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

1 Introduction **2**

1.1 Identifying information 2

1.2 Supplementary information 2

1.2.1 Date of issue and status 2

1.2.2 Authors 3

1.2.3 Reviewers 3

1.2.4 Scope 3

1.2.5 Context 3

1.2.6 Overview 4

1.2.7 Architecture evaluations 5

1.2.8 Rationale for key decisions 6

2 Stakeholders and concerns **8**

2.1 Stakeholders 8

2.2 Concerns 9

2.3 Concern–Stakeholder Trace-ability 9

2.3.1 List of concerns: 9

2.3.2 List of Stakeholders 10

3 Viewpoints **11**

3.1 Architecture Model 11
3.1.1 4+1 Model 11
3.2 Logical Viewpoint 12
3.2.1 Overview 12
3.2.2 Concerns and stakeholders 13
3.2.3 Model 13
3.2.4 Known issues with view 13
3.3 Development Viewpoint 14
3.3.1 Overview 14
3.3.2 Concerns and stakeholders 14
3.3.3 Model 15
3.3.4 Known issues with view 15
3.4 Physical Viewpoint 16
3.4.1 Overview 16
3.4.2 Concerns and stakeholders 16
3.4.3 Model 17
3.4.4 Known issues with view 17
3.5 Process Viewpoint 18
3.5.1 Overview 18
3.5.2 Concerns and stakeholders 19
3.5.3 Model 19
3.5.4 Known issues with view 19

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.

