University of the Witwatersrand

Architecture Description of Layered Architecture for HealFolio

Team Name:

Level Seven Crew

Product Name:

HealFolio

Team Members:

Jan Badenhorst
Adam Lerumo
Tumbone Asukile
Daniel da Silva

Senior Lecturer:

Dr. Terence van Zyl

September 12, 2016

Contents

Chapter 1

Introduction

This chapter describes introductory information items of the Architecture Description (AD), including identifying and supplementary information.

1.1 Identifying information

- The architecture being expressed is an Layered Architecture.
- The system of interest is HealFolio, an individual applications for which this is an architecture description.

Following ISO/IEC/IEEE 42010, *system* (or *system-of-interest*) is a shorthand for any number of things including man-made systems, software products and services, and software-intensive systems including “individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest”. ISO/IEC/IEEE 42010, 4.2

1.2 Supplementary information

1.2.1 Date of issue and status

- This project was started on the 24 of July 2016.

1.2.2 Authors

- Adam Lerumo - Product owner
- Tumbone Asukile - Developer
- Daniel da Silva - Developer
- Jan Badenhorst - Scrum Master

1.2.3 Reviewers

- Dr. Terence van Zyl - Senior Lecturer and intellectual guide.

1.2.4 Scope

The team will be designing software (HealFolio) aimed at improving service delivery for all types of medical practitioners; specifically the filing system currently used by medical practitioners. This software will provide doctors with easier access to patient records and provide more security. It will also help them make more accurate diagnoses, provide better prescriptions and more as the software is further developed.

1.2.5 Context

The software to be implemented will operate in the medical industry, predominantly among General Practitioners (GP's). The issues in this sector include things such as (but not limited to):

- Patient record sharing across many different practices.
- Security in the use of medical aid for visits to GP's and other practitioners.
- Patient reaction to prescribed medications.

The above list provides a brief outline of the core issues to be addressed by the project, more issues may be identified as the project progresses.

1.2.6 Overview

The Layered Architecture pattern the is being used in HealFolio, otherwise known as the n-tier architecture pattern. This pattern is the standard for most Java EE applications and therefore is widely known by most architects, designers, and developers. The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

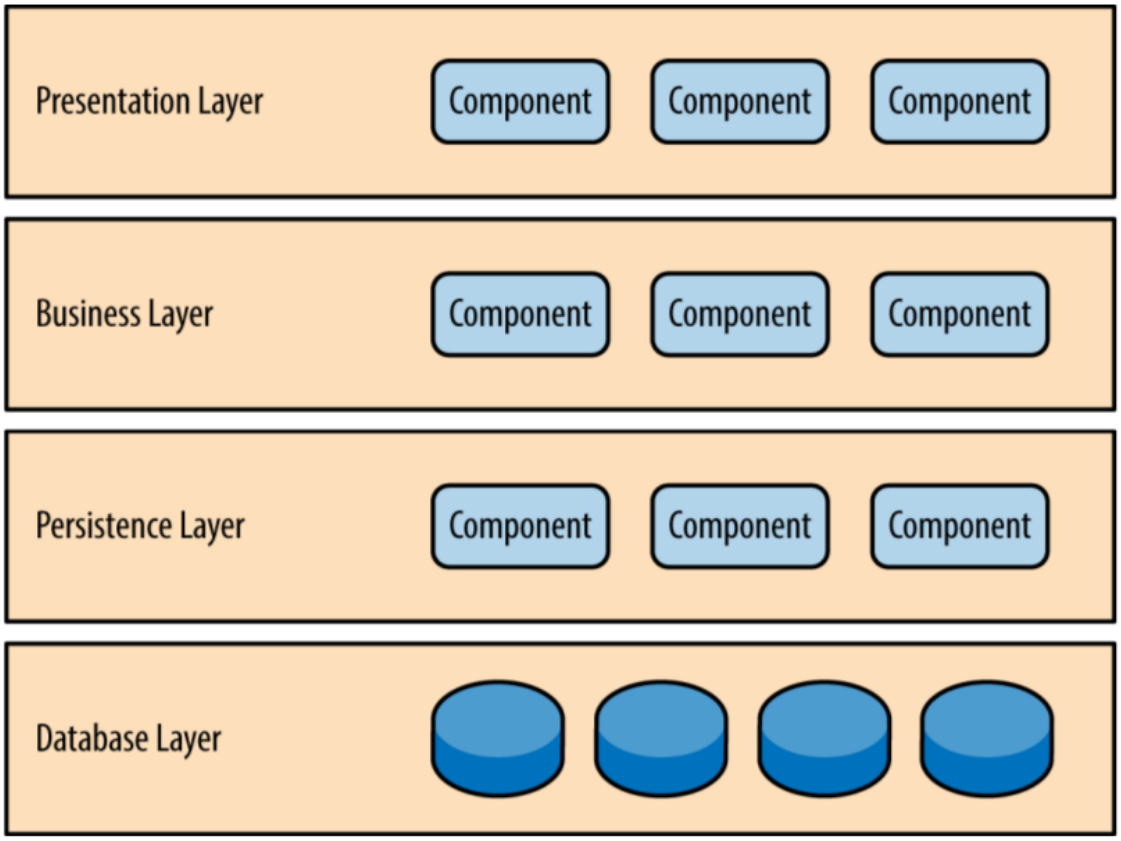


Figure 1.1: Layered Architecture Pattern

This project is leaning toward the Microservices Architecture pattern. There are several common core concepts that apply to this architecture pattern. The first of these concepts is the notion of separately deployed units. Each component of the microservices architecture is deployed as a separate unit, allowing for easier deployment through an effective and streamlined delivery pipeline, increased scalability, and a high degree of application and component decoupling within your application. Perhaps the most important concept to understand with this pattern is the notion of a service component. Rather than think about services within a microservices architecture, it is better to think about service components, which can vary in granularity from a single module to a large portion of the application. Service components contain one or more modules (e.g., Java classes) that represent either single-purpose function (e.g., viewing information for a specific patient) or an independent portion of a our application (e.g., signing up for the service).

