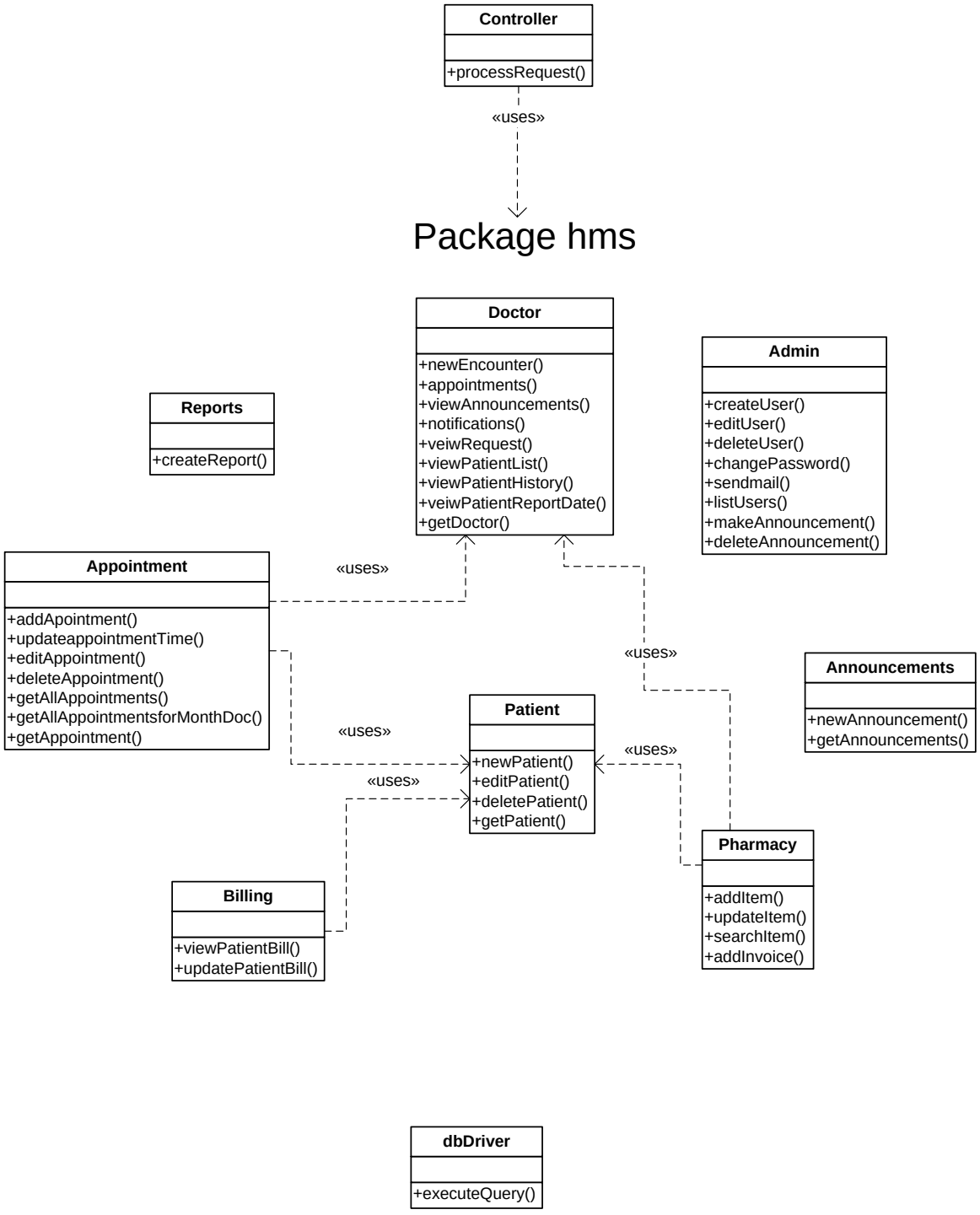
6.1.3.5 Billing: This module has methods that interact with the database to retrieve and update a patient's billing records.

6.1.3.6 Pharmacy: This module has methods that interact with the database to retrieve or update drug details and also update patient bill details.

6.1.3.7 Announcements: This module has methods that interact with the database to add, view, modify or delete announcements for various users of the system.

6.1.3.8 Reports: This module has methods that interact with the database to generate reports for various users.

### 6.2 Architecturally Significant Design Packages



The diagram above shows the main class diagram of the system. All the classes in the system are static which means that none of the classes will be instantiated. All the methods in the classes will also be static. All of the classes except controller come under the same package hms. The controller class is not in any package. The class controller interacts will every uses every class except for dbDriver, for the sake of clarity this is not depicted in the diagram. The functions of the important classes in the system are listed below.

### 6.2.1 **Patient**

#### 6.2.1.1 Static View

This class manipulates data related to a patient. It interacts with most of the major classes such as Appointment, Doctor, Encounter, Pharmacy and Billing. The methods in this class interact with the database to add, view or update patient records.