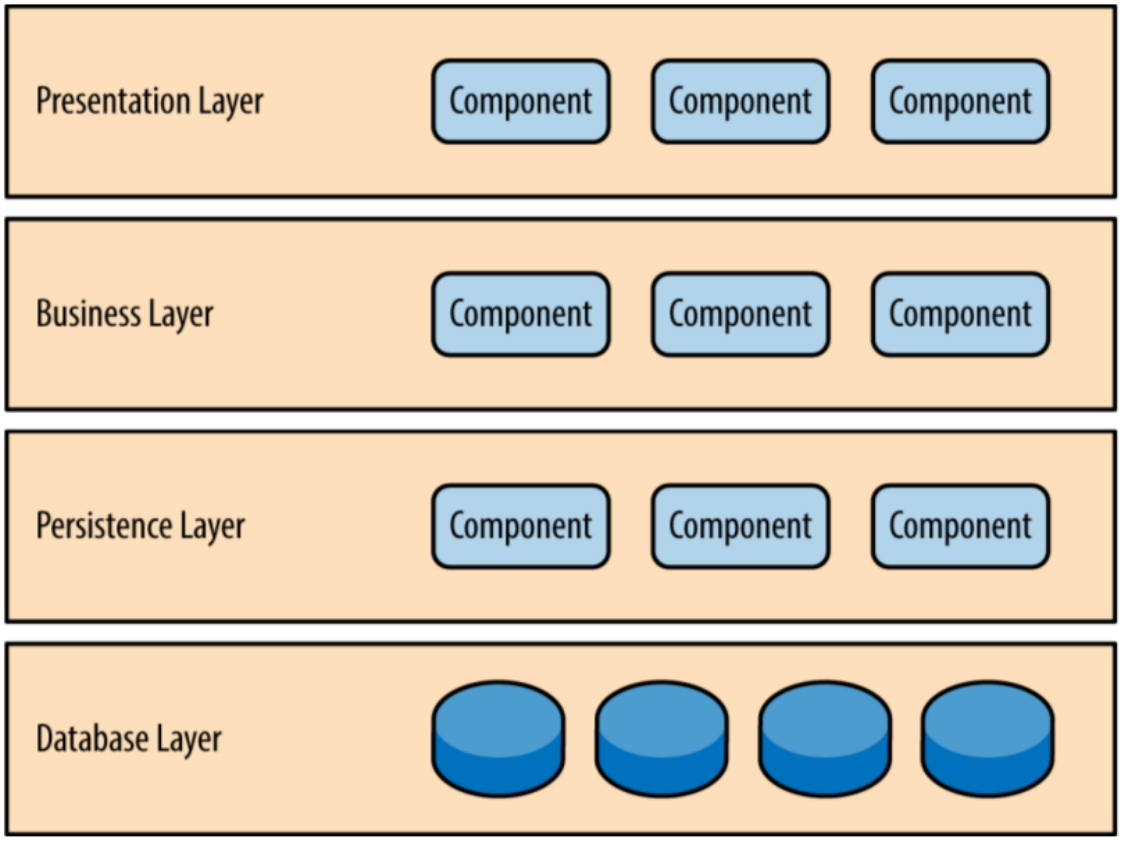


Figure 1.1: Layered Architecture Pattern

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 applica-

tion 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).

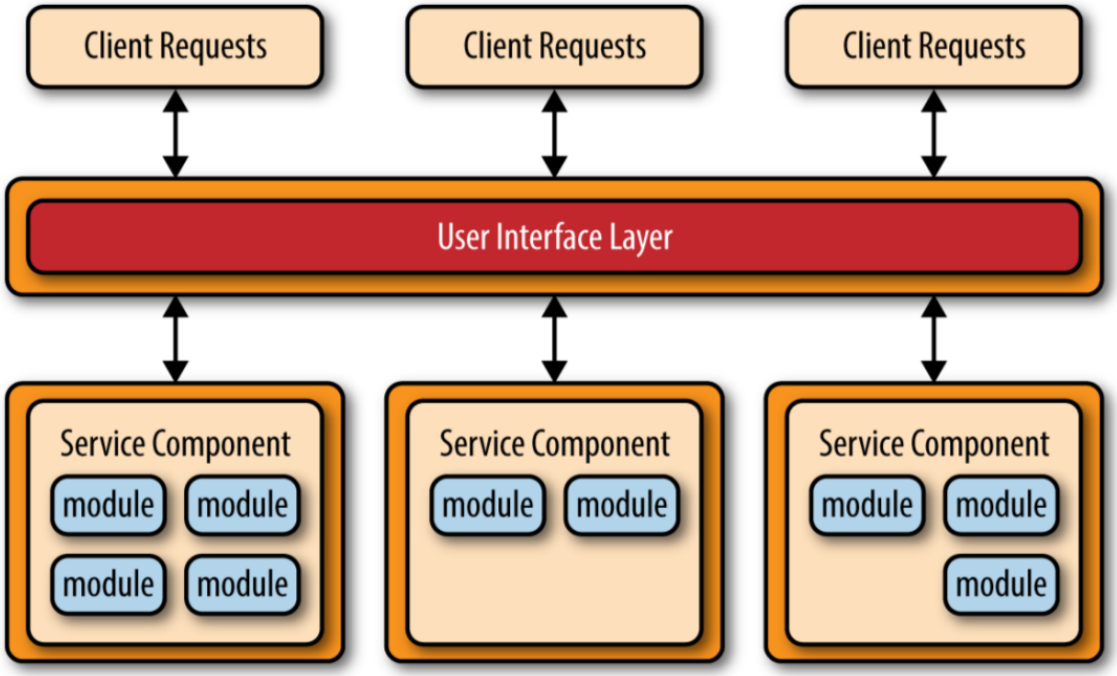


Figure 1.2: Basic Microservices architecture pattern

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 com-

bined 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 contains information items for stakeholders of the architecture, the stakeholders' concerns for that architecture, and the traceability of concerns to stakeholders. See also: ISO/IEC/IEEE 42010, 5.3

2.1 Stakeholders

A stakeholder is an individual, team, or organization with interests in, or concerns related to, the system-to-be. Generally, the system-to-be has several types of stakeholders: customers, end users, business analysts, systems architects and developers, testing and quality assurance engineers, project managers, the future maintenance organization, owners of other systems that will interact with the system-to-be.

- **Customers and end users**
  - The application will interact with people is the medical field, private practices, where doctors would be able to use relevant information from not only their past experience but a collective of as many practitioners that use the application. Where previous they had a to look a hard copy of patient and similar cases that would might be overlooked or very time consuming will now be able to call various report to there device from a vast database. The doctors and receptionists will be the users and operators of the application.

