Architecture Description of The Layered architecture for Optimum Hair Finder

Contents

Introduction

The introduction section of the Software Architecture and Design document provides the main architectural pattern of the system, its design concepts and implementation and supplementary information on the system. Additional information; like an overall view of the system and evaluations of the system, to support the main architectural model will also is included in this section of the document.

1.1 Identifying information

The Layered Architecture

The Layered or Tiered software architecture is organized in horizontal layers, each layer performing a specific role within the system. The layered architecture has the standard four layers in the **Optimum Hair Finder** software system. Each layer in the architecture has its own tasks and responsibilities which communicate with each other in an ordered approach. Each layer is an independent component that contributes the total functionality of the systems' design. The following are the layers in the proposed system:

A. Presentation layer

Handles the user interface and additional communications with users; in the proposed system this will be the web application.

B. Business layer

Also called the domain logic layer, is responsible for the organisation of the system: executing specific user requests that are sent into the system.

C. Persistence layer

Manages the interaction to the database; converts requests and puts them in proper form in order to send these requests to the database.

D. Database layer

This layer deals with all stored data on database and accessibility for user requests through to the system.

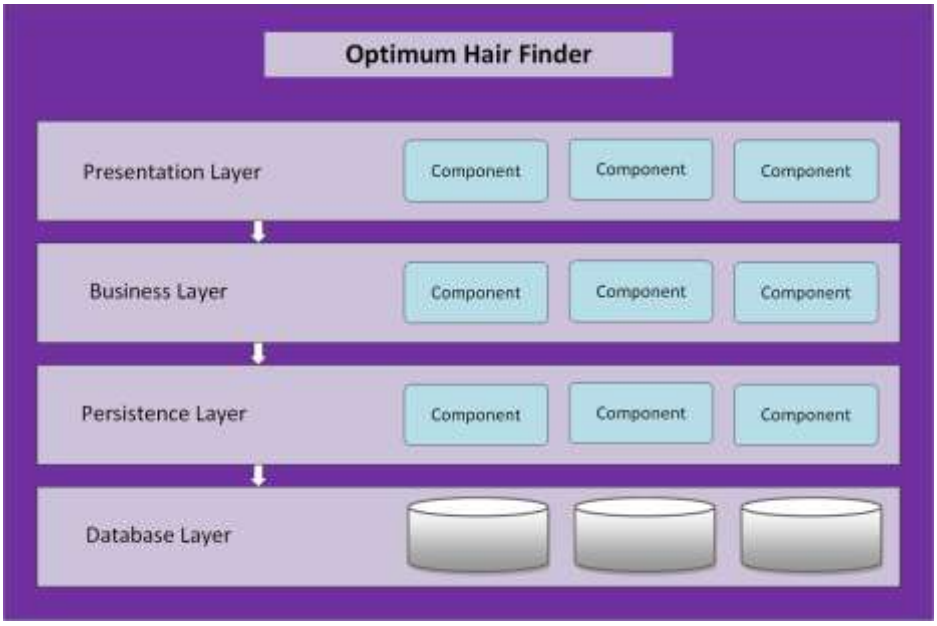


Figure 1: Layered Architecture Pattern

1.2 Supplementary information

Authors: Nova Team
Raesetje Bonjo Sefala 844165
Laura Bokgoshi 792647
Siphamandla Mzobe 850456
Tshepo Molefe 705457
Keku Mashego 671000

Reviewers: Raesetje Bonjo Sefala, Laura Bokgoshi, Siphamandla Mzobe, Tshepo Molefe, Keku Mashego

Approving authority: Raesetje Bonjo Sefala

Issuing organization: Nova

Date of issue: First version 15 August, 2016
Status: Work in progress

Change history: First version 15 August, 2016: view salon, search for salon, search for hairstyle, search for prices, find proximity
Second version 12 September, 2016: view salon, search for salon, search for hairstyle, search for prices, find proximity, book appointment, view hairstyles

Version control information: https://github.com/Makgoale-Bokgoshi/Optimum-Hair-Finder
First version 15 August, 2016
Second version 12 September, 2016
Third version 26 September, 2016
Final version: October, 2016

Configuration management information: No updates yet.

Scope: Allows users to search hairstyle of choice on system and receive results of salons that offer that hairstyle, prices, location and distance to salons. System also allows user accounts, so users can login for a personal feel and view salon details and book appointments with salons.

Context: This document contains the software architecture description of the Optimum Hair Finder system created by Nova. The descriptions are detailed using architecture design information and UML interaction diagrams. This document is prepared using ISO/IEC/IEEE 42010, Systems and software engineering—The Architecture Description Template.

Summary: This is document is produced to read and provides the architectural design and diagrams to outline the Optimum Hair Finder software system.

Glossary:

UML: Unified Modeling Language
SAD: Software Architecture Design
AD: Architecture Design
ADL: Architecture Description Language

1.2 Other information

1.3.1 Overview

This document describes the architecture pattern for the Optimum Hair Finder software system. It will include, in detail, the essential points or viewpoints of the system, stakeholders and concerns, descriptions of those viewpoints and accompanying diagrams to illustrate the interactions and models in the points. It will also include supplementary information, consistencies and inconsistencies in the system, as well the decisions and rationale of the software.

1.3.2 Architecture evaluations

Overall agility
Rating: Low
Making changes to system takes time even with the layered system we have since system design and implementations is a big system.

Ease of deployment
Rating: Low
Changes made to the system affect its ability of it to be easily deployed especially the system is a big one. Change to one layer will not cause entire system to be updated only the section that deals with that change. So, all components related to the one that's being changed may need to be re-evaluated and redeployed first before integration into system and redeployment of the entire system.

Testability
Rating: High
Each layer can be tested individually after each modification or before the deployment of the whole system.

Performance
Rating: Low
The interactivity in the use of layers causes time problems as some requests do not need passing through certain layers but still have to and so wasting time and therefore affecting the performance of the entire system.

Scalability
Rating: Low
The system will be hard to scale since it consists of multiple layers which are all linked somehow and follow certain pattern of interactivity. The only way to scale the system would be to split the layers up then scale each layer before integration again, which is not efficient.

Ease of development
Rating: High
The use of layers makes development of the software manageable because the different layers and components within will be familiar to developers.

Complexity
Rating: Low
The use of layers makes development of the software complex when it comes to the flow or organisation of the system. Certain requests are simple but because of the set structure of the design there is no way to pre-program certain methods for simpler request. Each step and verification will need to be checked in order for the system to run.

Loose Coupling
Rating: Low
The ordered approach of the interaction between layers causes problems when it comes to alteration of the software. If modifications are made to a layer, components within that layer are affected but not the other layers. Direct access to other layer is not allowed because of the many and methodical interdependencies between layers that cannot be randomly interrupted.

1.3.3 Rationale for key decisions

The system needed for Optimum Hair Finder was one that would allow specific **queries** access to a **database**. Depending on the type of query, certain extra steps would try to narrow down the request to the simplest form to be used to access the database. The system architecture design needed was one that would meet these requirements and the layered architecture came closest to meeting the requirements.

The layered architecture design offers a choice of a number of layers and a database layer. Since our system uses queries and queries are in a simple form with no room for misunderstandings, the use of a few layers would allows us to execute queries and produce results from the database. Our architecture design uses the four standard layers. The presentation, business logic, persistence and database layers together work well for our system.

The layered system in The Optimum Hair Finder software system is chosen because of its overall efficiency. The biggest feature in layered architectures is its **separation of concerns**. This feature allows components within a specific layer to interact with requests or instructions that concerns that layer only. This architectural model allows us to deal with the presentation, business logic, persistence and database of our system all individually. The organisation of these components of the system is more manageable and proficient with relation to the Optimum Hair Finder software system.

Another big feature in layered architectures is its **layer isolation** feature. The layers in a layered architecture are closed. This means that as a request moves from one layer to another layer, it must go through the layer below it to get to the next layer below that one. This feature allows the modification of details or implementation within a specific layer which will not affect the rest of the system, however, this continuous interactivity causes the change of such an architectural design to be very difficult and expensive.