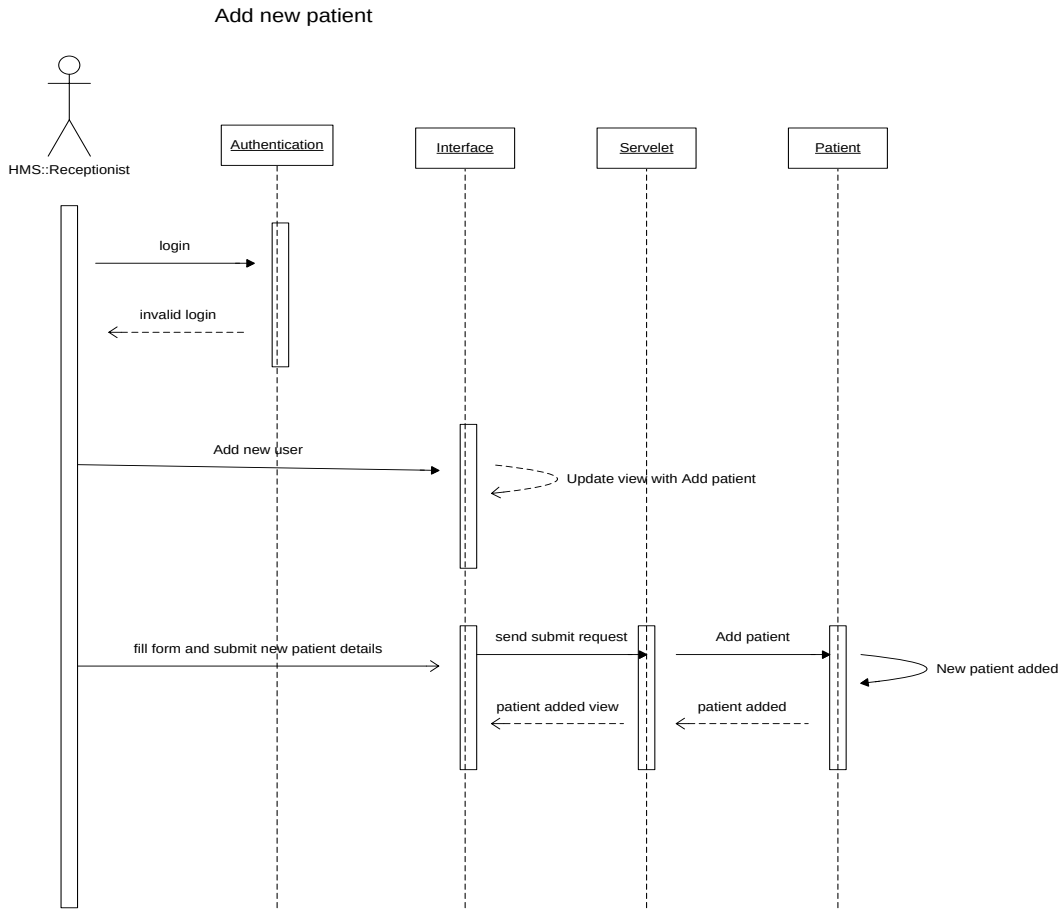
#### 6.2.1.2 Dynamic View

Major public methods of this class are as follows:

viewPatientHistory() : This function takes in a patient id and returns the records related to the patient from the database. This function will search and display patient details for every encounter, along with test reports and prescriptions.

updatePatientRecord() : This function takes in the patient id and the details to be updated for the patient. The details can be a new prescription, a report or an invoice.

addPatient() : This method will register a new patient in the system. It will accept new patient details such as name, date of birth and other demographic details, validates the data and adds a new patient record to the database.




### 6.2.2 **Doctor**

#### 6.2.2.1 Static View

This class handles all the doctor functions in the system such as adding an encounter after examining a patient, viewing patient list, viewing patient history etc. It interacts with the classes such as Appointment and Encounter. This module has methods that interact with the database to retrieve and update patient records. The database is also accessed by another method in this module to retrieve list of appointments for the doctor.

#### 6.2.2.2 Dynamic View

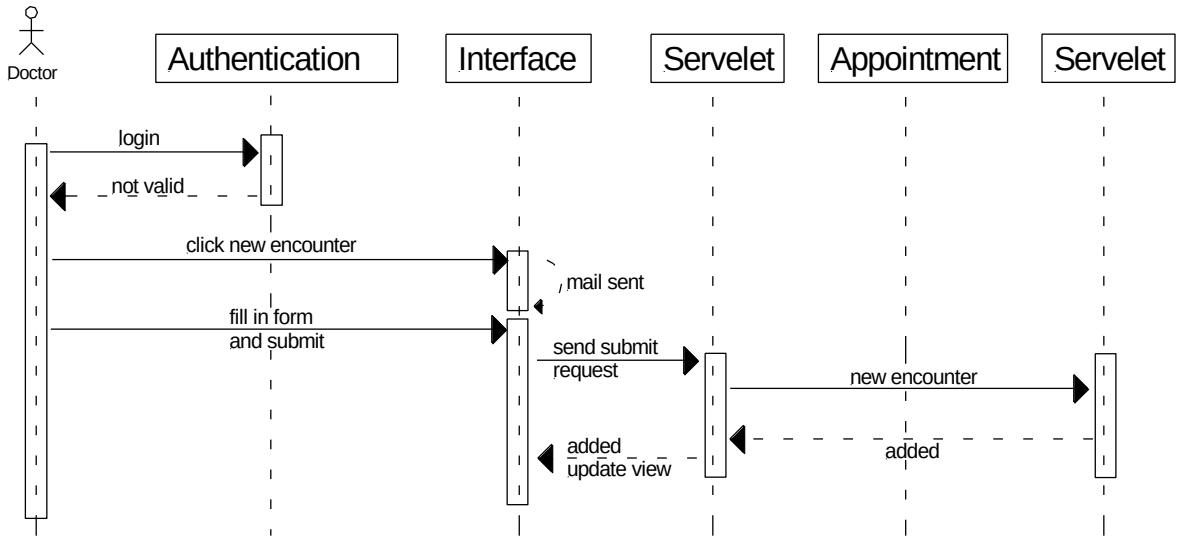
Major public methods of this class are as follows:

viewSchedule() – This function displays the schedules for a particular doctor.

addEncounter() – This function takes in the appointment id, and the details of the

encounter entered by the doctor in form. Each encounter is associated with an appointment with a unique appointment id and an encounter id.