- **Systems architects and developers**
  - The team Level Seven Crew the acquirers, owners, developers, builders and testers of the application HealFolio.
  - Dr. Terence van Zyl is a person of interest and will be evaluating as well as consult on the progress the team makes.

2.2 Concerns

- **What are the purpose(s) of the system-of-interest?**
  Making referencing of information that would be otherwise too much of an ordeal, therefore streamlining the service giving by medical practitioners.
- **What is the suitability of the architecture for achieving the system-of-interest's purpose(s)?**
  Layered Architecture
- **How feasible is it to construct and deploy the system-of-interest?**
  With the opens source tools available, the direction of the course instructor and the time allocated makes tackling this project possible.
- **What are the potential risks and impacts of the system-of-interest to its stakeholders throughout its life cycle?**
  The sharing or leakage of confidential information can be an issue that has to addressed continuously thought-out the development and improvement of the application. The stakeholders might feel sharing diagnosis would be violating doctor patient privileges, and this needs to be pressed that security is key the application's development.
- **How is the system-of-interest to be maintained and evolved?**
  With an eye on future the application will start of as a layered architecture and keeping in mind to migrate the application toward microservices architecture with future improvement and growth.

2.3 Concern–Stakeholder Trace-ability

2.3.1 List of concerns:

1. **Concern 1** - Sharing or leakage of confidential information
2. **Concern 2** - Doctor patient confidentiality.
3. **Concern 3** - Future growth and expansion.

2.3.2 List of Stakeholders

1. **Stakeholder 1** - Medical Practitioners
2. **Stakeholder 2** - Team Level Seven Crew
3. **Stakeholder 3** - Lecturer

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

	Stakeholder 1	Stakeholder 2	Stakeholder 3	
Concern 1	**x**	**x**	-	
Concern 2	**x**	**x**	**x**	
Concern 3			**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 architec-

ture for all stakeholders. It provides four essential views:

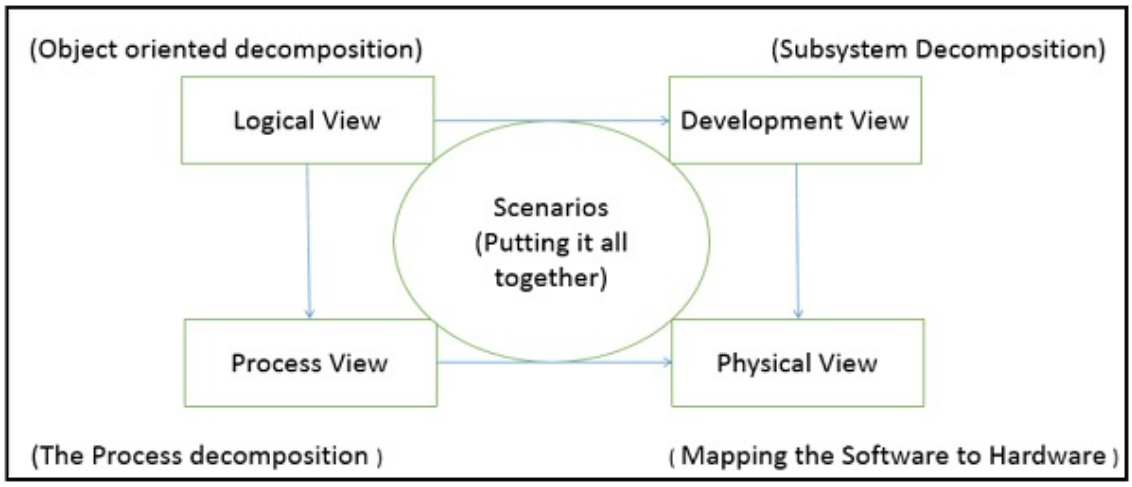


Figure 3.1: 4+1 Model

- **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.