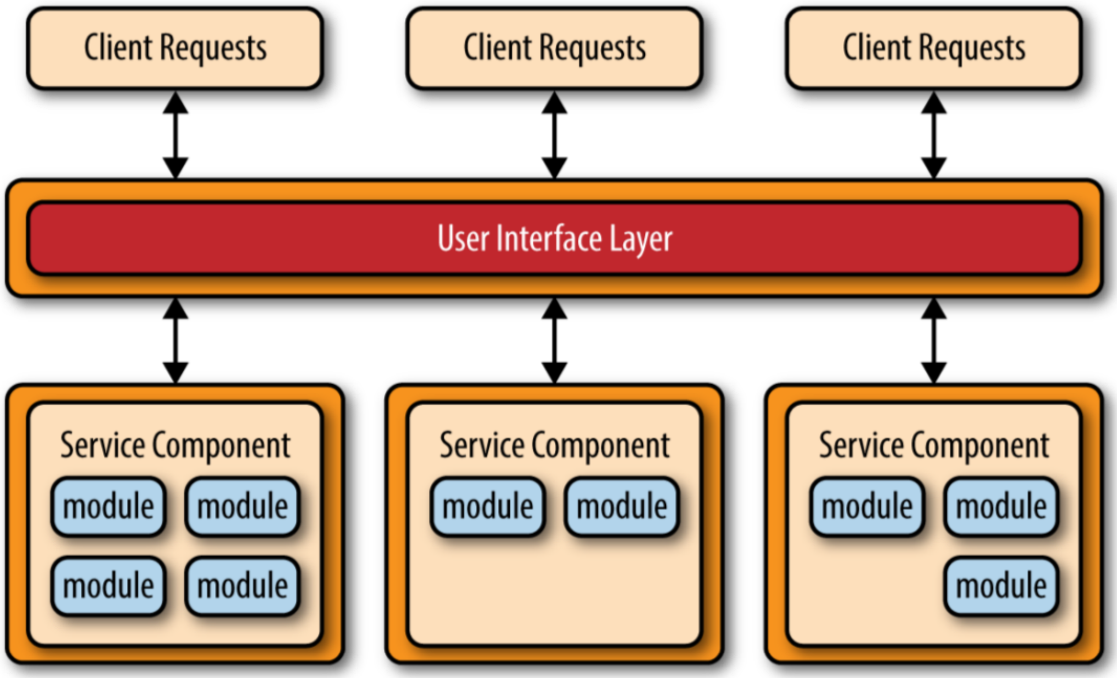


Figure 1.2: Basic Microservices architecture pattern

1.2.7 Architecture evaluations

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. In the case of our application so far the business layer and persistence layer are combined into a single business layer (a controller), particularly when the persistence logic is embedded within the business layer components. Thus our small application has only three layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. The presentation layer is responsible for handling all user interface and browser communication logic, whereas as the business layer is responsible for executing specific business rules associated with the request and retrieving of information. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesnt need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. The business layer doesnt need to be concerned about how to format patient data for display on a screen it needs to get the data from the database layer, perform business logic against the data (e.g., finding the diagnostic history of certain patients), and pass that information up to the presentation layer.

One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic. This type of component classification makes it easy to build effective roles and responsibility models into our architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

1.2.8 Rationale for key decisions

The layered architecture pattern is a solid general-purpose pattern, making it a good starting point for most applications, particularly when you are not sure what architecture pattern is best suited for your application. However, there are a couple of things to consider from an architecture standpoint when choosing this pattern.

The first thing to watch out for is what is known as the architecture sinkhole anti-pattern. This anti-pattern describes the situation where requests flow through multiple layers of the architecture as simple pass-through processing with little or no logic performed within each layer. Every layered architecture will have at least some scenarios that fall into the architecture sinkhole anti-pattern. The key, however, is to analyze the percentage of requests that fall into this category. If one finds a majority of requests are simple pass-through processing, you might want to consider making some of the architecture layers open (meaning requests are allowed to bypass this open layer and go directly to the layer below it), keeping in mind that it will be more difficult to control change due to the lack of layer isolation.

Another consideration with the layered architecture pattern is that it tends to lend itself toward monolithic applications, even if you split the presentation layer and business layers into separate deploy-able units. While this may not be a concern for some applications, it does pose some potential issues in terms of deployment, general robustness and reliability, performance, and scalability.

Chapter 2

Stakeholders and concerns

This chapter is intended to identify and outline the various stakeholders and concerns associated with the product. It provides a traceability matrix to determine who key choices in the architecture will affect.

2.1 Stakeholders

The stakeholders are divided into two main categories below. These two categories are divided based on the activity of the stake holders namely using the product and providing the product.

- **Customers and end users** The application will interact with people in the medical field, such as GPs and specialists. Patients are also secondary users in that they generate the information.
  - *Medical practitioners*
  - *Patients*
- **Systems architects and developers** This section includes people involved in the development, deployment and continuance of the system. We (Level Seven Crew) currently consider ourselves as the development team.
  - *Organization providing the system*
  - *System administrator*

  - *Maintenance team*
  - *Development team*

2.2 Concerns

This section aims at answering critical questions as an aid to identifying the concerns as listed in 2.3.1.

- **What are/is the purpose(s) of the system-of-interest?**
  Providing (often sensitive) information about patients to doctors allowing them to make well-informed diagnoses/prescriptions etc.
- **What is the suitability of the architecture for achieving the system-of-interest's purpose(s)?**
  The layered architecture is suitable in that it allows easy maintenance for things such as updating business rules, new information requirements etc.
- **How feasible is it to construct and deploy the system-of-interest?**
  Given the time allowance, the open source tools at our disposal and the guidance of Dr. Terence van Zyl, constructing the system is entirely feasible. The deployment would involve paying a hosting service which we consider outside the scope of the course. The application will be hosted locally during development while firebase is hosted for free by google. We are still, however, mindful of providing a system that can be easily deployed.
- **What are the potential risks and impacts of the system-of-interest to its stakeholders throughout its life cycle?**
  There are multiple legal and social implications of a system like this. Most of these are related to security and as such can be remedied with consideration for this. The lists below outline a few of the risk factors involved in the system.
  - **Social** Some of these may fall out of scope of the system as it is exceedingly difficult to write software
    - Unauthorized entities accessing sensitive information on patients.
    - Doctors mis-using the system for personal reasons. (eg accessing information on a spouse without permission)
    - Doctors sharing information with people who the patient did not authorize.

  - **Legal** These relate to net neutrality laws. This is a key concern in the development of this system, as we may find laws regarding medical practices clashing with net neutrality laws.
    * Users need to have access to the data stored about them.
    * Users need to be able to update data stored about them.
    * Users need to be able to remove data stored about them.
    * Users need to be able to determine who views there information.

- **How is the system-of-interest to be maintained and evolved?** The system needs to be flexible in order to adapt to ever-changing laws regarding net neutrality and privacy. The medical field is also rapidly changing with new diagnostics and treatments being invented all the time, the system needs to be able to provide facilities which would provide user-value should the need arrive. This relates to changing business values and the decision to use a layered architecture model.

2.3 Concern–Stakeholder Trace-ability

2.3.1 List of concerns:

- **Concern 1** - Security
  Security is a major concern as the information stored is highly sensitive. This refers mainly to cyber attacks with the aim of stealing information.

- **Concern 2** - Doctor patient confidentiality
  Doctor patient confidentiality ties into security but is the more specific case of the doctor's ability to share patient profiles.

- **Concern 3** - Future growth and expansion
  The system needs to be able to grow in multiple directions e.g. New functionalities as well as new patient information.

- **Concern 4** - Doctors mis-using the information
  Doctors should not be allowed to look up patients' information where the patient hasn't approved.