Doctor New Encounter




### 6.2.3 **Appointment**

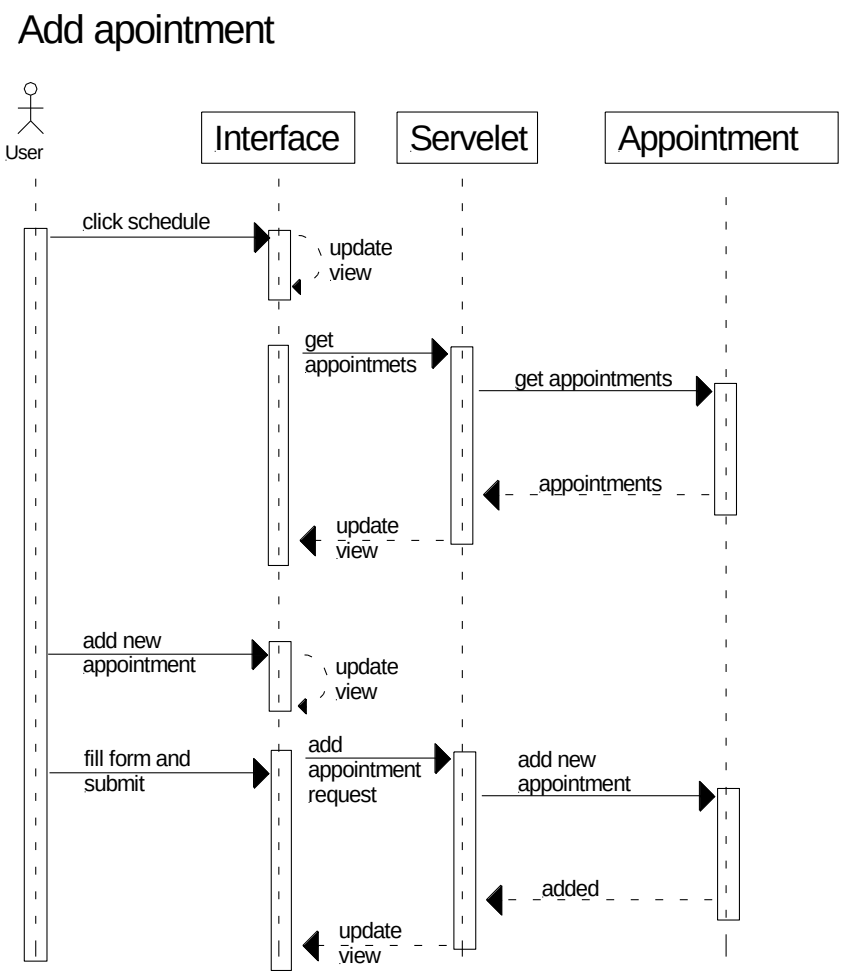
#### 6.2.3.1 Static View

This class will handle all the requests related to scheduling a doctor appointment. The requests could be adding an appointment, editing an appointment, or adding details to an appointment. This class will interact with Patient and Doctor classes.This module has methods that interact with the database to retrieve or update appointment schedules.

#### 6.2.3.2 Dynamic View

Major public methods of this class are as follows:

addAppointment() – This function will take as parameters details of the appointment such as patient name, time, date and the doctor id.




updateAppointmentTime() – This function will take in appointment id, and the changed time as input parameters. The function will update an existing appointment.
getAppointmentForMonthForDoc() – This function will fetch all the appointments for the given doctor id and the month specified in the input parameters. It will be invoked only when a doctor logs in to the system to view his schedule.
getAppointmentsForMonth() – This function will fetch all the appointments for all the doctors in the specified month. This function will be invoked only when a receptionist logs into the system and wants to view, add, or update the appointments of a doctor.