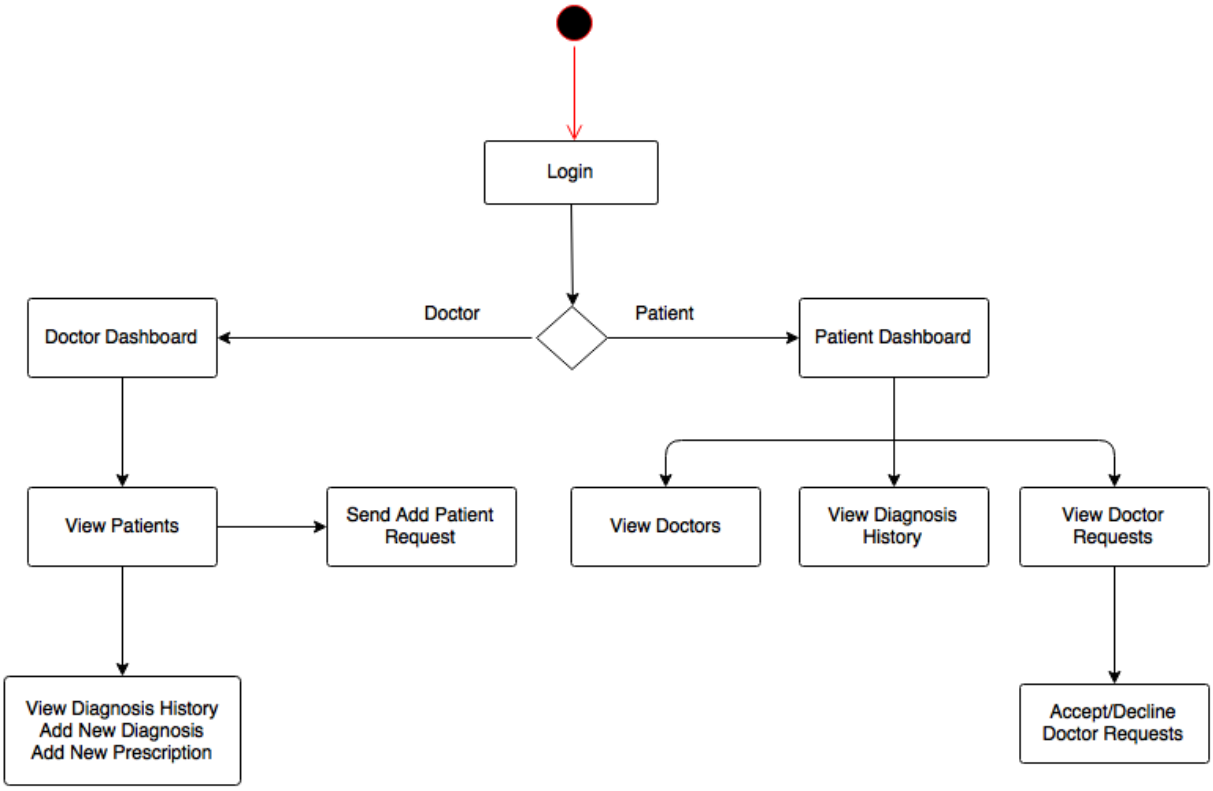
3.2.3 Model

- 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.

Figure 3.2: 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.