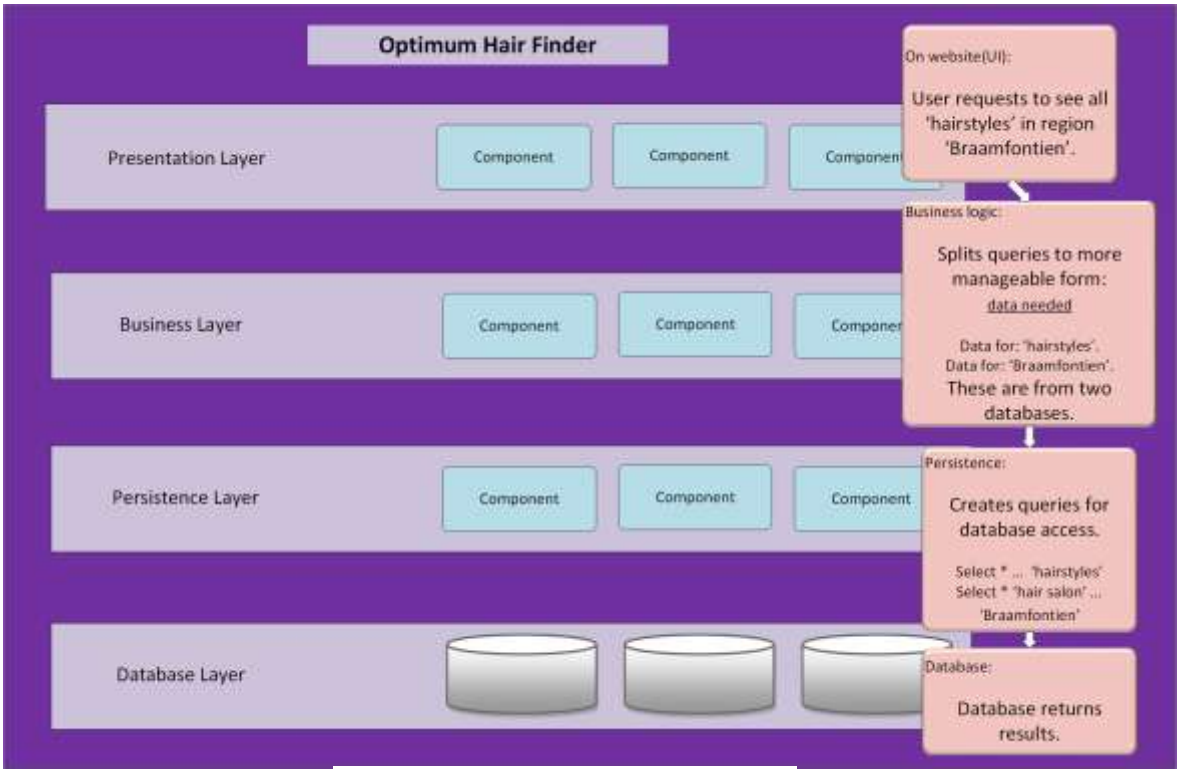


Figure 2: Example of use of Layered Architecture in system

Stakeholders and concerns

This chapter contains the key roles of the stakeholders of the architecture, the stakeholders' concerns for that architecture, and the traceability of concerns to stakeholders.

2.1 Stakeholders

The stakeholder for the architecture is the Nova Team, the users of our system and Dr. Terence van Zyl the lecturer.

Users: general public
Operators: Nova Team
Acquirers: None
Owners: Nova
Suppliers: None
Developers: Nova Team
Builders: Nova Team
Maintainers: Nova Team

2.2 Concerns

	Concerns
Concern 1	**What are the purpose(s) of the Optimum Hair Finder?**
Concern 2	**What is the suitability of the architecture for achieving the Optimum Hair Finder's purpose(s)?**
Concern 3	**How feasible is it to construct and deploy the Optimum Hair Finder?**
Concern 4	**How is the Optimum Hair Finder to be maintained and evolved?**
Concern 5	**How will the development of the Optimum Hair Finder be evaluated?**
Concern 6	**How will Optimum Hair Finder impact its' users?**
Concern 7	**With relation to the architecture design pattern(Layered architecture); how will the sinkhole anti-pattern affect the overall performance of the system?**
Concern 8	**How will Optimum Hair Finder handle the suggestion that Layered systems appear monolithic?**
Concern 9	**Will Optimum Hair Finder be able to keep design pattern consistent?**
Concern 10	**Are the layers of the system partitioned correctly?**
Concern 11	**Are the layers of the system defined explicitly and are the relations between them logical?**
Concern 12	**Are the system's components functional, logical and do they relate?**

Table 1: Table of concerns

2.3 Concern–Stakeholder Traceability

	Nova Team	Users	Dr. Terrence
Concern 1	-	X	X
Concern 2	X	X	X
Concern 3	X	-	-
Concern 4	X	-	-
Concern 5	X	X	X
Concern 6	X	X	X
Concern 7	X	-	-
Concern 8	X	-	-
Concern 9	X	-	-
Concern 10	X	-	-
Concern 11	X	-	-
Concern 12	X	-	-

Table 2: Table of association of stakeholders to concerns in an Architecture Design

Viewpoints

Functional
Information
Development
Operational

3.1 Functional

3.2 Overview

The functional viewpoint describes the system's functional elements; their responsibilities, interfaces, and interactions. This is the main viewpoint of the system since it is responsible for the systems' functional aspect and will determine the systems' scalability, security and performance.

Key features:

- indicates relationships
- indicates interactions
- indicates dependencies

3.3 Concerns and stakeholders

3.3.1 Concerns

Functional capabilities
identity of external entities and services and data used
nature and characteristics of external entities
identity and responsibilities of external interfaces
nature and characteristics of external interfaces
Functional design philosophy
overall completeness, consistency, and coherence

3.3.2 Typical stakeholders

Developers: Nova Team
Maintainers: Nova Team

3.4 Model kinds

Functional structure model

3.5 Functional structure model

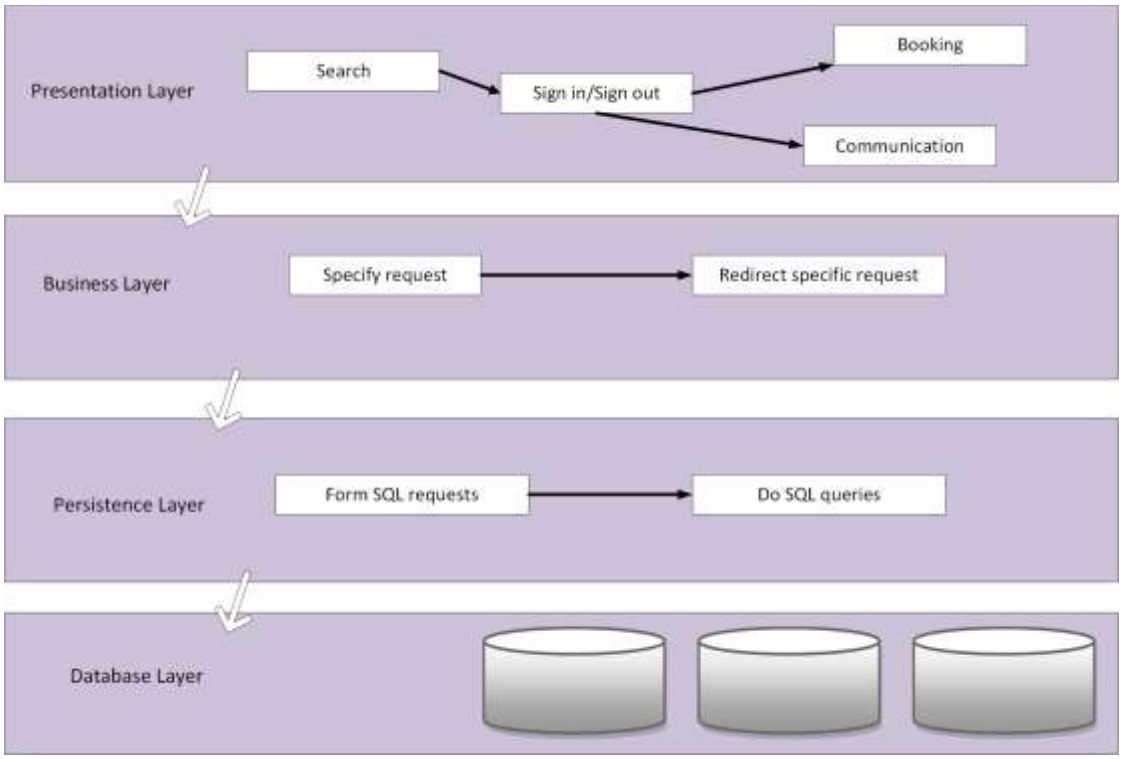


Figure 3: Functional Structure Model

3.5.1 Functional structure model conventions

Languages: PHP, SQL, JavaScript

3.6 Operations on views

Construction methods
Main models and objects are put into the most logical form into the architecture design. In our case the objects of the system should be under correct layer in system and there should be a procedural method that is logical to follow and will work.