### 6.2.4 **Admin**.

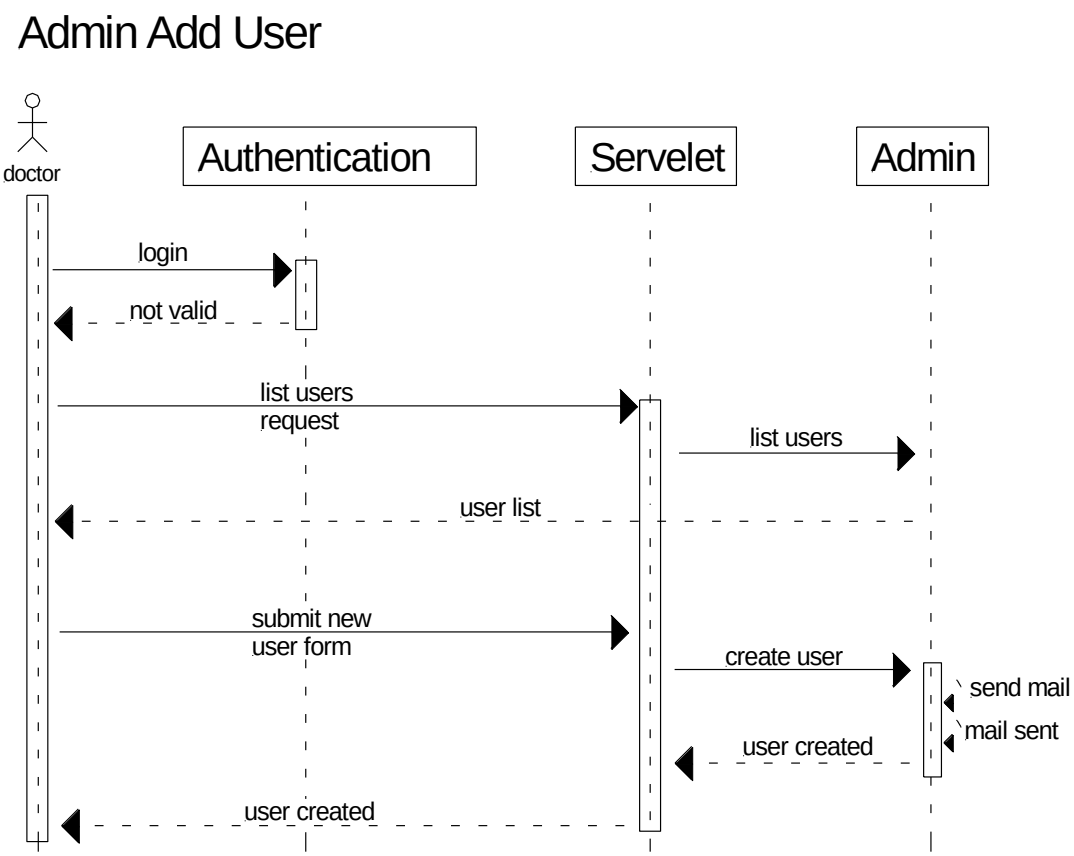
#### 6.2.4.1 Static View

Functions in this class are mostly perform administrator level functions such as creating a new user in the system, updating/editing user information or resetting passwords for the users. The methods in this class interact with the database to add, view or update user records.

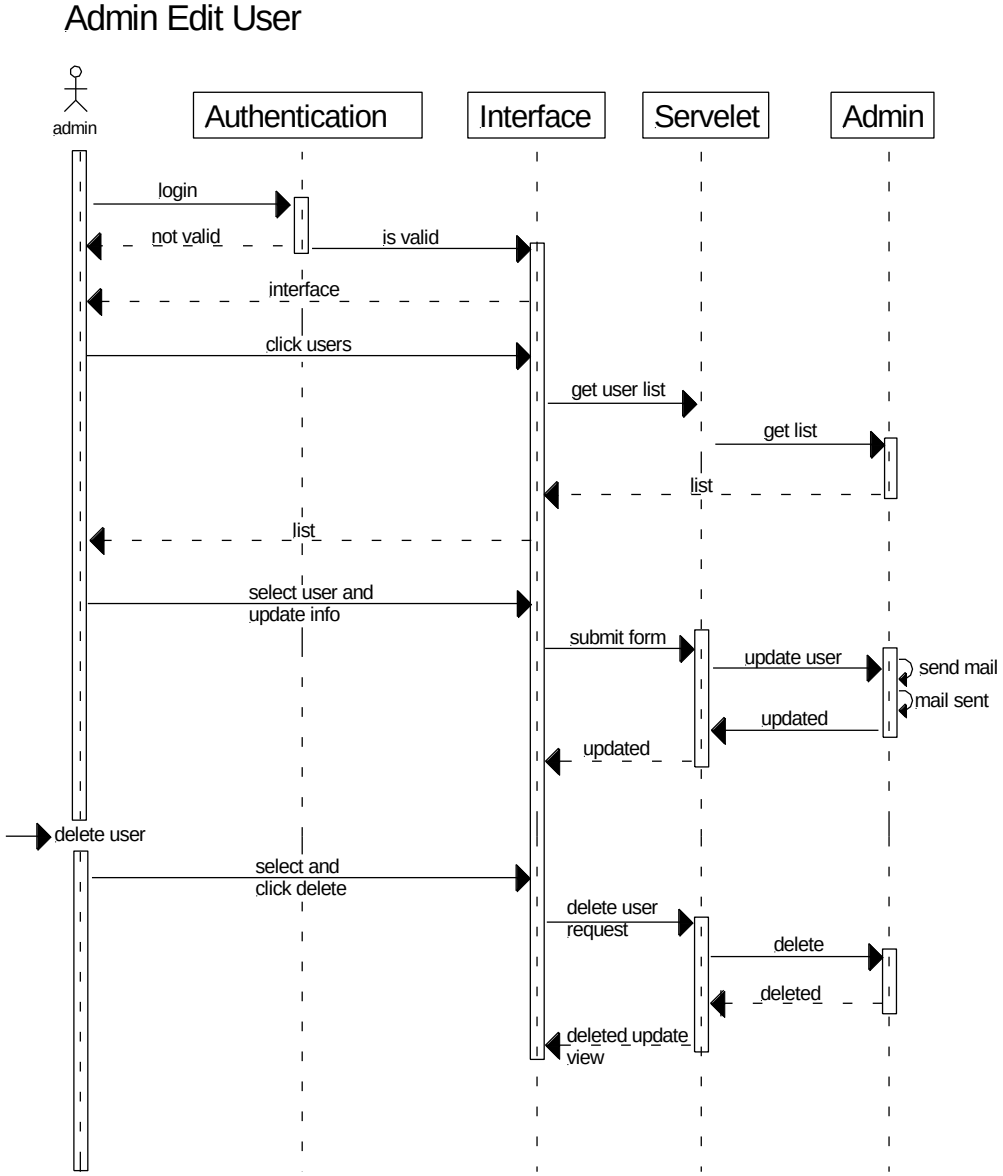
6.2.4.2 Dynamic View

Major public methods of this class are as follows:

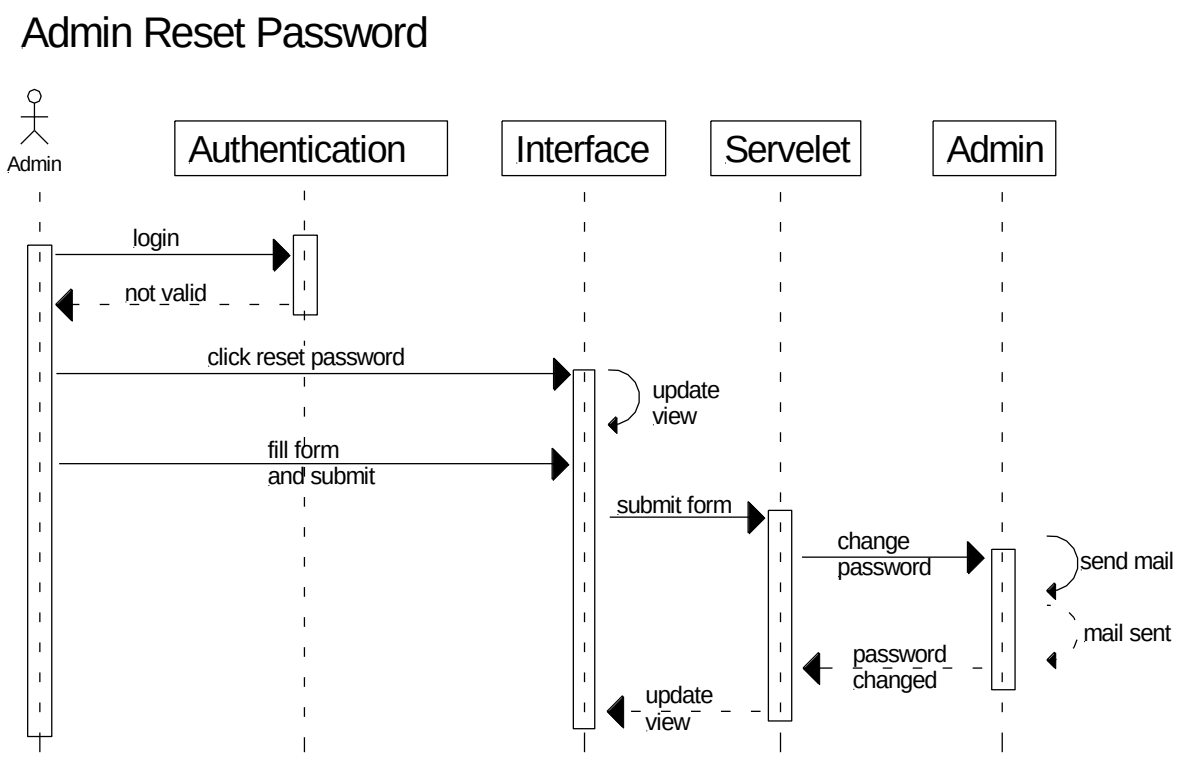
addUser() : This function accepts the details for a new user. It validates the data and checks if the username is already in use. If the username already exists, it notifies the admin that the username already exists. Otherwise the new user is created and a mail is sent to the user with his sign in credentials.



editUser() : This function accepts the details to be changed for a user and updates the user details in the database.




resetPassword() – This function should take in userid and the new desired password, update it in the database and send the user an e-mail, notifying him about the password change.




### 6.2.5 **Pharmacy**

#### 6.2.5.1 Static View

This class manipulates data related to the stock and invoice generation in the pharmacy module. It interacts with the Patient and Billing module. The methods in this class interact with the database to add, view or update drug details and to add invoices to patient bills for generating final patient bill.

#### 6.2.5.2 Dynamic View

Major public methods of this class are as follows:

addInvoice() : This function takes in all the details for the invoice of a patient and associates them to a patient id. This function should generate a unique id for each invoice before storing the data in the database. This data will be used in the billing module of the system while generating the final bill for the patient.
updateStockItem() : This function will take a stock item id(drug id) and the details related to the item such as manufacturer, date of expiration and quantity in the stock. This function also manipulates data for the drug inventory.

### 6.2.6 **Billing**

#### 6.2.6.1 Static View

This class is responsible for displaying and generating a patient's bill and updating the patient's billing record when the patient pays the bill. It interacts with the Patient and

Pharmacy modules. This module has methods that interact with the database to retrieve and update a patient's billing records.

6.2.6.2 Dynamic View

Major public methods of this class are as follows:

generateBill() : This function takes in a patient id and generates the patient's bill.

payBill() : This function will accept payment details for a patient and update the patient billing record.

6.2.7 **Announcements**

6.2.7.1 Static View

This class is responsible for adding, viewing, modifying and deleting announcements for the various users in the system. It does not interact with any other class. This module has methods that interact with the database to add, view, modify and delete announcements.

6.2.7.2 Dynamic View

Major public methods of this class are as follows:

addAnnouncementsl() : This function allows announcements to be added to the database.

viewAnnouncements() : This function allows announcements to be retrieved from database.

modifyAnnouncements() : This function allows announcements to be modified and updated in the database.

deleteAnnouncements() : This function allows announcements to be deleted from the database.

6.2.8 **Reports**

6.2.8.1 Static View

This class is responsible for generating reports in the system. It interacts with most of the other c. This module has methods that interact with the database to retrieve required details and generate reports.