- **Concern 5** - Network latency and loading times
  The doctor should not have to wait exorbitant amounts of time for loading, consults generally last maximum 15 minutes.

- **Concern 6** - Ease of use
  The layout should be logical and easy to navigate.

- **Concern 7** - Deployment
  Deployment needs to be done in a way that provides access to multiple areas in South Africa to reduce instances of missing information.
- **Concern 8** - Changing business rules
  Ties in to Future growth and expansion.
- **Concern 9** - Debugging
  The product needs to be able to be debugged with a minimal amount of effort.
- **Concern 10** - Net neutrality
  Users need to be able to access, update and remove their information at will.

2.3.2 List of Stakeholders

- **Stakeholder 1** - Medical practitioners
- **Stakeholder 2** - Patients
- **Stakeholder 3** - Organization providing the system
- **Stakeholder 4** - System administrator
- **Stakeholder 5** - Maintenance team
- **Stakeholder 6** - Development team

Table 2.1: Showing association of stakeholders to concerns in an AD

	S.holder 1	S.holder 2	S.holder 3	S.holder 4	S.holder 5	S.holder 6
Concern 1	**X**	**X**	**X**			
Concern 2		**X**				
Concern 3			**X**	**X**	**X**	
Concern 4		**X**	**X**			
Concern 5	**X**	**X**				
Concern 6	**X**	**X**				
Concern 7	**X**	**X**	**X**	**X**		
Concern 8			**X**	**X**	**X**	
Concern 9					**X**	**X**
Concern 10	**X**	**X**	**X**			

Chapter 3

Viewpoints

Definition: A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views.

3.1 Architecture Model

A model is a complete, basic, and simplified description of software architecture which is composed of multiple views from a particular perspective or viewpoint.

A view is a representation of an entire system from the perspective of a related set of concerns. It is used to describe the system from the viewpoint of different stakeholders such as end-users, developers, project managers, and testers.

3.1.1 4+1 Model

The 4+1 View Model was designed by Philippe Kruchten to describe the architecture of a software-intensive system based on the use of multiple and concurrent views. It is a multiple view model that addresses different features and concerns of the system. It standardizes the software design documents and makes the design easy to understand by all stakeholders.

It is an architecture verification method for studying and documenting software architecture design and covers all the aspects of software architecture for all stakeholders. It provides four essential views:

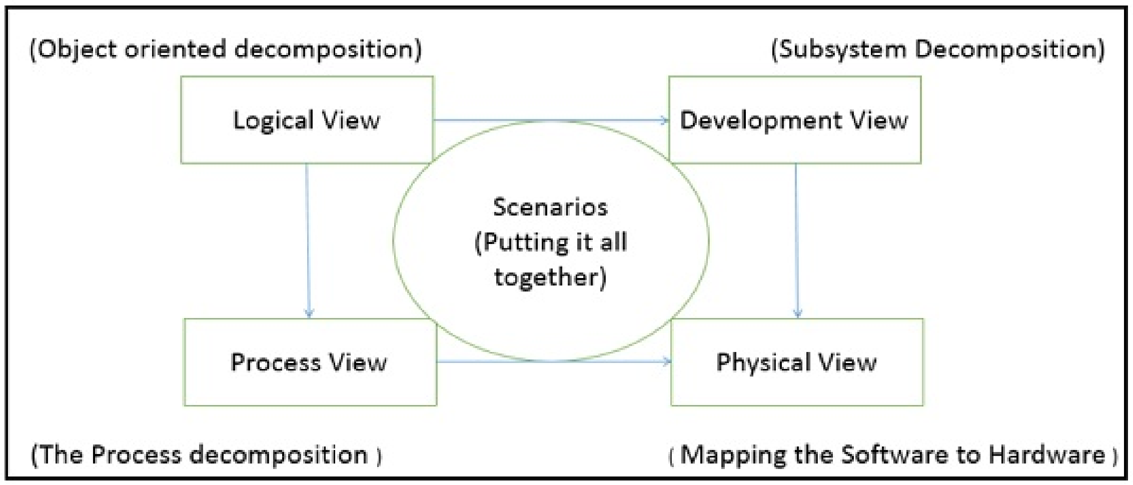

(Object oriented decomposition)
(Subsystem Decomposition)
Logical View
Development View
Scenarios (Putting it all together)
Process View
Physical View
(The Process decomposition)
(Mapping the Software to Hardware)

Figure 3.1: 4+1 Model

- **The logical view or conceptual view** - It describes the object model of the design.
- **The process view** - It describes the activities of the system, captures the concurrency and synchronization aspects of the design.
- **The physical view** - It describes the mapping of software onto hardware and reflects its distributed aspect.
- **The development view** - It describes the static organization or structure of the software in its development of environment.

This view model can be extended by adding one more view called scenario view or use case view for end-users or customers of software systems. It is coherent with other four views and are utilized to illustrate the architecture serving as plus one view, (4+1) view model. The following figure describes the software architecture using five concurrent views (4+1) model.

3.2 Logical Viewpoint

3.2.1 Overview

Describes the systems functional elements, their responsibilities, interfaces, and primary interactions. A Functional view is the cornerstone of most ADs and is often the first part of the description that stakeholders try to read. It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on. It also has a significant impact on the systems quality properties such as its ability to change, its ability to be secured, and its run-time performance.

3.2.2 Concerns and stakeholders

Concerns

- Functional capabilities.
- External interfaces.
- Internal structure.
- Functional design philosophy.

Stakeholders

- All stakeholders.