Implementation methods
Will have functions in code that handle specific object and object purpose.

4.1 Information

4.2 Overview

This viewpoint describes systems' storage, manipulation, management, and distribution structures. i.e. the administrative aspect of the system.

Key features:

- indicates content and structure management
- indicates ownership, latency and references
- indicates data migration

4.3 Concerns and stakeholders

4.3.1 Concerns

information structure, content and usage
information ownership
enterprise-owned information
volatility of information semantics
information storage models
information flow, consistency and quality
timeliness, latency, and age
archiving and information retention

4.3.2 Typical stakeholders

Users: general public
Operators: Nova Team
Acquirers: None
Owners: Nova
Suppliers: None;
Developers: Nova Team
Builders: Nova Team
Maintainers: Nova Team

4.4 Model kinds

Static information structure model
Information flow model

4.5.1 Static information structure model

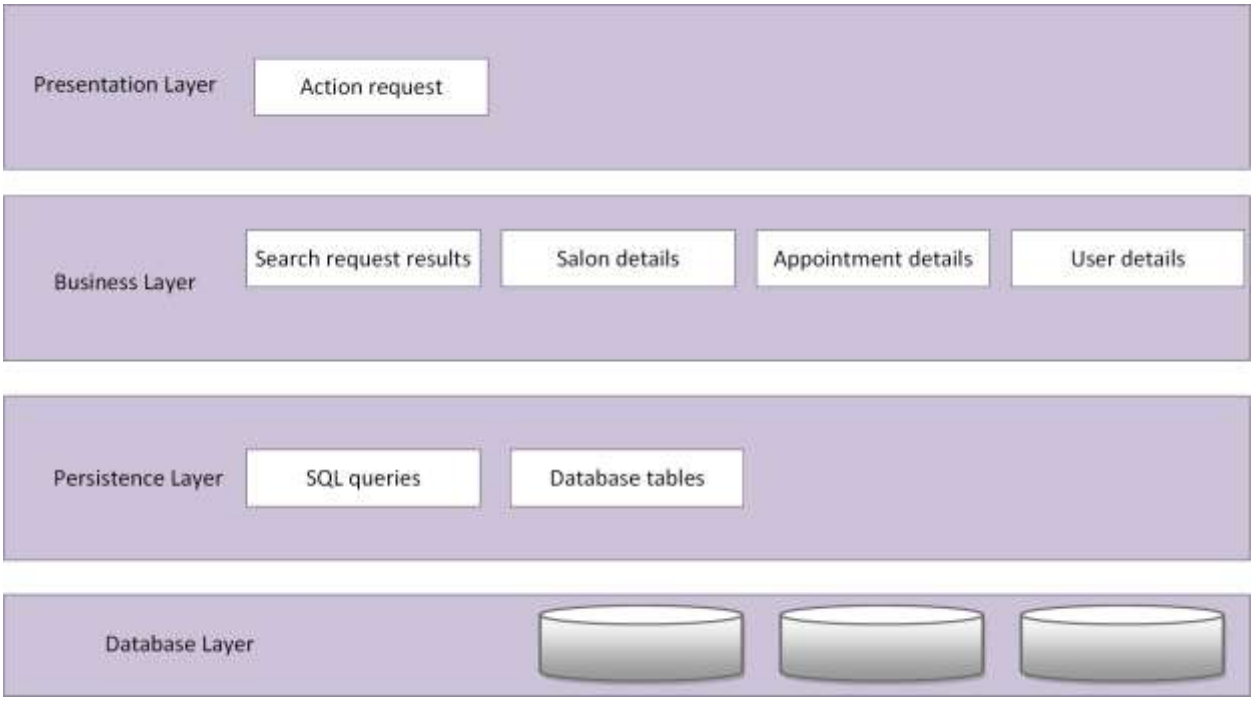


Figure 4: Static Information Structure Model

4.5.1.1 Static information structure model conventions

Languages: Layman terms

4.5.2 Information flow model

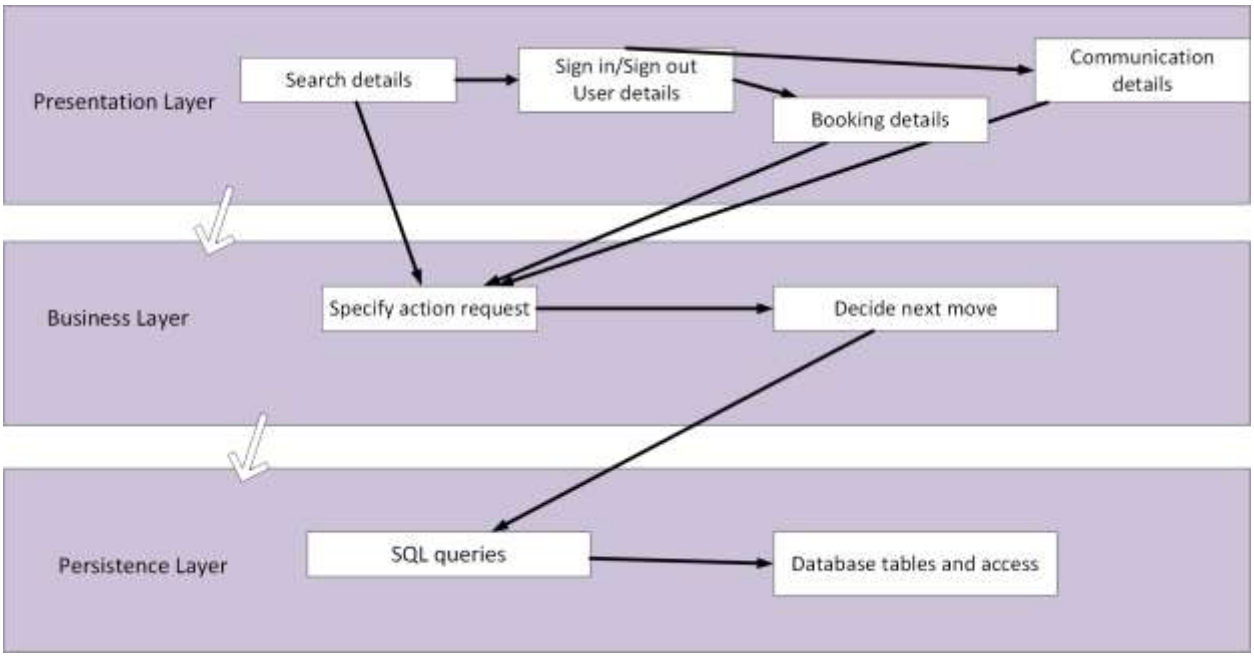


Figure 5: Information Flow Model

4.5.2.1 Information flow model conventions

Languages: Layman terms

4.6 Operations on views

Construction methods
Information aspects and pre-set statements for system are mapped to their respective layers.

Implementation methods
Will be methods within object functions.

5.1 Development

5.2 Overview

The development viewpoint describes the software development process; building the software, testing, maintaining, and enhancing the system.

Key features:

- indicates the code structure and dependencies
- indicates build and configuration management

- indicates system-wide design constraints

5.3 Concerns and stakeholders

5.3.1 Concerns

module organization
common processing
standardization of design
standardization of testing
instrumentation
codeline organization