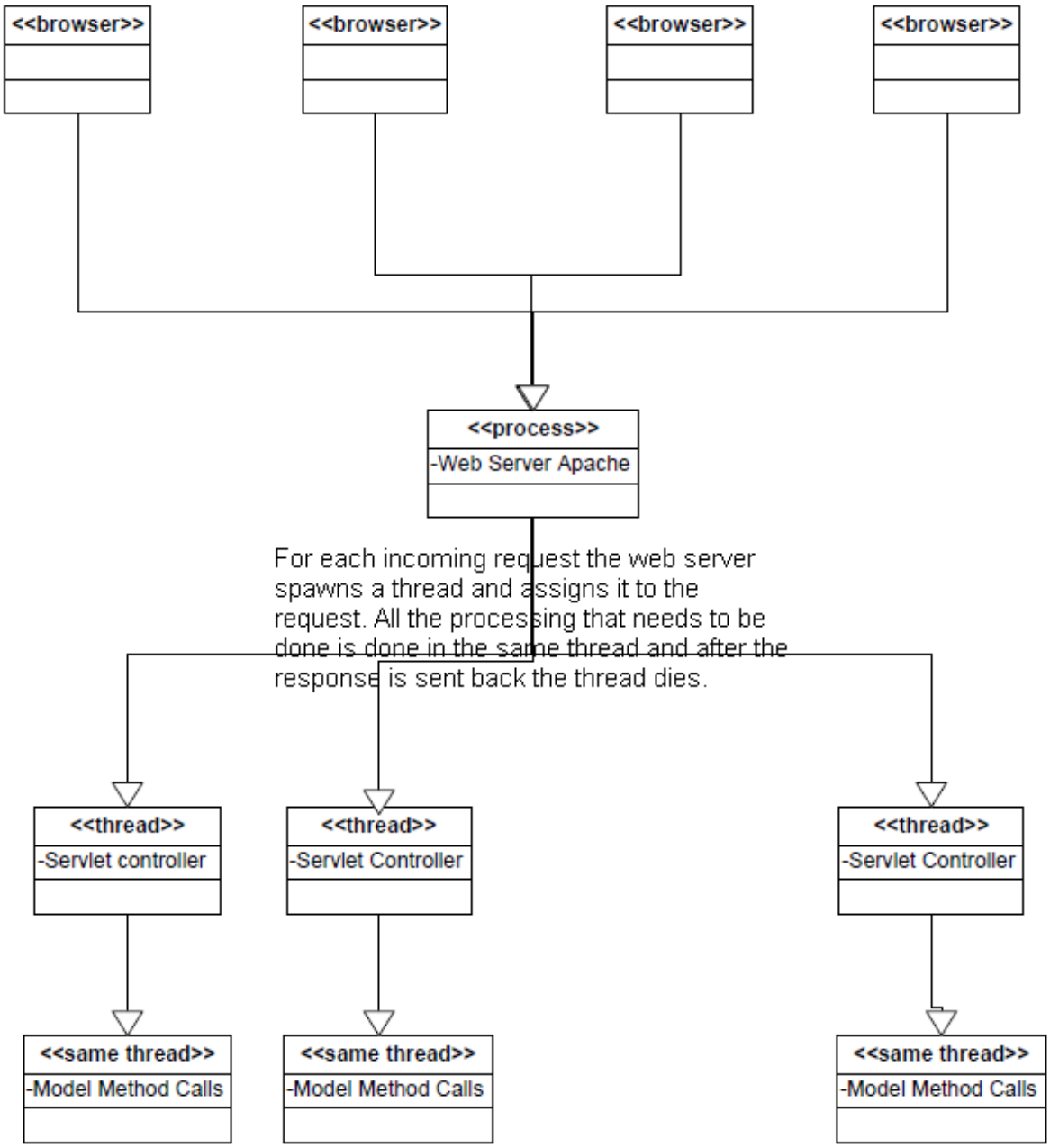
6.2.8.2 Dynamic View

Major public methods of this class are as follows:

generateReportl() : This function will generate reports after retrieving requested details from the database.

.