3.2.3 Model

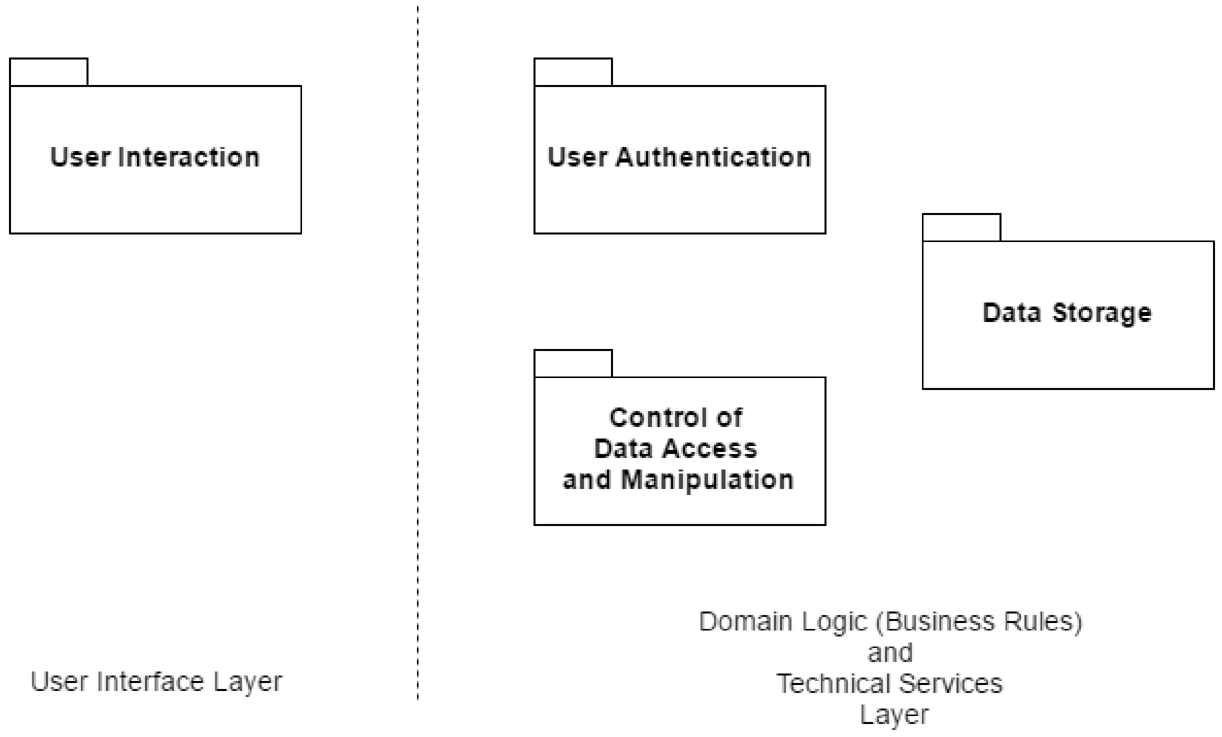
- Functional structure model.

3.2.4 Known issues with view

- Poorly defined interfaces.
- Poorly understood responsibilities.
- Infrastructure modeled as functional elements.
- Overloaded view.
- Diagrams without element definitions.
- Difficulty in reconciling the needs of multiple stakeholders.
- Wrong level of detail.
- ‘God elements
- Too many dependencies.

3.2.5 Application Overview

Figure 3.2: Functionality Diagram


User Interaction
User Authentication
Data Storage
Control of Data Access and Manipulation
User Interface Layer
Domain Logic (Business Rules) and Technical Services Layer

Describes the functionality of the Healfolio Web Application system, that uses a 2-tier architecture.

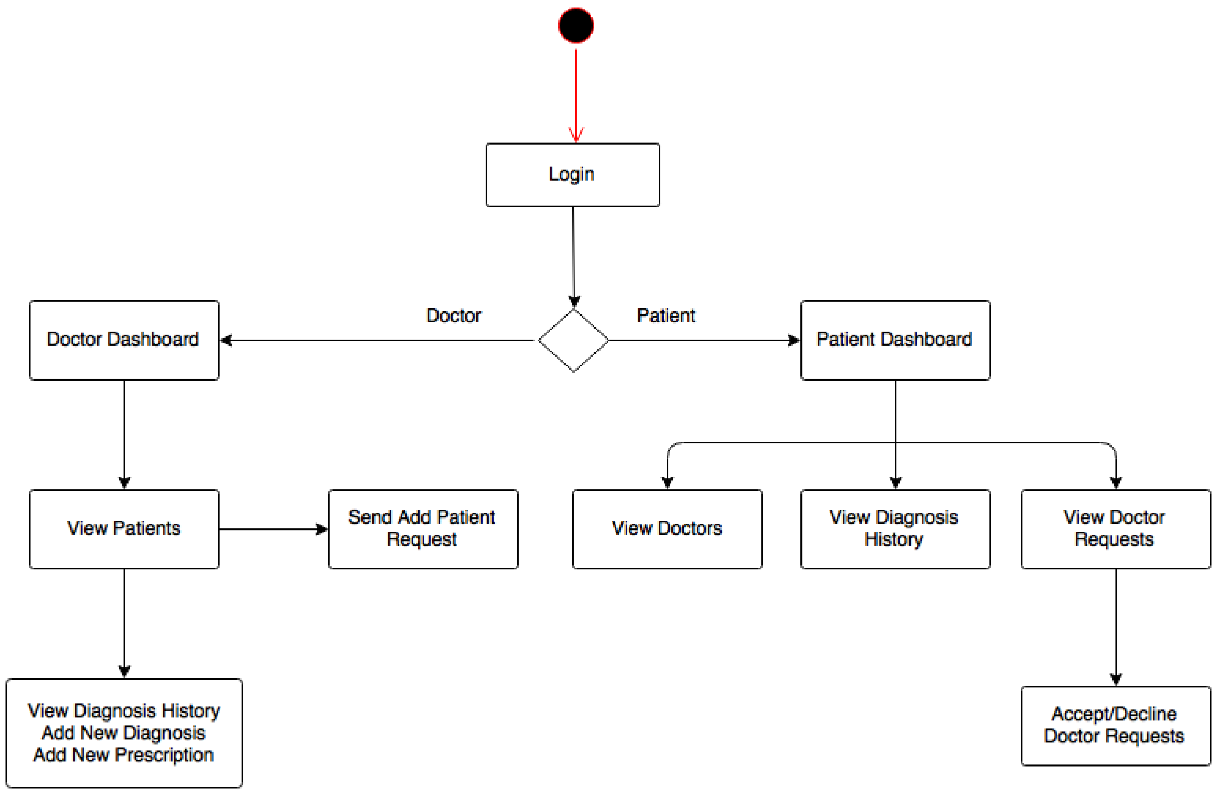
- **Layer 1**

  This layer is responsible for the users interaction with the application. The users access to this layer is implemented by use of compatible web browsers pre-installed on the users device such as a personal computer or mobile device.

- **Layer 2**

  This layer is responsible for enforcing the business rules/logic of the application as well as storing all the data concerned with the Application. This layer is hosted on a cloud services platform. This layer is responsible for enforcing simultaneous access of the data by users. It provides various application analytics tools as well as backup and restoration of the application in case of failure.

Figure 3.3: Logical Model



3.3 Development Viewpoint

3.3.1 Overview

Describes the architecture that supports the software development process. Development views communicate the aspects of the architecture of interest to those stakeholders involved in building, testing, maintaining, and enhancing the system.

3.3.2 Concerns and stakeholders

Concerns

- Module organization.
- Common processing.
- Standardization of design.
- Standardization of testing.
- Instrumentation.
- Code-line organization.

Stakeholders

- Production engineers, software developers and testers.