5.3.2 Typical stakeholders

Developers: Nova Team
Builders: Nova Team
Maintainers: Nova Team

5.4 Model kinds

Module structure model

5.5 Module structure model

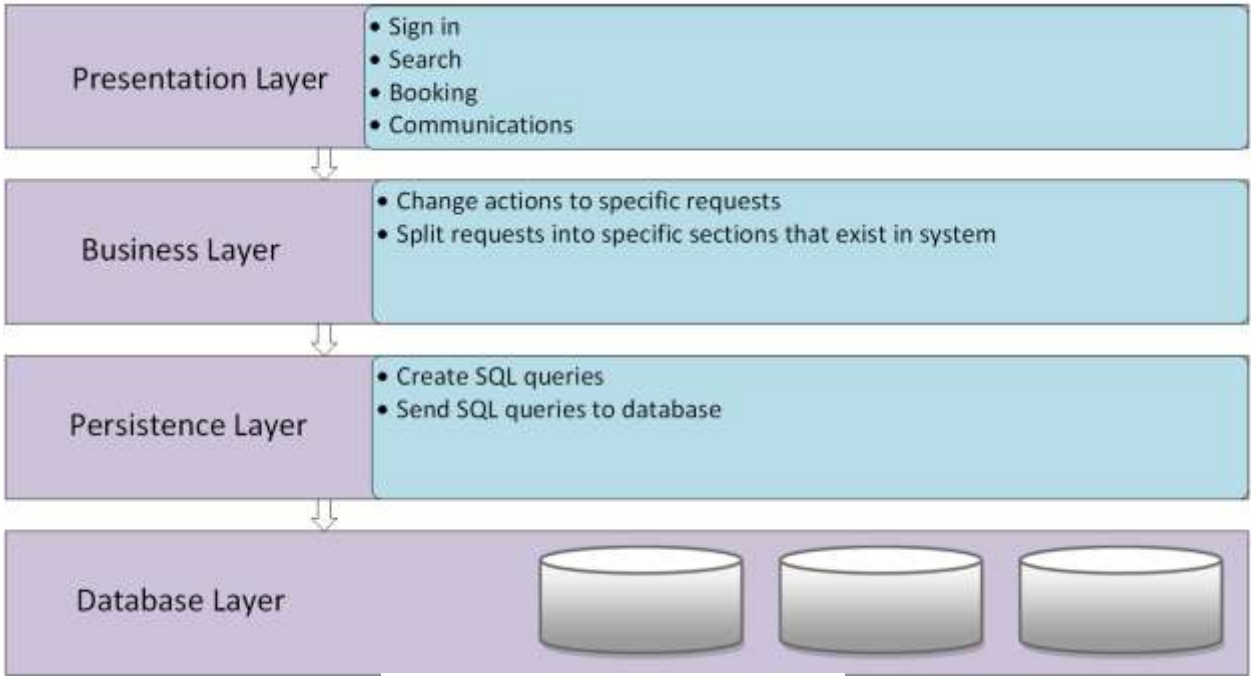


Figure 6: Module Structure Model

5.5.1 Module structure model conventions

Languages: Layman terms, SQL, PHP, JavaScript

5.6 Operations on views

Construction methods
Each module/component of the system will be put in appropriate layer of software architecture. Interactions of components will be taken into considerations so that placement of components makes sense in layer it is in.

Implementation methods
Will be methods within object functions or smaller functions that can be used within the bigger object functins.

6.1 Operational
6.2 Overview

The operational viewpoint describes how the system will be operated, administered, and supported and is significant. The operation of a system must be planned when the system is being designed, this helps identify system-wide strategies and solutions that might arise in early stages.

Key features:

- indicates installation requirements
- indicates management of system structures
- indicates operation the system

6.3 Concerns and stakeholders

6.3.1 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

6.3.2 Typical stakeholders

Operators: Nova Team
Developers: Nova Team
Builders: Nova Team
Maintainers: Nova Team

6.4 Model kinds

Administration model

6.5 Administration model

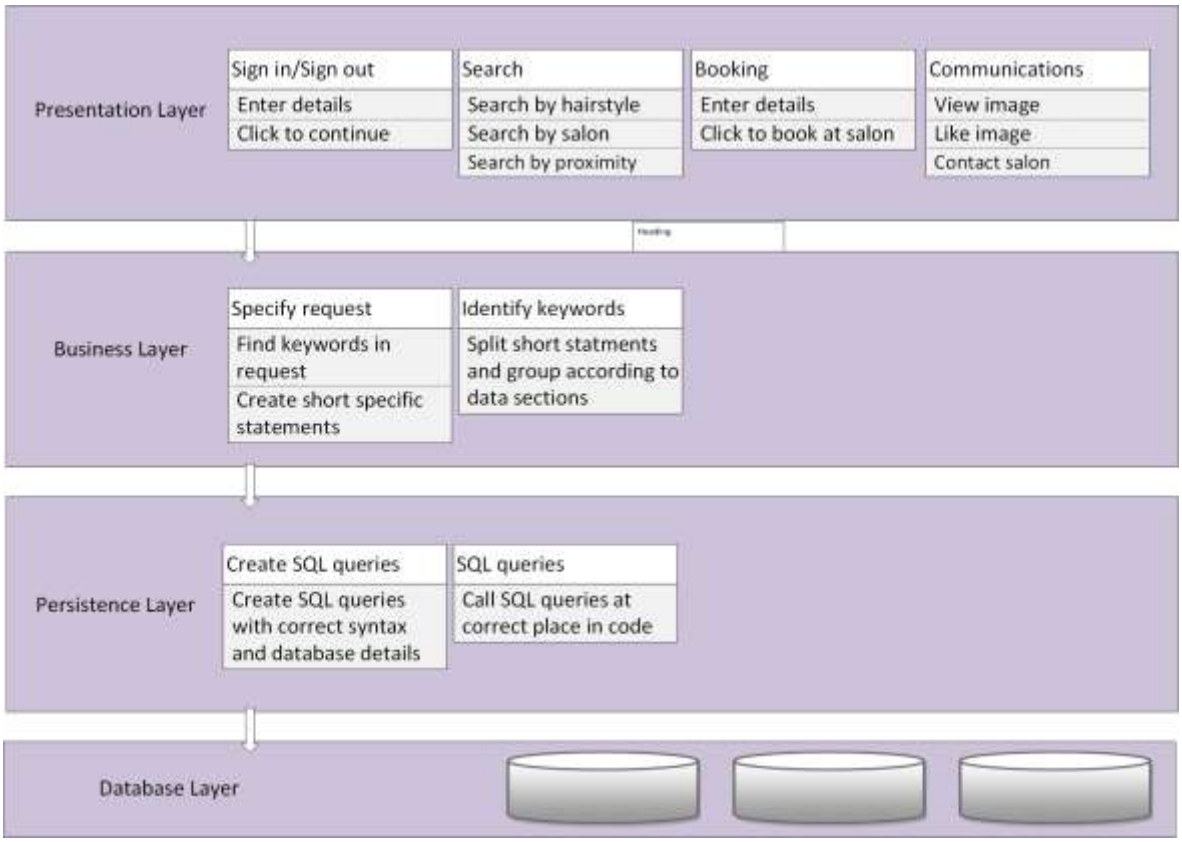


Figure 7: Administration Model

6.5.1 Administration model conventions

Languages: SQL, PHP, JavaScript

6.6 Operations on views

Construction methods
All data used and inputted into system will be logically placed within database. Data flow in processes is logical, follows proper procedure patterns and will give final results when system is being used.

Implementation methods
Information usage is used as variables within code. Storage and variable access in code is logical, used in proper functions and methods and referred to correctly.

Views

Logical
Process
Development
Physical

4.1 View: Logical

The logical view outlines the main services the system provides. It is based on the systems' functional requirements, what the system should provide to the user. This view outlines this system in terms of the objects and classes and their interactions. This outline is not only to show functionality that went into the system but to also show common mechanisms and design elements across the various parts of the system. The logical view is defined by the **functional viewpoint**.