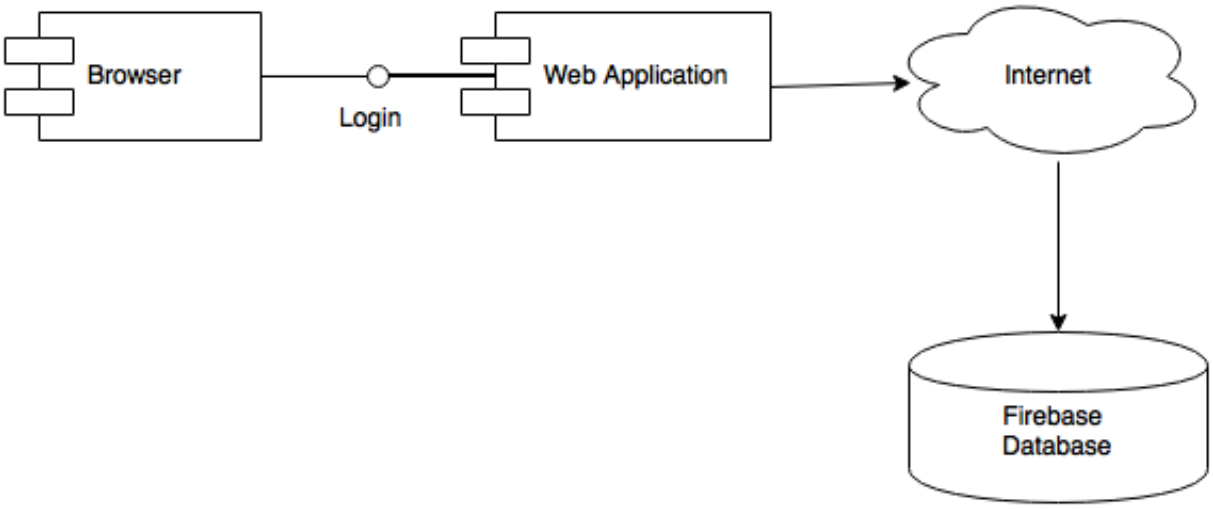
3.3.3 Model

- 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.

Figure 3.3: 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.

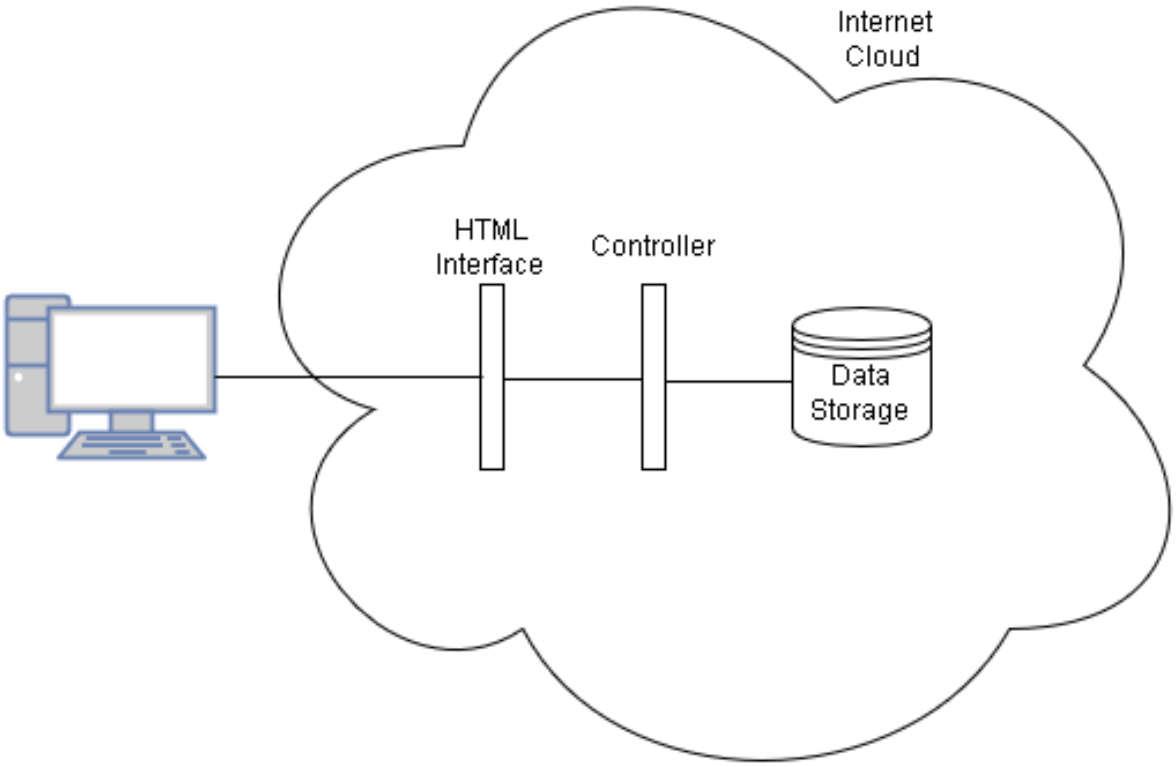
3.4.3 Model

- 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.

Figure 3.4: 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.

Figure 3.5: Process Model