3.3.3 Model

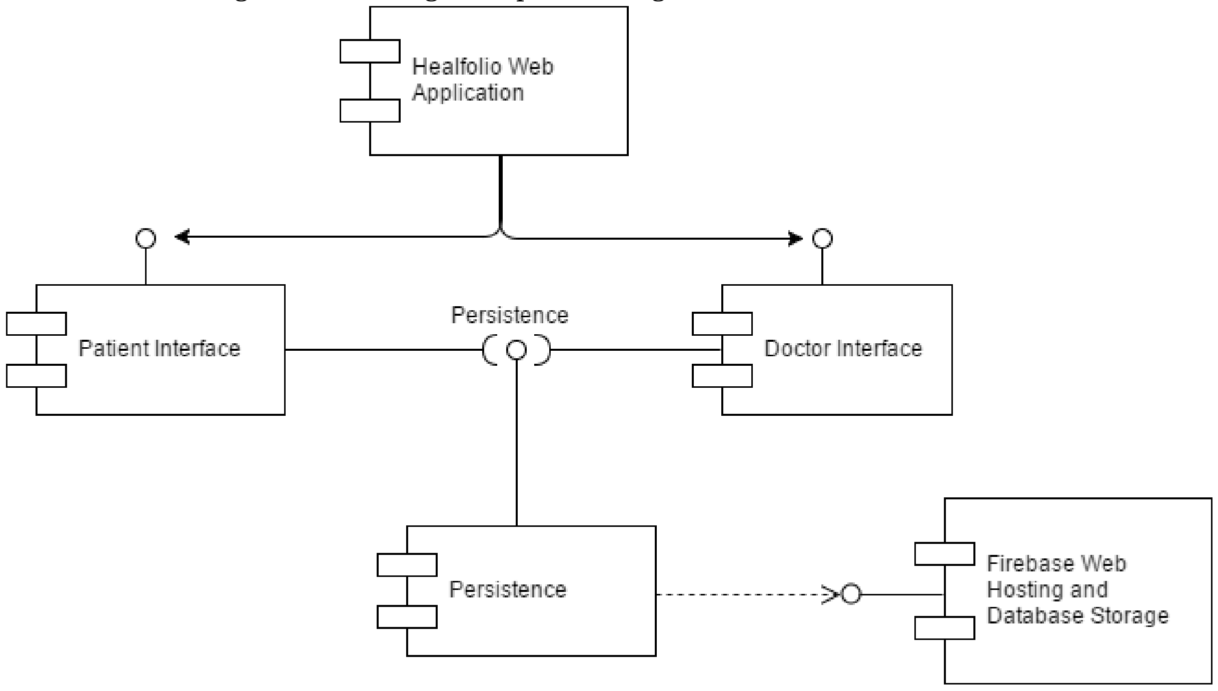
- Module structure models.
- Common design models.
- Code-line models.

3.3.4 Known issues with view

- Too much detail.
- Overburdening the AD.
- Uneven focus.
- Lack of developer focus.
- Lack of precision.
- Problems with the specified environment.

3.3.5 Application Overview

Figure 3.4: Package Component Diagram


Healfolio Web Application
Patient Interface
Persistence
Doctor Interface
Persistence
Firebase Web Hosting and Database Storage

Components

- **Healfolio Web Application**

  This is the application that runs all the business logic. It is stored on the web hosting platform and the served to the user via a web browser. It handles the logic to serve the next two packages as shown in the diagram; the doctor interface and patient interface.

- **Doctor Interface**

  This Interface is viewed by an authenticated doctor operating a web browser. It allows the doctor to request addition of patients to their list of patients. The doctor will then be able to view the patients records such as personal information, medical diagnoses and prescriptions. The doctor will also be able to create new or follow up diagnosis as well as assign prescriptions.

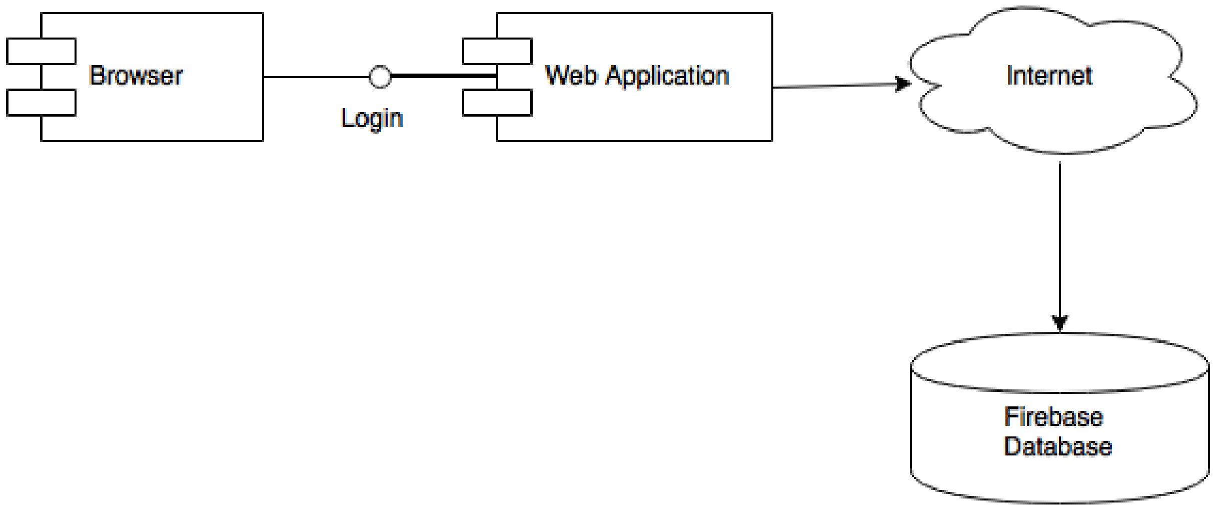
- **Patient Interface**

  This interface is viewed by an authenticated patient operating a web browser. It allows the patient to accept or decline request from doctors to view and manipulate their medical records. It further allows the patient to view their records stored in the database such as diagnosis and assigned prescriptions.

- **Firebase Web Hosting and Database Storage**

  Hosting and Storage platform that stores all the user accounts and doctor-patient interaction data generated and required for the application.

Figure 3.5: Deployment Model



3.4 Physical Viewpoint

3.4.1 Overview

Describes the environment into which the system will be deployed, including capturing the dependencies the system has on its run-time environment. This view captures the hardware environment that your system needs (primarily the processing nodes, network interconnections, and disk storage facilities required), the technical environment requirements for each element, and the mapping of the software elements to the run-time environment that will execute them.

3.4.2 Concerns and stakeholders

Concerns

- Run-time platform required.
- Specification and quantity of hardware or hosting required.
- Third-party software requirements.
- Technology compatibility.
- Network requirements.
- Network capacity required.
- Physical constraints.

Stakeholders

- System administrators, developers, testers, communicators, and assessors.

3.4.3 Model

- Run-time platform models.
- Network models.
- Technology dependency models.
- Inter-model relationships.

3.4.4 Known issues with view

- Unclear or inaccurate dependencies.
- Unproven technology.
- Unsuitable or missing service-level agreements.
- Lack of specialist technical knowledge.
- Late consideration of the deployment environment.
- Ignoring inter-site complexities.
- Inappropriate headroom provision.
- Not specifying a disaster recovery environment.