4.1.1 Models

UML Class Diagram

4.1.2 UML Class Diagram

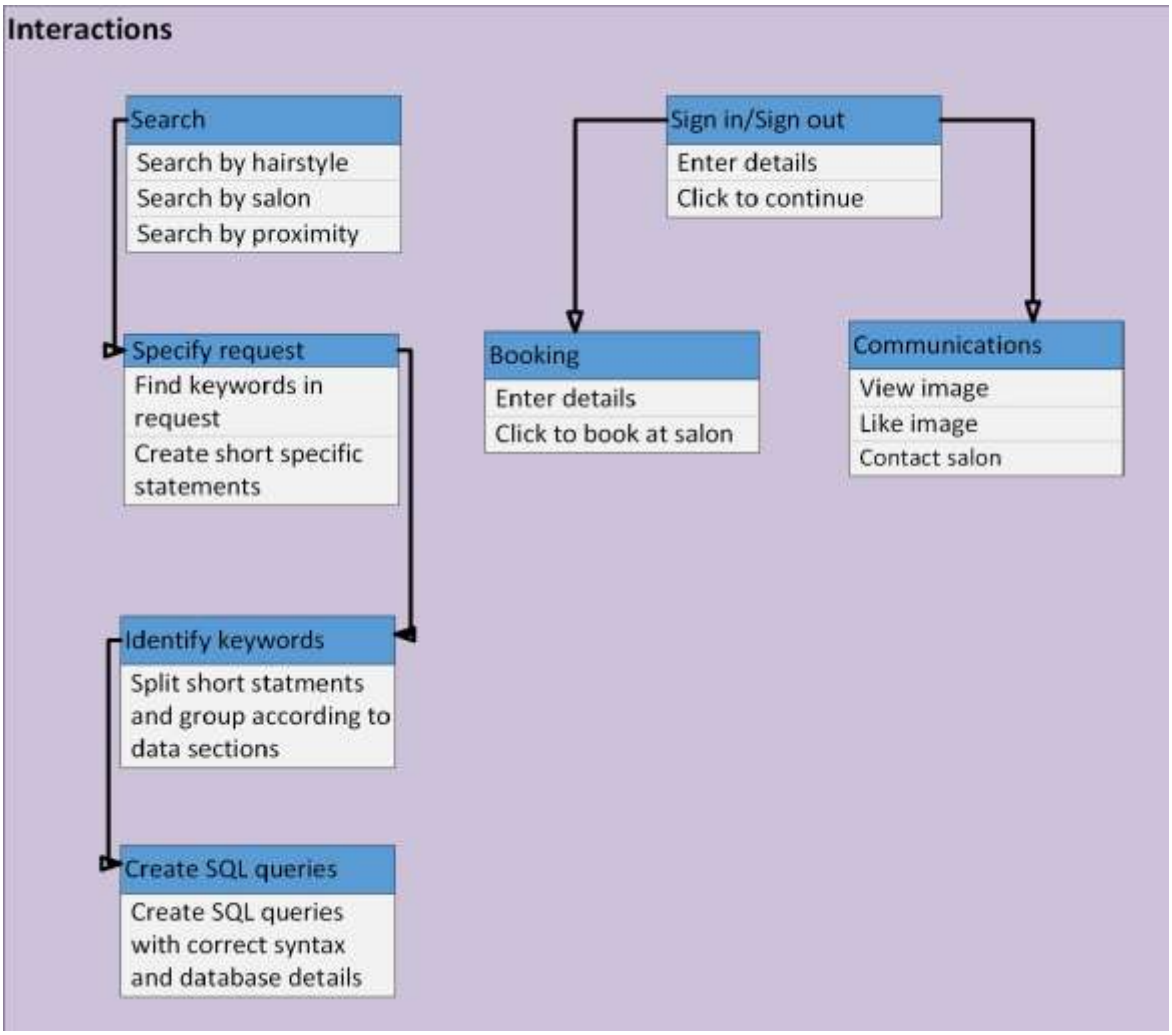


Figure 8: UML Class Diagram

4.1.3 Known Issues with View

Incorrect interface descriptions
Confusion with infrastructure and system models
Many dependencies; complexity
Not enough details to describe models

4.2 View: Process

The process view describes the main functionality of the system, the non-functional requirements of the software. This view outlines the tasks and distribution, systems' integrity, fault-tolerance and how the functionality of the system relates to the logic. The process view outlines the different levels that make up the entire structure of the system; the processes followed and communication within that structure. This view is defined by the **operational viewpoint**.

4.2.1 Models

UML Activity Diagram
UML Sequence Diagram

4.2.2 UML Activity Diagram

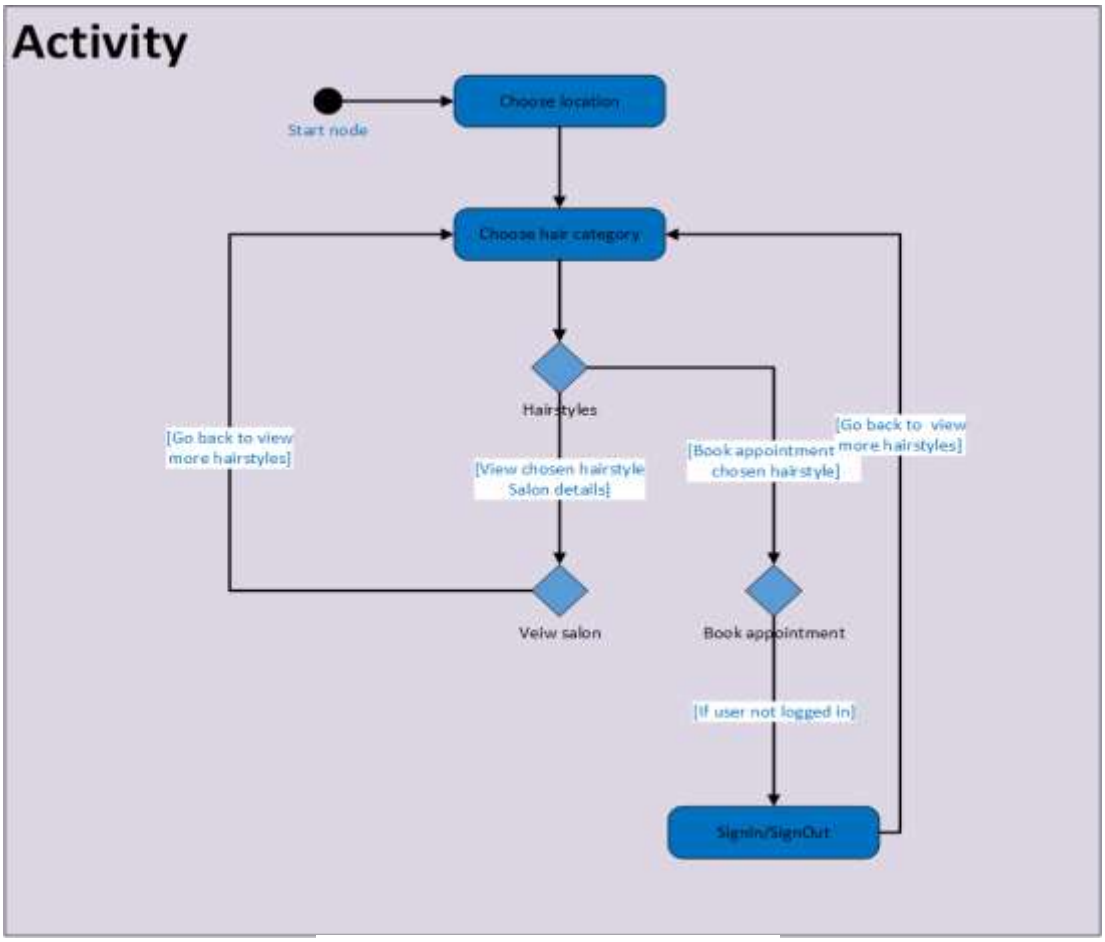


Figure 9: UML Activity Diagram

4.2.3 UML Sequence Diagram

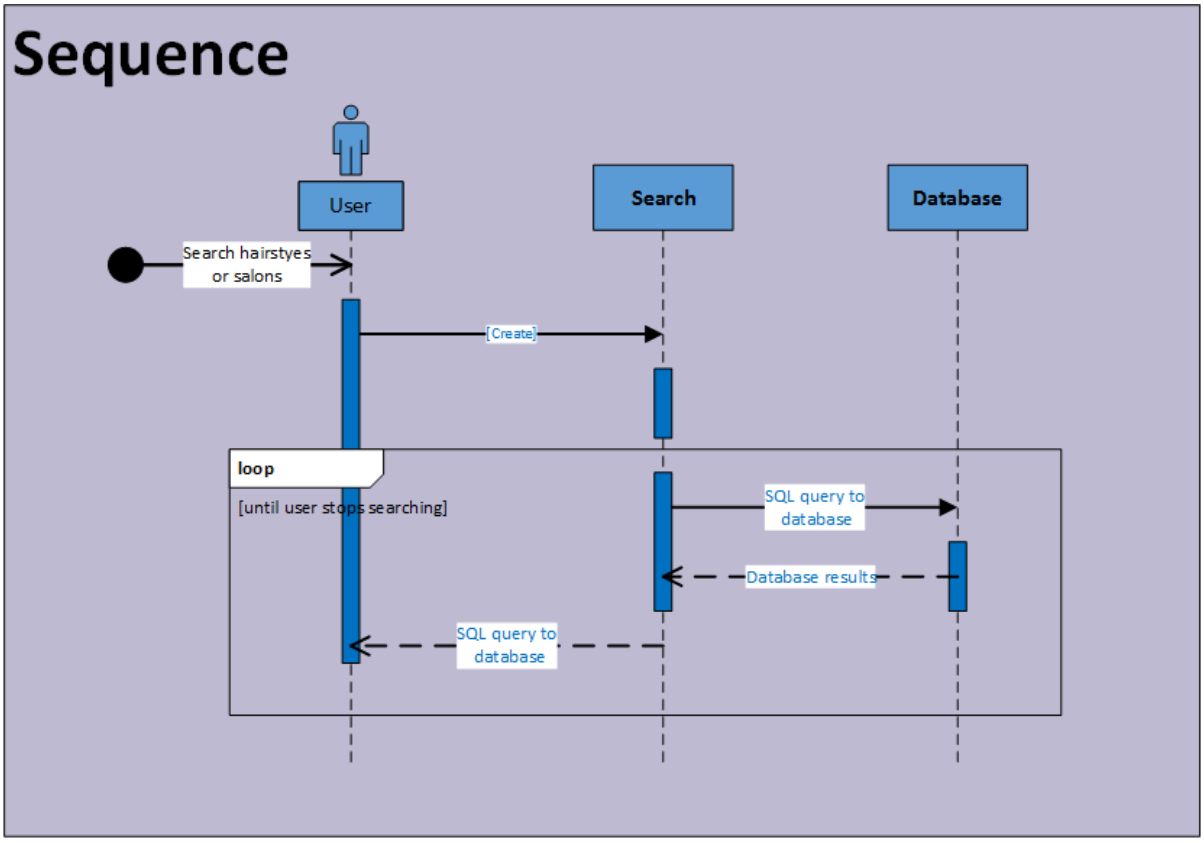


Figure 10: UML Sequence Diagram

4.2.4 Known Issues with View

Unable to trace flow
Insufficient backup models
Lack of migration planning
Missing organizational tools
Environment constraints
Lack of integration into production environment

