**Software Architecture**

**Document**

Version X.X

for

GetARoom

Prepared by

|  |  |  |
| --- | --- | --- |
| Mihai Damsachin | 27177895 | mi\_damas@encs.concordia.ca |
| Jesus Imery | 27174276 | imeryjesus@gmail.com |
| Meetaz Alshbli | 25558840 |  |

Samuel Campbell 26457959 samuel.campbell@gmail.com

|  |  |
| --- | --- |
| Instructor: | Dr. Constantinos Constantinides |
| Course: | SOEN 343: Software Architecture and Design I |
| Date: | <date of submission> |

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 11/14/2016 | 1 | Part 1 | Meetaz Alshbli |
| 11/17/2016 | 1 | Communication diagrams | Meetaz Alshbli  Samuel Dufresne |
| 11/18/2016 | 1 | Parts #2- #3 | Samuel Campbell |
| 11/20/2016 | 1 | Revision of SAD part 1 | Mihai Damaschin |
| 11/20/2016 |  | Functional and non-functional requirements, size and performance, quality | Meetaz Alshbli  Jesus Imery  Samuel Dufresne |
| 11/20/2016 | 1 | Revision of all document  Did parts 7, 8 ,10 | Samuel Campbell |
| 11/21/2016 | 1 | Completed Data view Section | Samuel Campbell |

**Table of contents**

[**1. Introduction**](#_30j0zll)[**6**](#_30j0zll)

[1.1 Purpose](#_phzp1q5zdbcv) [6](#_phzp1q5zdbcv)

[1.2 Scope](#_3znysh7) [6](#_3znysh7)

[1.3 Definitions, acronyms, and abbreviations](#_yt1z2okdsj93) [7](#_yt1z2okdsj93)

[**2. Architectural representation**](#_gd10d521cje3)[**8**](#_gd10d521cje3)

[2.1 Logical view :](#_cjc1spr3336e) [8](#_cjc1spr3336e)

[2.2 Development view](#_atwpov1i9x68) [9](#_atwpov1i9x68)

[2.3 Process view :](#_oj1yg1s8vv4h) [9](#_oj1yg1s8vv4h)

[2.4 Physical view](#_3v1uvhkhpvae) [9](#_3v1uvhkhpvae)

[2.5 Use case view](#_k1wn9e56ga63) [9](#_k1wn9e56ga63)

[2.6 Data view:](#_2jg7yh6vdcp2) [9](#_2jg7yh6vdcp2)

[**3. Architectural requirements: goals and constraints**](#_t4cfa884aed7)[**10**](#_t4cfa884aed7)

[3.1 Technical Platform](#_t1eg3t3q40on) [10](#_t1eg3t3q40on)

[3.1.1 Server Side](#_5ymdwxq4ukif) [10](#_5ymdwxq4ukif)

[3.1.2 Client Side](#_mnpqnjcthkn3) [10](#_mnpqnjcthkn3)

[3.2 Functional requirements (Use case view)](#_wlpuqfg6tx3j) [11](#_wlpuqfg6tx3j)

[3.3 Non-functional requirements](#_17dp8vu) [13](#_17dp8vu)

[**4. Use case view (Scenarios)**](#_rjrningtceqs)[**15**](#_rjrningtceqs)

[4.1 Actors](#_n6xjl7vdkybn) [16](#_n6xjl7vdkybn)

[4.1.1 User](#_xomzk0oprj9f) [16](#_xomzk0oprj9f)

[4.1.2 System](#_v7em4oyflx1v) [16](#_v7em4oyflx1v)

[4.2 Brief Use Case Description](#_50z20rieu76x) [17](#_50z20rieu76x)

[4.2.1 Login](#_6fojyyop5kab) [17](#_6fojyyop5kab)

[4.2.1.1 Communication Diagrams](#_g4ofrz5j724q) [17](#_g4ofrz5j724q)

[4.2.1.1.1 Login](#_dl70ostl959p) [17](#_dl70ostl959p)

[4.2.2 Logout](#_qhcy6wkrehs) [18](#_qhcy6wkrehs)

[4.2.2.1 Communication Diagram](#_zhlsceruxyux) [18](#_zhlsceruxyux)

[4.2.2.1.1 Logout](#_benppl99xqjs) [18](#_benppl99xqjs)

[4.2.3 Make Reservation](#_gylqqnkp64qe) [18](#_gylqqnkp64qe)

[4.2.3.1 Communication Diagrams](#_ybiogr5v2pp4) [19](#_ybiogr5v2pp4)

[4.2.3.1.1 Initiate Reservation Session](#_he7vzcllixmp) [19](#_he7vzcllixmp)

[4.2.3.1.2 Add Reservation](#_3zev4r8hkr8n) [19](#_3zev4r8hkr8n)

[4.2.3.1.3 End Reservation Session](#_eqahn8evd136) [20](#_eqahn8evd136)