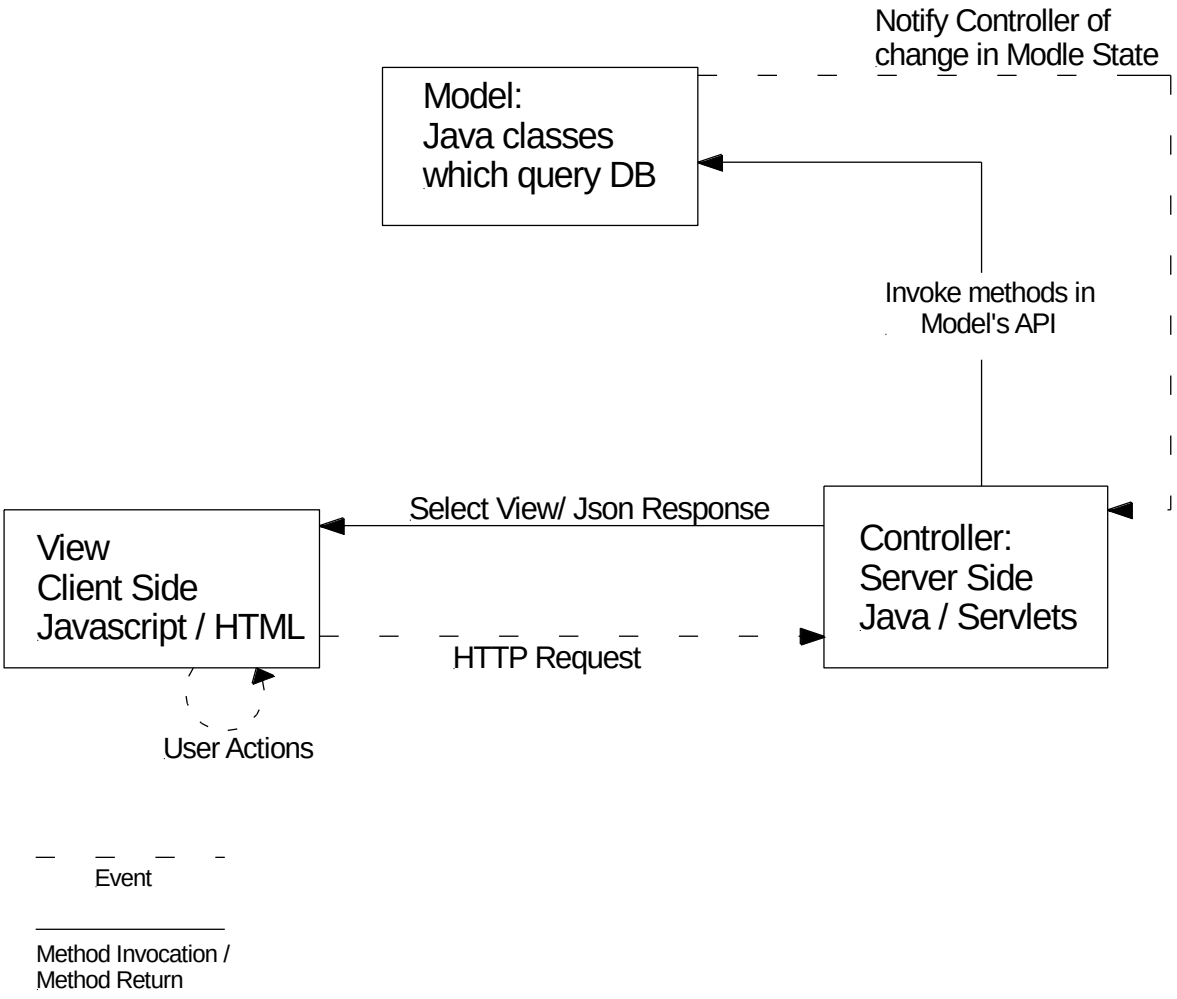
## 7. Process View

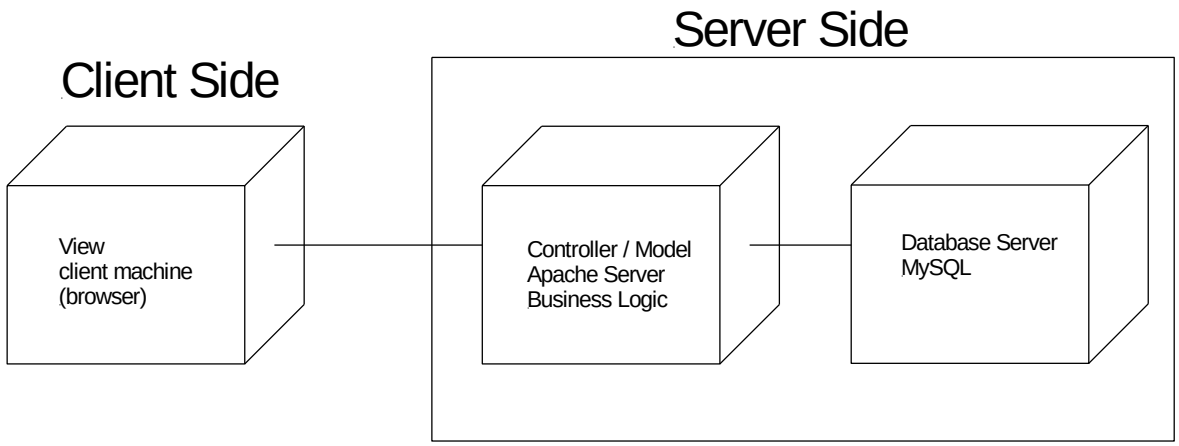



The web server is running as a single process.Whenever a client (browser) sends in a request to perform a specific function; the web server spawns a new thread and assigns it

to the appropriate Servlet. All the data fetching and data processing needed for the request is done in the same spawned thread. Once the response for the request is sent back, the thread dies. For each incoming request the web server spawns a new thread to service the request.

## 8. Deployment View




Server Side
Client Side
View
client machine
(browser)
Controller / Model
Apache Server
Business Logic
Database Server
MySQL

### 8.1 Client Side Technology

**JavaScript**

The client side views will be generated using JavaScript and HTML. These pages will be rendered by the browser. The HTTP Requests and Responses will also be handled by the browser.

### 8.2 Server Side Technology

**Operating Systems**

The Medical Information System will be developed using J2EE. The platform independence of the Java language means that the Medical Information System is deployable on any operating system for which a Java Virtual Machine is available. The Medical Information System will be deployed and tested on the following operating systems:

- Windows XP
- UBUNTU

These operating systems will have Apache Tomcat 6.0 as a web server under which the application will be deployed.