4.3 View: Development

The development view deals with the organisation in the software development environment. The software is packaged in a way that allows groups of developers of work on certain parts of the system, so this view shows the building blocks of system and describes organization of the system modules. This view takes into account internal requirements that allows for easy of development, software management and reuse. The development view is defined by the **development viewpoint**.

4.3.1 Models

Component Diagram

4.3.2 Component Diagram

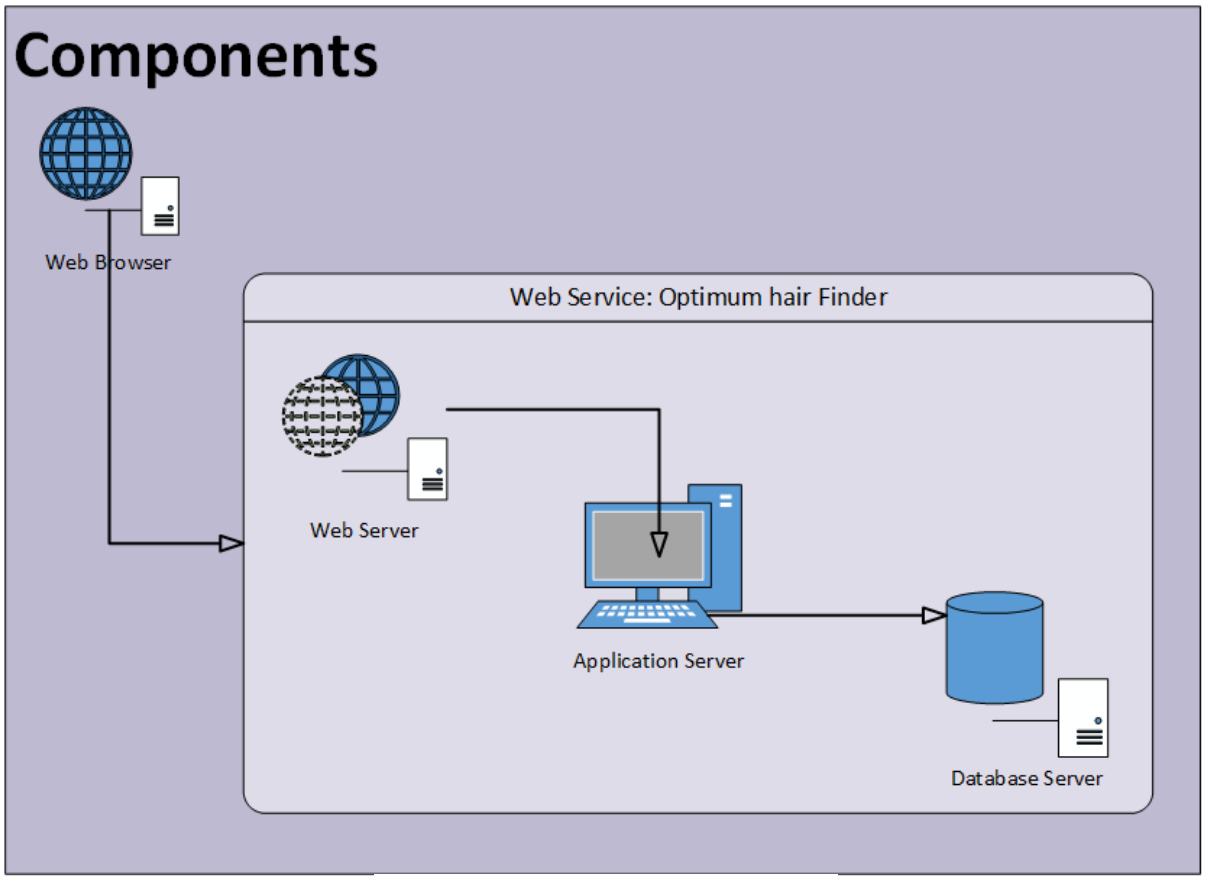


Figure 11: Component Diagram

4.3.3 Known Issues with View

Too much details
Incorrect focus
Lack of development
Lack of precision

4.4 Physical view

The physical view takes into account the non-functional requirements of the system. This view shows the installation, configuration and deployment of the software of the system. The software executes on a network of physical computers that need to be mapped onto the various processing nodes. Usually we use physical parts for development and testing, and we try for efficient physical components so that they do not interfere with the actual code. The process view is defined by the **information viewpoint**.

4.4.1 Models

Deployment Diagram

4.4.2 Deployment Diagram

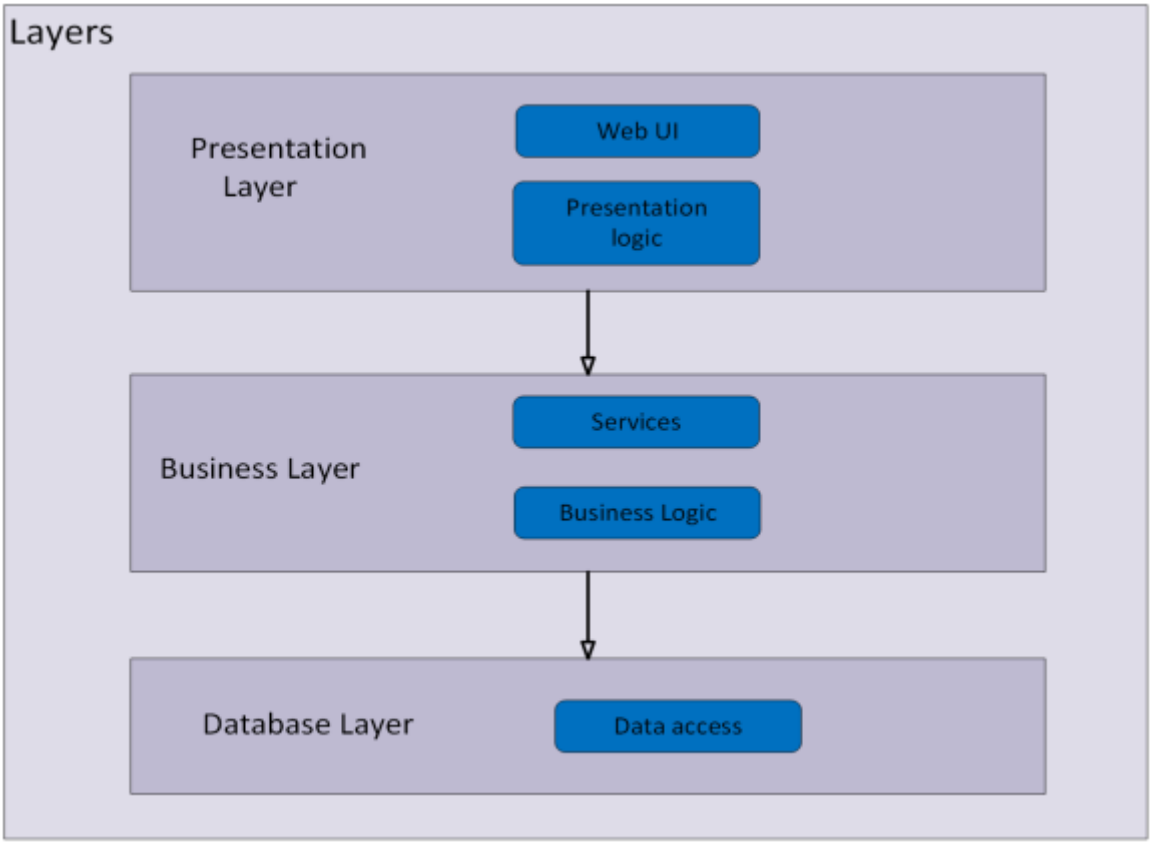


Figure 12: Deployment Diagram

4.4.3 Known Issues with View

Presentation inconsistencies
Lack updates models
Interface complexity
Overload and inconsistent database actions