[4.2.4 Remove Reservation](#_i7m012tql7gn) [20](#_i7m012tql7gn)

[4.2.4.1 Communication Diagrams](#_2xx2ao8dun4m) [20](#_2xx2ao8dun4m)

[4.2.4.1.1 Initiate Reservation Session](#_cm2tntyf0s0q) [20](#_cm2tntyf0s0q)

[4.2.4.1.2 Remove Reservation](#_oyblz5cjy0qh) [21](#_oyblz5cjy0qh)

[4.2.4.1.3 End Reservation Session](#_1s9fbnbtlzjn) [21](#_1s9fbnbtlzjn)

[4.2.5 View Reservation](#_7yn5punghy4t) [21](#_7yn5punghy4t)

[4.2.5.1 Communication Diagram](#_i1zgnhbax9ug) [22](#_i1zgnhbax9ug)

[4.2.5.1.1 View Reservations](#_i7iv6ofebd68) [22](#_i7iv6ofebd68)

[4.2.6 Modify Reservation](#_wtksxpjiyau1) [22](#_wtksxpjiyau1)

[4.2.6.1 Communication Diagrams](#_3xuixylufhp7) [22](#_3xuixylufhp7)

[4.2.6.1.1 Initiate Reservation Session](#_i6zdk2o5bkgh) [22](#_i6zdk2o5bkgh)

[4.2.6.1.2 Modify Reservation](#_t038xncliece) [23](#_t038xncliece)

[4.2.6.1.3 End Reservation Session](#_9n780jn5g0kh) [23](#_9n780jn5g0kh)

[**5. Logical view**](#_26in1rg)[**24**](#_26in1rg)

[5.1 Tier diagram](#_s496b8cksiie) [25](#_s496b8cksiie)

[5.2 Subsystems](#_nil4p23p872r) [26](#_nil4p23p872r)

[5.3 Architecturally significant design packages](#_jkchpcc6avae) [26](#_jkchpcc6avae)

[5.4 Use case realizations](#_1ksv4uv) [26](#_1ksv4uv)

[**6. Development (Implementation) view**](#_44sinio)[**28**](#_44sinio)

[6.1 Reuse of components and frameworks](#_2jxsxqh) [28](#_2jxsxqh)

[6.1.1 Angular JS:](#_mcmcmkshaor9) [28](#_mcmcmkshaor9)

[6.1.2 DropWizard:](#_yp6z7k3e3pve) [29](#_yp6z7k3e3pve)

[6.1.3 MySQL:](#_uw7rzffe4yq3) [29](#_uw7rzffe4yq3)

[**7. Process View**](#_z337ya)[**30**](#_z337ya)

[**8. Deployment (Physical) view**](#_hotun0waghfb)[**31**](#_hotun0waghfb)

[**9. Data View**](#_aem6zkaetbqf)[**33**](#_aem6zkaetbqf)

[**10. Size and performance**](#_d8b5kkrsryyx)[**34**](#_d8b5kkrsryyx)

[10.1 Size](#_mbx3gfn5vzft) [34](#_mbx3gfn5vzft)

[10.2 Performance](#_o6zy7yjt63gl) [34](#_o6zy7yjt63gl)

[**11. Quality**](#_2xcytpi)[**34**](#_2xcytpi)

[11.1 Portability](#_ygdlugkfja4v) [34](#_ygdlugkfja4v)

[11.2 Scalability](#_gmts16vajqii) [34](#_gmts16vajqii)

[11.3 Reliability, Availability:](#_la3civ7axpmr) [35](#_la3civ7axpmr)

[11.4 Security:](#_pnv6gb5ehu7c) [35](#_pnv6gb5ehu7c)

[11.5 Configurability:](#_f08x294hk5fx) [35](#_f08x294hk5fx)

[**References**](#_hqazrz8c5b5e)[**35**](#_hqazrz8c5b5e)

**List of figures**

[Figure 1: The 4+1 view model.](#_1t3h5sf)

# 1. Introduction

This SAD document describes the high level and low level designs of the GetARoom system. This text illustrates the architecture of the system through class and interaction diagrams. Furthermore, this document contains guidelines on how to create and maintain the system as well as provides an overview of the hardware and software platforms.

## 1.1 Purpose

The purpose of this Software Architecture Document is to provide a straightforward architectural view of our GetARoom system that illustrates the reasoning behind every decision made throughout the system design. It separates logical sections of the whole system, and provides an architectural description of these sections so that the reader may understand the design and implementation details of these system components. The document is directed primarily to software developers, designers, managers, and stakeholders.

The document is separated into six sections. First, the scenarios describing the system’s use cases along with an overview of said use cases. Second, a logical view to illustrate the object-oriented classes, their behavior and the states that comprise our system. Third, a development view that illustrates how the system components are packaged. Fourth, a process view that displays a workflow representation of the system. Fifth, a physical view of the system demonstrated by a deployment diagram. Lastly, a data view shown by an entity-relationship (E/R) diagram.