3.4.5 Application Overview

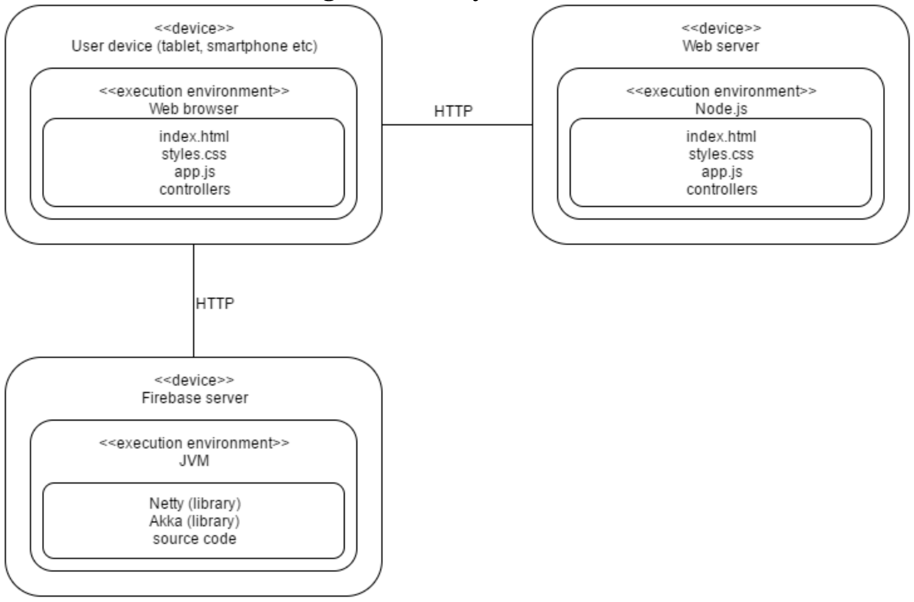
The Application and Database are both hosted using the Firebase Cloud Services and Storage Platform. It stores the application logic and interface templates that enable the user to interact with the application. It also stores the database document and handles the applications user authentication as a service. The application was created using the AngularFire JavaScript framework for the business logic and html/css was used to create the user interface templates. It is a Model-View- Controller hybrid architecture that provides real time database synchronization and further allows for effortless 3-way data binding among the different components. The User will be able to interact with the application using any modern web browser that provides support for HTML/CSS and JavaScript.

The following list summarizes the plug in script dependencies that have been incorporated into the application:

- JQuery
  - https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.0/jquery.min.js
- Bootstrap JavaScript
  - https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.0.1/js/bootstrap.min.js
- Firebase App
  - https://www.gstatic.com/firebasejs/3.1.0/firebase-app.js
- Firebase Auth
  - https://www.gstatic.com/firebasejs/3.1.0/firebase-auth.js
- Firebase Database
  - https://www.gstatic.com/firebasejs/3.1.0/firebase-database.js
- Firebase
  - https://www.gstatic.com/firebasejs/3.2.1/firebase.js
- Angular JavaScript
  - https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular.min.js
- Angular Route
  - https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular-route.min.js
- Angular Animate
  - https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular-animate.min.js
- AngularFire
  - https://cdn.firebase.com/libs/angularfire/2.0.1/angularfire.min.js
- Font Awesome
  - https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font- awesome.min.css

- Bootstrap Cascading Style Sheets (CSS)
  - https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.0.1/js/bootstrap.min.js

Figure 3.6: Physical Model



3.5 Process Viewpoint

3.5.1 Overview

Describes how the system will be operated, administered, and supported when it is running in its production environment. For all but the simplest systems, installing, managing, and operating the system is a significant task that must be considered and planned at design time. The aim of the Operational viewpoint is to identify system-wide strategies for addressing the operational concerns of the systems stakeholders and to identify solutions that address these.

3.5.2 Concerns and stakeholders

Concerns

- Installation and upgrade.

- Functional migration.
- Data migration.
- Operational monitoring and control.
- Alerting.
- Configuration management.
- Performance monitoring.
- Support.
- Backup and restore.
- Operation in third-party environments.

Stakeholders

- System administrators, developers, testers, communicators, and assessors.

3.5.3 Model

- Installation models.
- Migration models.
- Configuration management models.
- Administration models.
- Support models.

3.5.4 Known issues with view

- Lack of engagement with the operational staff.
- Lack of back-out planning.
- Lack of migration planning.
- Insufficient migration window.
- Missing management tools.
- Production environment constraints.

- Lack of integration into the production environment
- Inadequate backup models.
- Unsuitable alerting.