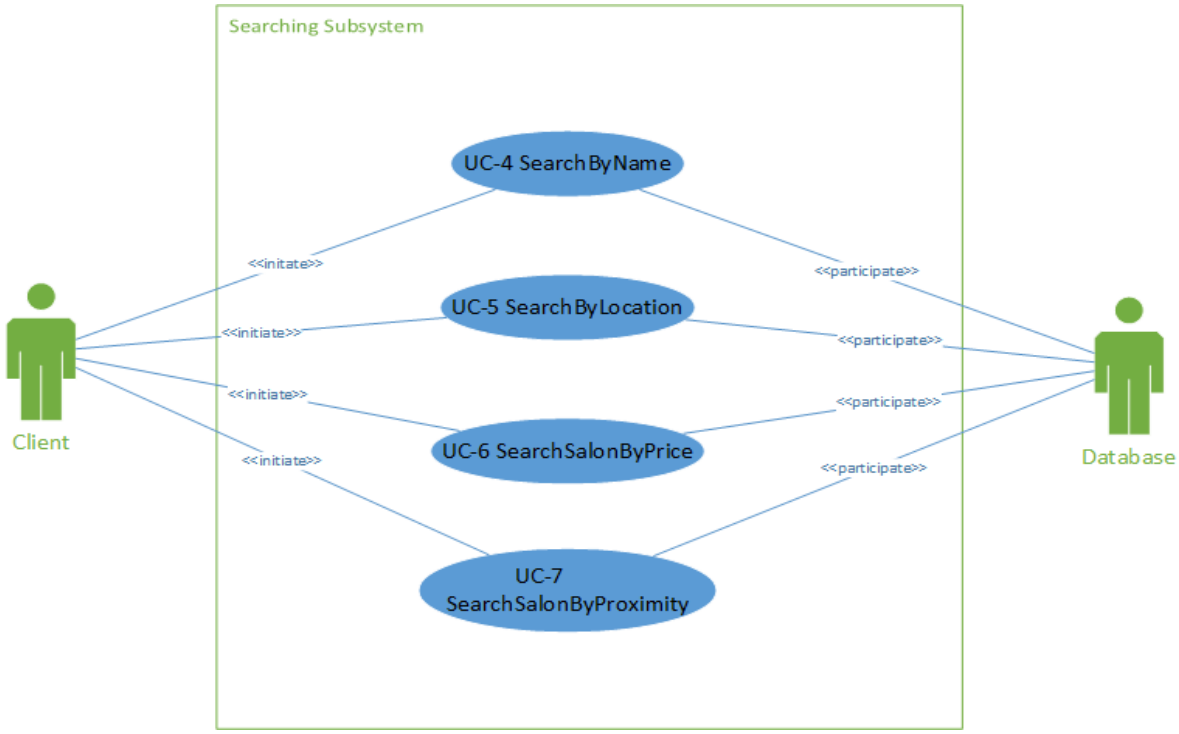
4.5 View: Scenarios

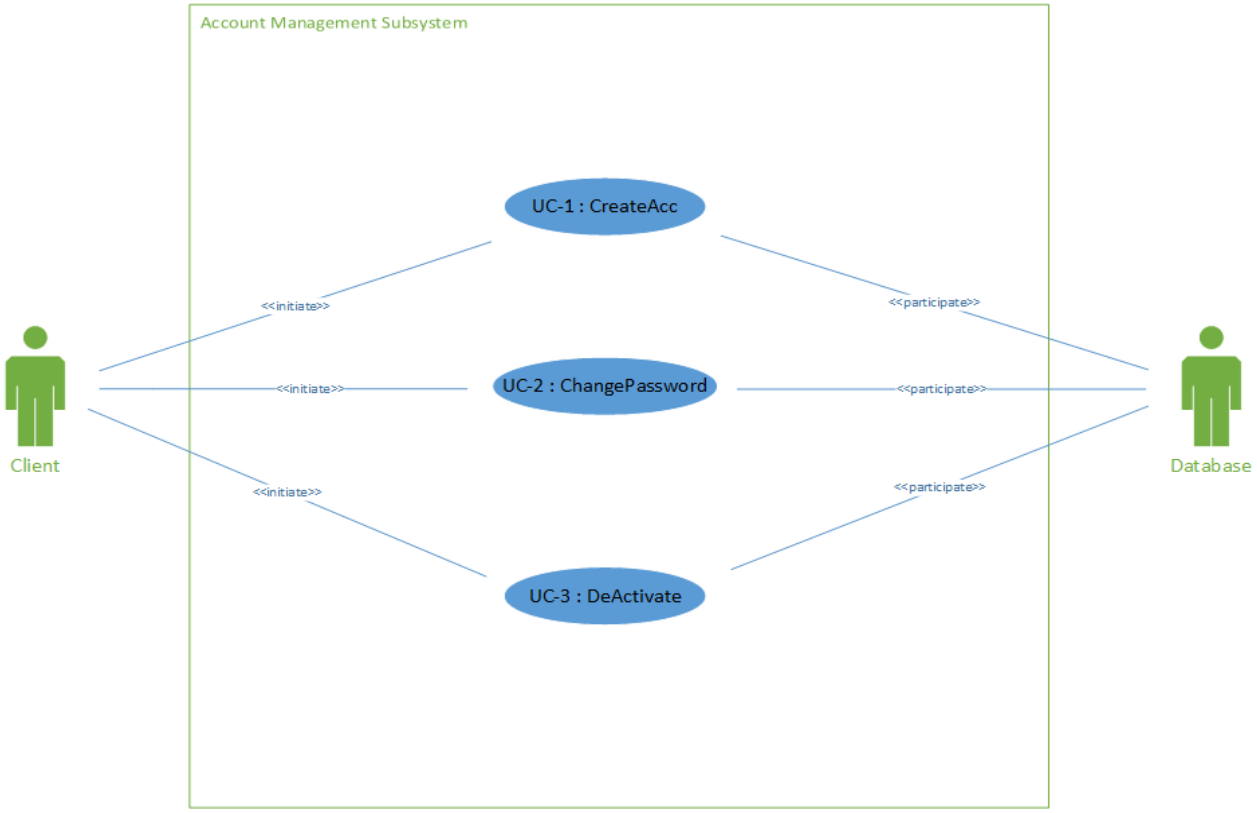
The scenarios view outlines the other views working in conjunction. It illustrates the “scenarios” which allows the 4 other views to work together. The scenarios described are the main requirements of the system and include the objects or main models shown in diagrams and interactions indicated. The scenarios view is important as it shows the architectural elements during the design of a system and shows the roles in relevance after the design. i.e. The scenarios is a sort of pseudo system that outlines main system requirements.

4.5.1 Models

Use Case Diagram

4.5.2 Use Case Diagram


Searching Subsystem
UC-4 SearchByName
UC-5 SearchByLocation
UC-6 SearchSalonByPrice
UC-7
SearchSalonByProximity
<<initate>>
<<initiate>>
<<initiate>>
<<initiate>>
<<participate>>
<<participate>>
<<participate>>
<<participate>>
Client
Database


Account Management Subsystem
UC-1 : CreateAcc
UC-2 : ChangePassword
UC-3 : DeActivate
<<initiate>>
<<initiate>>
<<initiate>>
<<participate>>
<<participate>>
<<participate>>
Client
Database