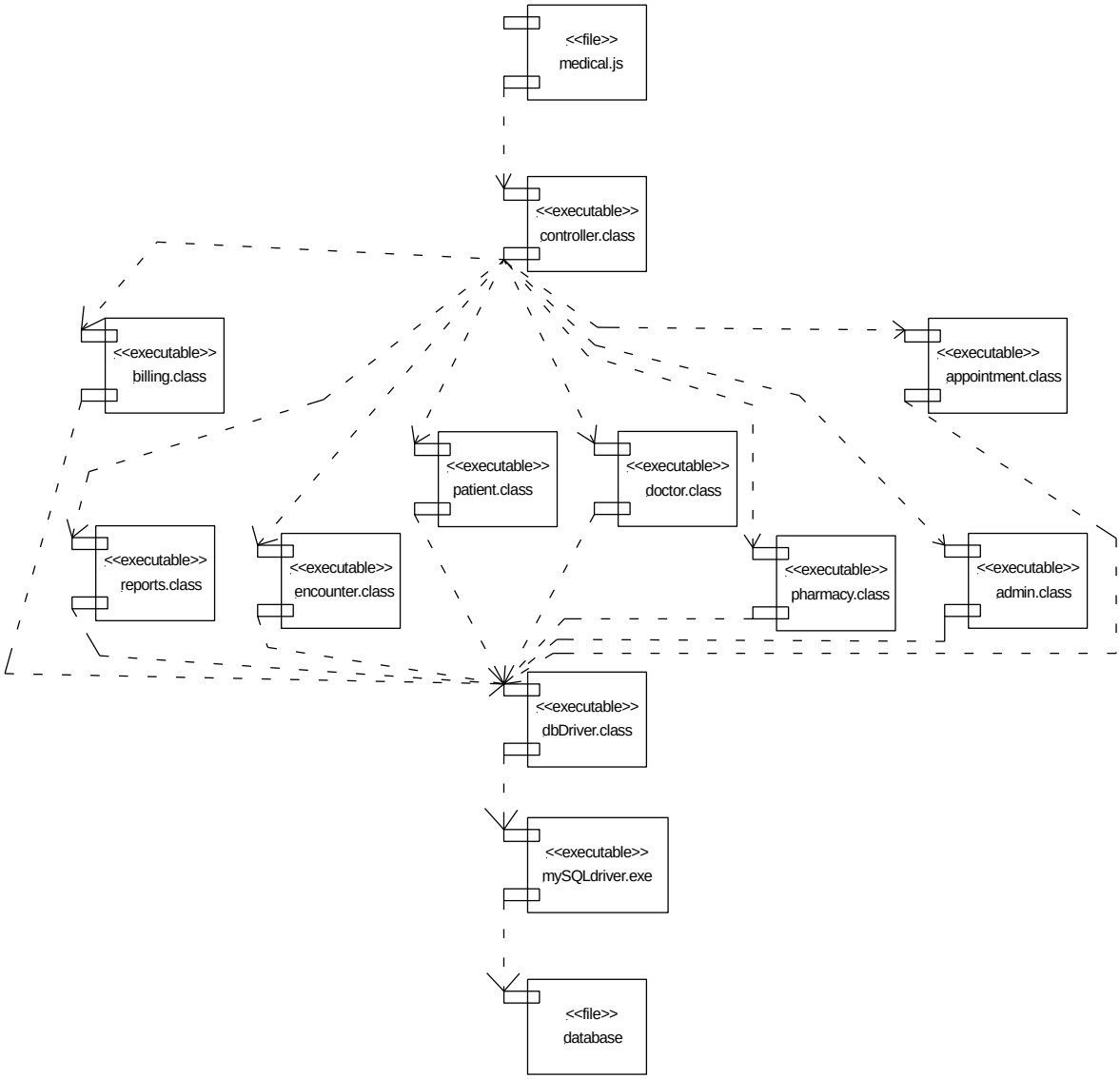
**Database Servers**

The Medical Information System will rely on MySQL 3.0 for the database. The database will be accessed through the Sun JDBC API since the JDBC driver provides an abstract layer for database access. The DBMS can be substituted to fit the customer's needs without modification to the source code.

## 9. Implementation View

### 9.1 Implementation View Description

The implementation view diagram shows which files are needed to deploy this system. Since all of the functionality of the system is server side, all these files are assumed to be deployed on the server. Note that in large systems it may be necessary to install mySQLdriver.exe and database on a dedicated machine. Decencies within the hms package are not shown improve diagram readability, see the main class diagram in section 6.2 for details on dependencies with package hms. The diagram below shows the implementation view of the system.


<<file>>
medical.js
<<executable>>
controller.class
<<executable>>
billing.class
<<executable>>
appointment.class
<<executable>>
patient.class
<<executable>>
doctor.class
<<executable>>
reports.class
<<executable>>
encounter.class
<<executable>>
pharmacy.class
<<executable>>
admin.class
<<executable>>
dbDriver.class
<<executable>>
mySQLdriver.exe
<<file>>
database

# 10. Quality

## 10.1 Usability

There will be thorough documentation of the system which will help new users to start using the system in a short time. The modern user interface will also make navigation to different screens very simple.

## 10.2 Meet Requirements

The final product will meet all requirements and specifications mentioned in the Software Requirements and Specifications document.

## 10.3 Regression Testing

Regression testing will be done to the system to ensure that existing components of the system are not affected whenever new components are added to the system.

## 10.4 Stress Testing

Stress testing will be done on the system to ensure that when the system is in demand the response time is not hampered much. This will be done because the HMS system will have a very daily intense usage.

# 11. Design Rationale

The design pattern used is the MVC (Model View Controller) design pattern. This pattern separates the application's business logic, controller and the presentation layers. The modularity of this design pattern allows for easy maintenance of the application. Even if there is a change in any one level of the application, be it view, controller or model, there is no need to change the entire hierarchy.

The other architecture we considered using was a layered approach. We did not use this model because of the increased processing and data overhead between the different layers that could affect system performance. Also if one of the layers is changed, all the layers in the hierarchy must be changed to accommodate the change.