3.5.5 Application Overview

Figure 3.7: Patient/Doctor Verification Activity Diagram

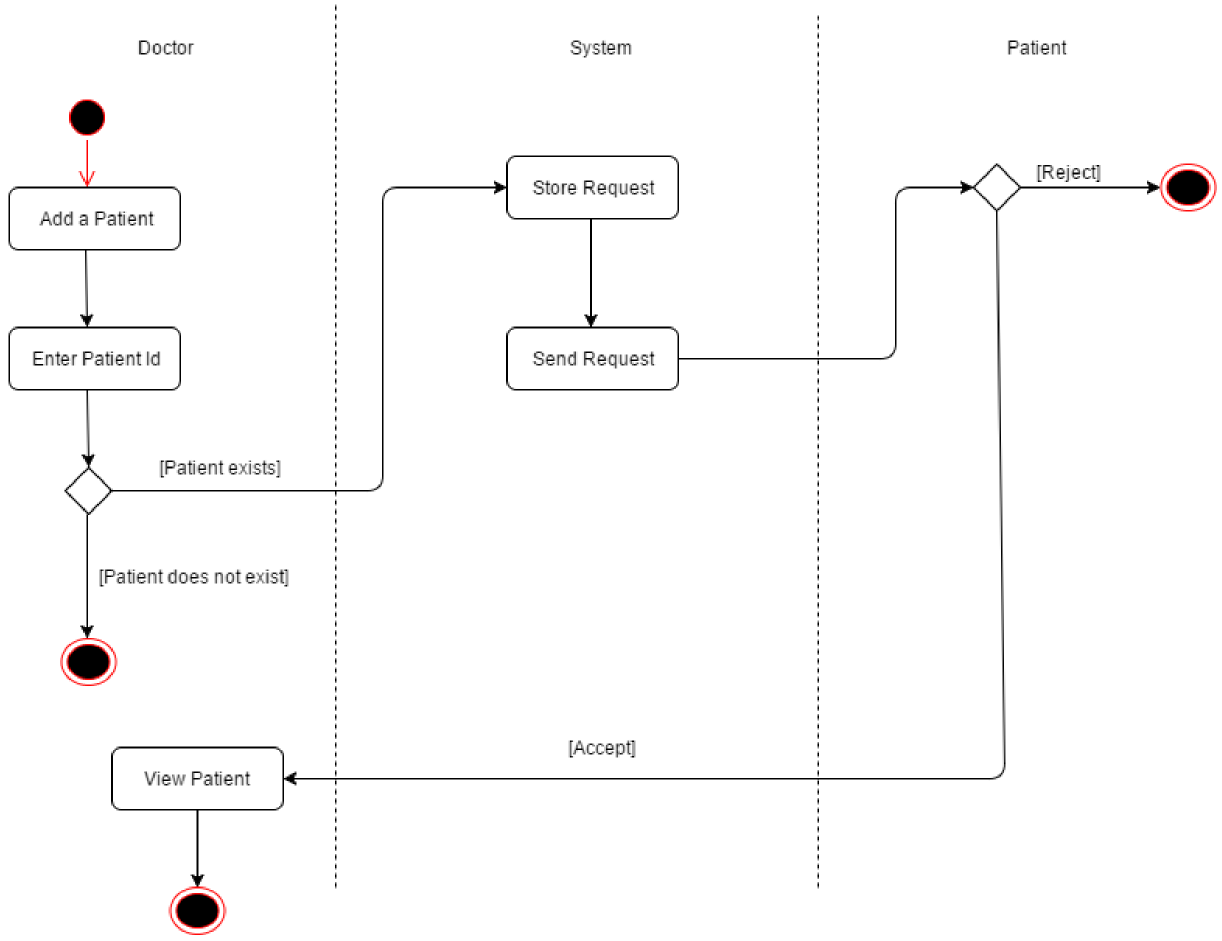


Figure 3.8: Doctor adding Diagnosis Activity Diagram

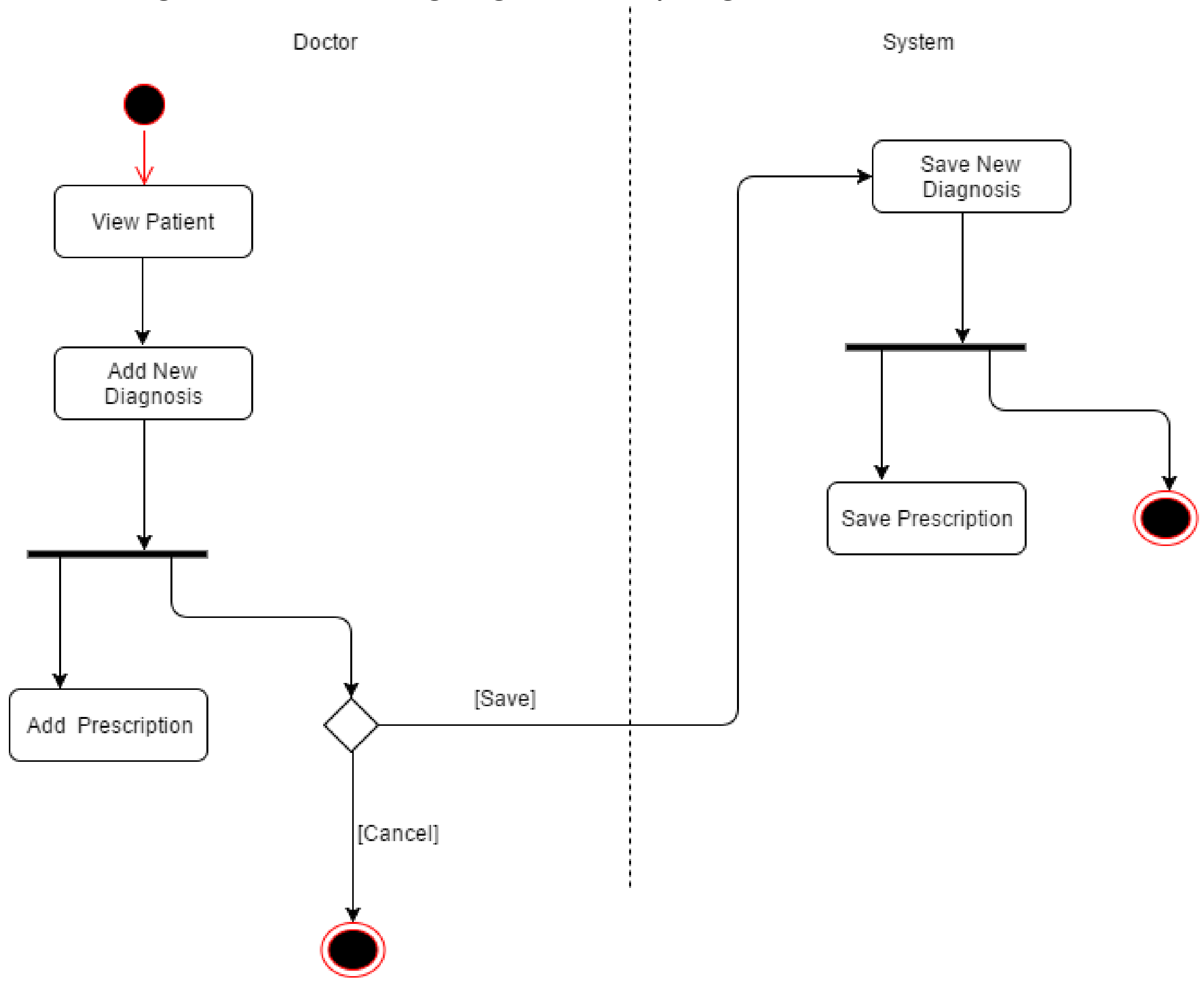


Figure 3.9: Doctor adding Follow-Up Activity Diagram

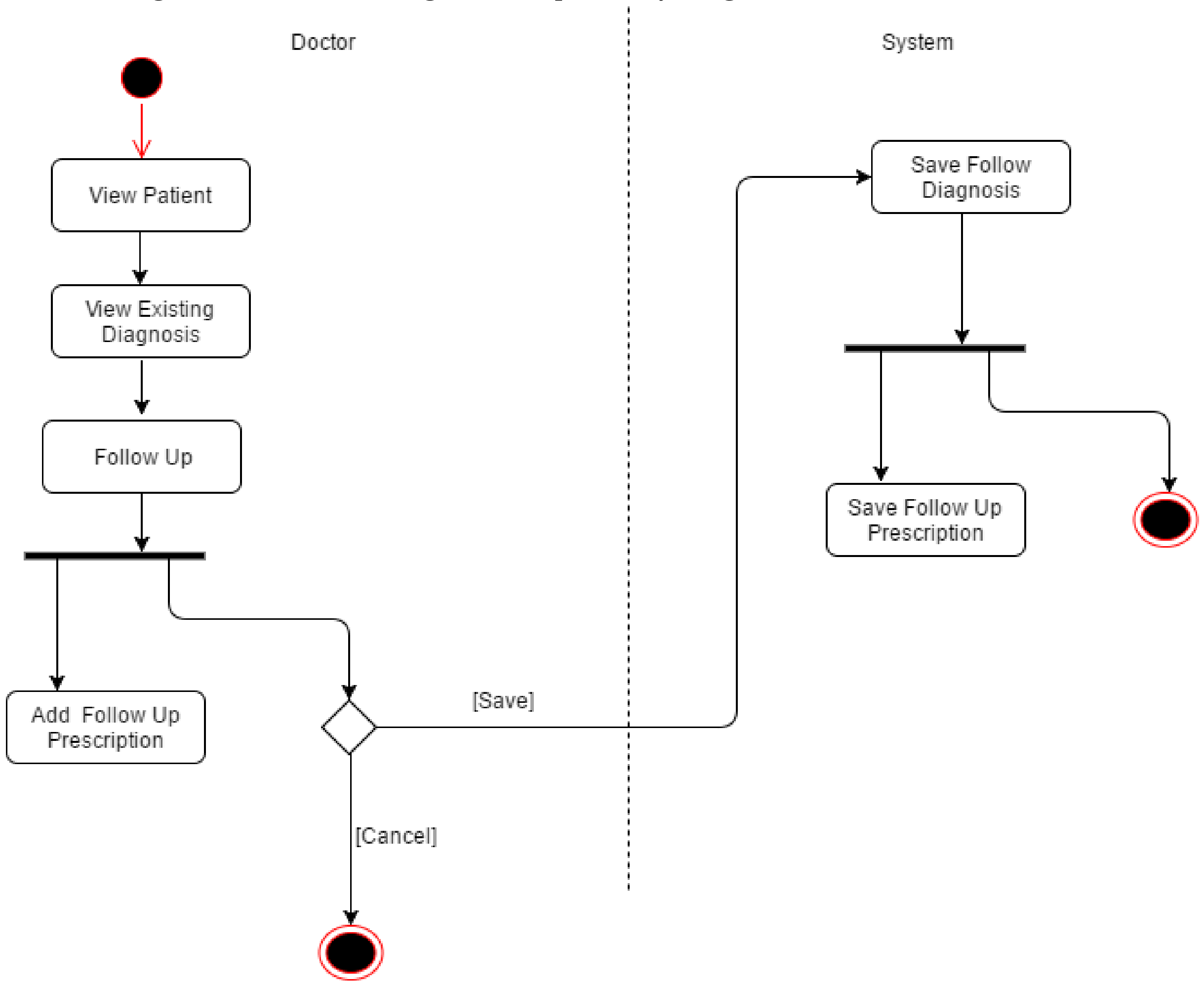
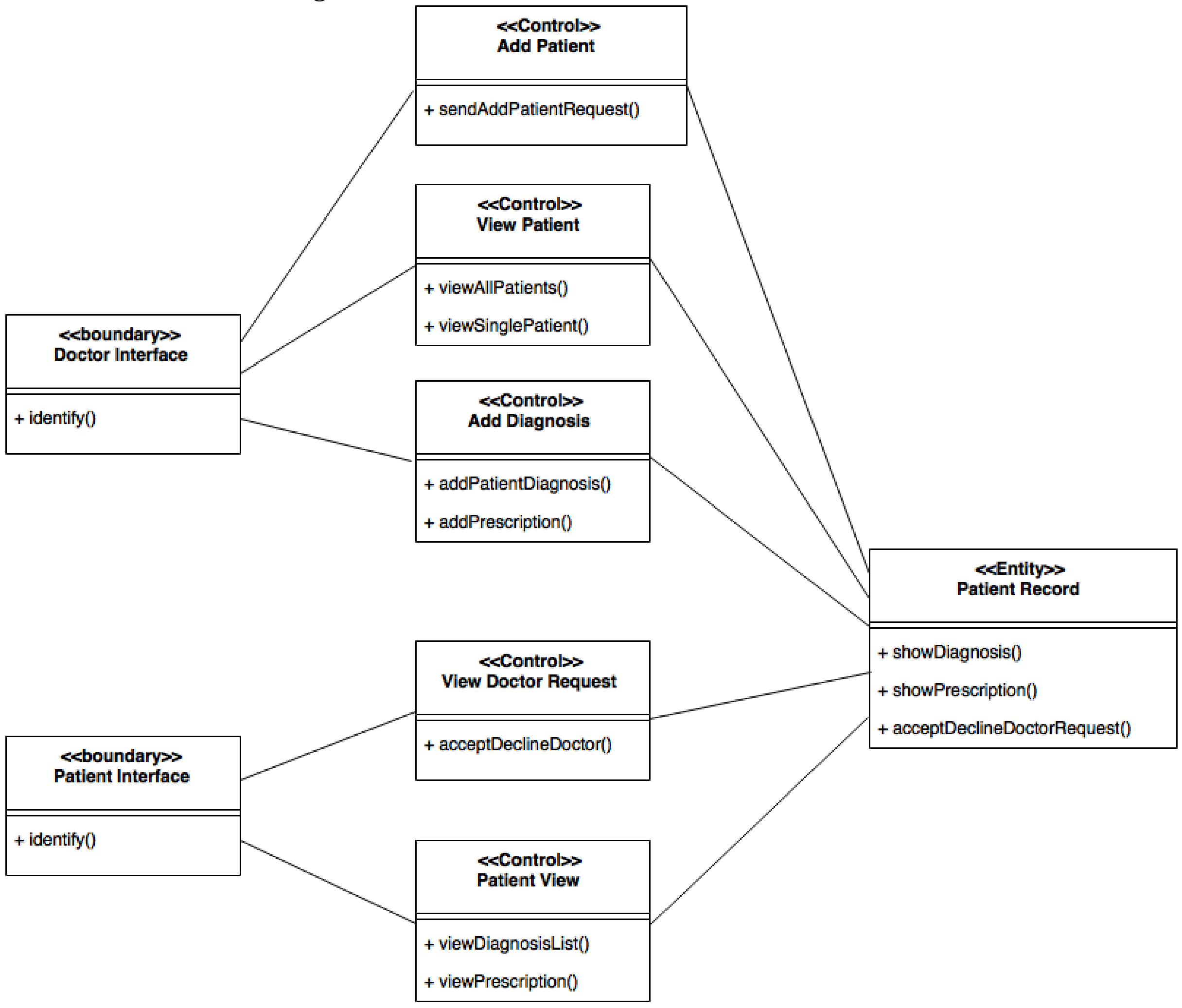


Figure 3.10: Process Model


<<Control>> Add Patient
+ sendAddPatientRequest()
<<Control>> View Patient
+ viewAllPatients()
+ viewSinglePatient()
<<boundary>> Doctor Interface
+ identify()
<<Control>> Add Diagnosis
+ addPatientDiagnosis()
+ addPrescription()
<<Entity>> Patient Record
+ showDiagnosis()
+ showPrescription()
+ acceptDeclineDoctorRequest()
<<Control>> View Doctor Request
+ acceptDeclineDoctor()
<<boundary>> Patient Interface
+ identify()
<<Control>> Patient View
+ viewDiagnosisList()
+ viewPrescription()