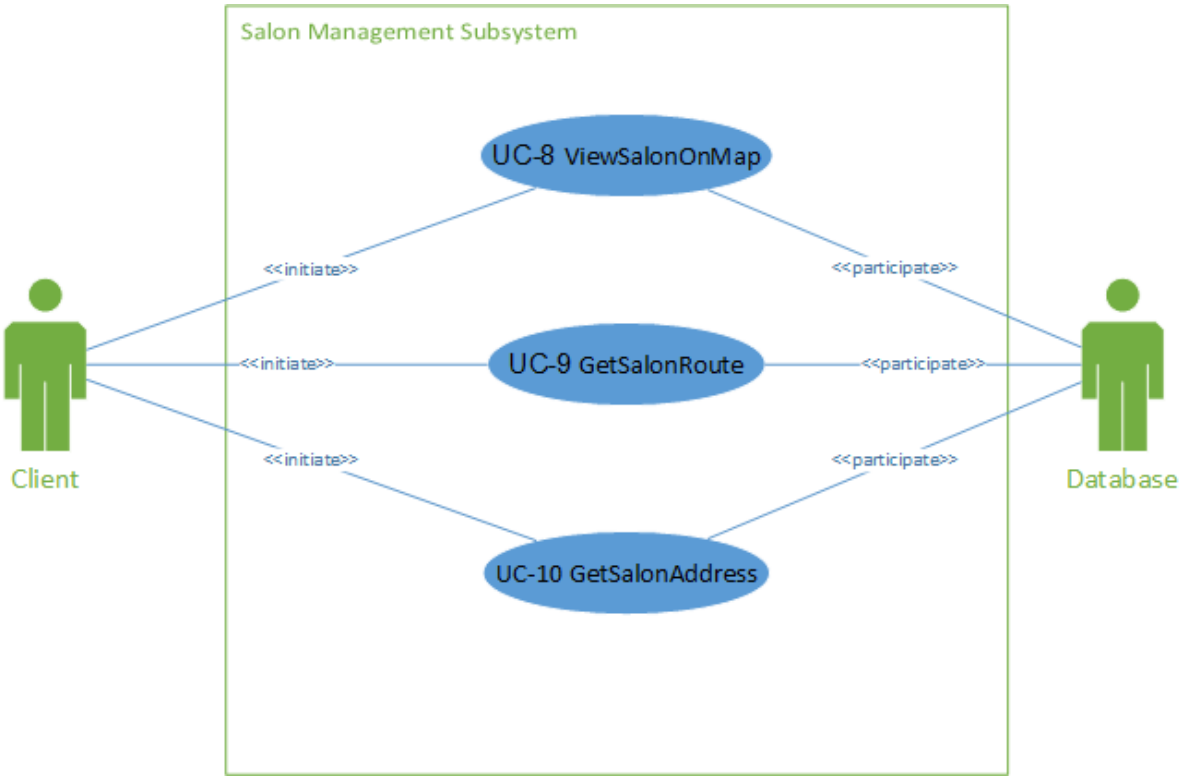


Figure 13: Use Case Diagrams

4.5.3 Known issues with view

Use case - system inconsistencies
Unachievable scenarios
Incorrect scenario implementation
Expendability of system with regard to use case

Consistency and correspondences

5.1 Known inconsistencies

The layered pattern in is arranged in a way that runs the system as a procedure. This could have initially been produced to eliminate confusion, inconsistency and interactions problems; however, it is not always the most optimal. The logical, process, development and physical levels of this layered structure have to follow one another in the execution of the system but some requests handled find some steps unnecessary. This creates what's called the **sinkhole anti-pattern**. The sinkhole anti-pattern is the result of the layer not being to do anything with a certain request so it is moved to next layer. The movement, from one to the next with not execution is view as a sort of sinkhole, until request reaches last layer.

5.2 Correspondences in the AD

The views in this AD are connected, elements from one view are connected to elements from the next and that causes the continuity that we see.

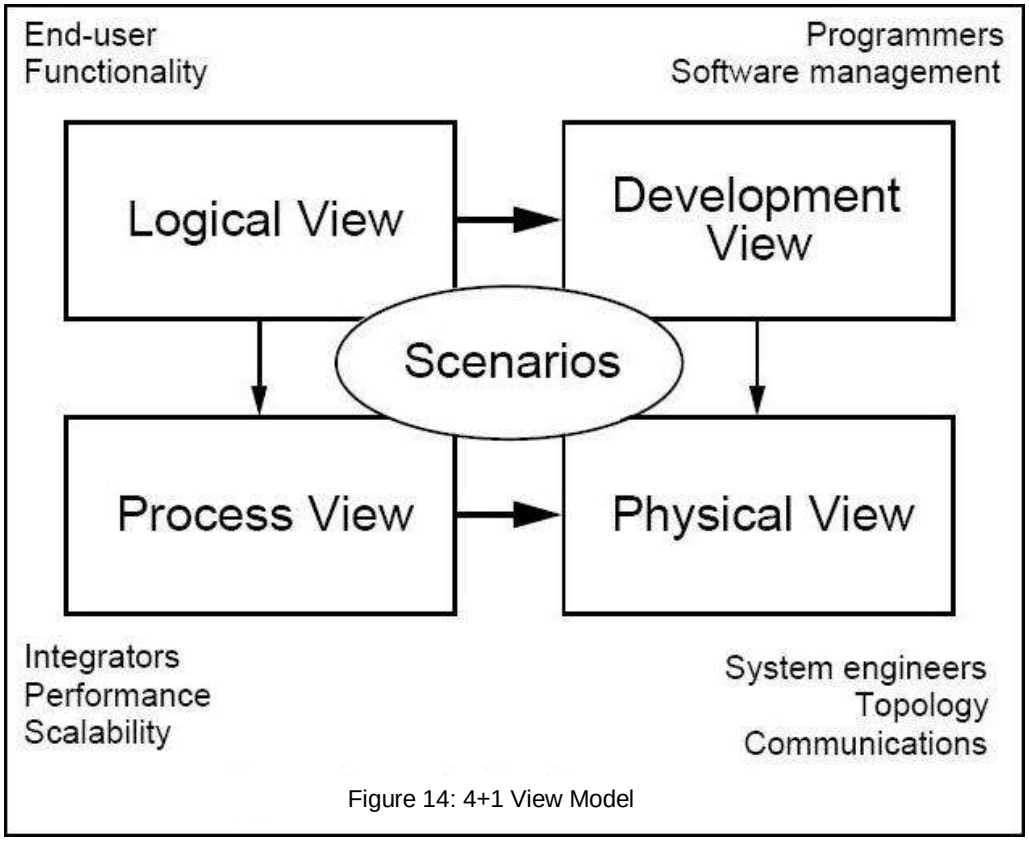


Figure 14: 4+1 View Model

5.3 Correspondence rules

From logical view to process view
The logical view illustrates the objects of the system, so from this we are able to understand the underlying function of the system. The objects will also show relationships

between them and process flow. This leads onto the process view since the process view mostly deals with the handling of tasks within the system. The process view shows the concurrency, the systems' integrity and process it takes to complete a task, it is a more in-depth view of the system.

From logical to development
Since the logical is like an overview of the system with the main objects indicated, the development of the system can also be determined from this view. Development is how the actual product at the end will be implemented; the subsystems to be worked on, team orgnisation, the amount of actual code that will be needed and a complete schedule. From the structure of the objects, teams and tasks can be assigned, coding languages most beneficial, if code can be reused can be decided on, time it will take to complete tasks and how all those will be put together to create a final product.

From process to physical
The process view describes the steps that are run in the system while the physical illustrates the actual parts of the system. The correspondence between these two views shows how the processes are mapped onto the physical aspect of the system.

Bibliography

Clements, Paul C. et al. Documenting Software Architectures: views and beyond. 2nd. Addison Wesley, 2010.

Finkelstein, A. et al. "Viewpoints: a framework for integrating multiple per-spectives in system development". In: International Journal of Software Engineering and Knowledge Engineering 2.1 (Mar. 1992), pp. 31–57.

Heesch, Uwe van, Paris Avgeriou, and Rich Hilliard. "A Documentation Framework for Architecture Decisions". In: The Journal of Systems & Software 85.4 (Apr. 2012), pp. 795–820. DOI: 10.1016/j.jss.2011. 10.017.

IEEE Std 1471, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Oct. 2000.

ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description. Dec. 2011, pp. 1–46.

Ran, Alexander. "ARES Conceptual Framework for Software Architecture". In: Software Architecture for Product Families Principles and Practice. Ed. by M. Jazayeri, A. Ran, and F. van der Linden. Addison-Wesley, 2000, pp. 1–29.

Rozanski, Nick and Eoin´ Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. 2nd. Addison Wes-ley, 2011.

Microsoft. MSDN Library. DevelopmentTools and Languages.Visual Studio 2015 https://msdn.microsoft.com/en-us/library/dn762121.aspx

Ivan Maersic. Software Engineering. Rutgers University, New Brunswick, New Jersey. September, 2012.

Dr. Andreas Schroeder. Microservices Architecture.

Mark Richards. Software Architecture Pattern: Understand common architecture patterns and when to use them. Ed. Heather Scherer. O' Reilly Media, Inc. ISBN: 978-1-491-92424-2, February, 2015. pp. 9-18

Philippe Kruchten. Architectural Blueprints-The "4+1" View Model of Software Architecture. Rational Software Corp. November,1995. pp. 42-50