## 1.2 Scope

This software architecture document represents all the architectural designs along with the database structure. This text covers significant decisions which influence the entirety of the web application.

## 1.3 Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | A registered Individual who interacts with the web application |
| SAD | Software Architecture Document |
| GetARoom | Name of the web application |
| AJAX | Asynchronous Javascript and XML |
| XML | Extensible Markup Language |
| HTTP | HyperText Transfer Protocol |
| Stakeholder | Investors, employees and customers |
| UML | Unified Modeling Language |
| MySQL | Relational database management system |
| WWW | World wide web |
| UI | User Interface |
| SQL | Structured Query Language |
| JSON | Javascript Object Notation |
|  |  |

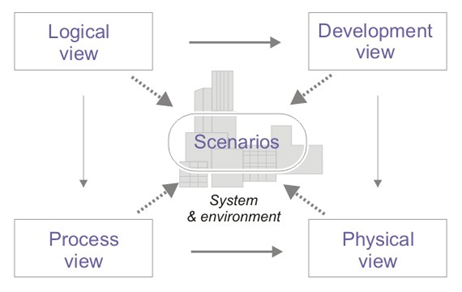
## 

## 1.4 References

[1] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications (SRS), IEEE Computer Society.

# 2. Architectural representation

This document details the architecture using the views defined in the “4+1” architecture model.



## 

## 

## 2.1 **Logical view** :

Audience : Designers

Area : Functional Requirements: Define a system’s function and its components. Additionally this describes the most important Use-Case realizations.

Related Artifacts: Design Model

## **2.2 Development view**

Audience : Programmers

Area : Software Components: Describes modules and subsystem of the application.

Related Artifacts: Implementation model, components

## **2.3 Process view** :

Audience : Integrators

Area : Non-Functional Requirements: Describes the design’s concurrency and synchronization aspects.

Related Artifacts: No specific artifacts

## **2.4 Physical view**

Audience : Deployment Managers

Area : Topology: Describes the mapping of the software unto the hardware and shows the system’s distributed aspects.

Related Artifacts: Deployment Model

## **2.5 Use case view**

Audience : All stakeholders and end-users

Area : Describes the set of scenarios and use cases that represent some significant central functionality of the system.

Related Artifacts: Use-Case Model, Use-case Documents

## **2.6 Data view**:

Audience: Data administrators

Area: Describes the persistent elements in the data model of the database

Related Artifacts: Data model

# 3. Architectural requirements: goals and constraints

This section describes the necessary requirements which are important in the development of the software architecture.

## 3.1 Technical Platform

### 3.1.1 Server Side

GetARoom will be using Nginx which will receive HTTP requests on port 80. The requests shall conform to the standard HTTP and TCP/IP protocols. Furthermore, they shall be forwarded to port 8080 on the server where the Jetty web server/application container is bound to. From this, GetARoom shall be able to connect to the server MySQL database from localHost. Finally the server shall be hosted on the linux operating system.

### 3.1.2 Client Side

The GetARoom tool shall only be accessible by the internet from the client’s side. A user may do so by utilizing a modern web browser such as Safari, Chrome, Chromium or Firefox.

## 3.2 Functional requirements (Use case view)

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| Use Case UC1 | Login | Any user who wishes to utilize the website must login first to access any of its functions | SAD -> 4.2.1  SRS -> 4.1 |
| Use Case UC3 | Make Reservation | User shall be able to reserve a room on a multiple time slots | SAD -> 4.2.3  SRS -> 4.2 |
| Use Case UC4 | Remove Reservation | User shall be able to remove any of his existing reservation(s) from the database | SAD -> 4.2.4  SRS -> 4.4 |
| Use Case UC5 | View Reservation | User shall be able to view all of his existing reservation per room | SAD -> 4.2.5  SRS -> 4.6 |
| Use Case UC6 | Modify Reservation | User shall be able to modify the time slot for any of his existing reservation(s) in the database | SAD -> 4.2.6  SRS -> 4.3 |

## 3.3 Non-functional requirements

|  |  |
| --- | --- |
| **Name** | **Architectural relevance** |
| Performance | The system average response time shall be lower than 3 seconds |
| Security | Account protection: The system shall authenticate the user by verifying both the username and password credentials’ existence in the database and the correspondence of the password to the username.  Confidentiality: All users remain anonymous from each other. |
| Persistence | Data persistence shall be implemented using relational databases. |
| Availability/Reliability | Availability and reliability shall be addressed by the J2EE platform. The system aims for 24 hours a day uptime.  Downtime of the system should only happen during maintenance occurring at irregular intervals. Users shall be notified of the event 48 hours prior with a warning on the homepage of the website. |

# 

# 

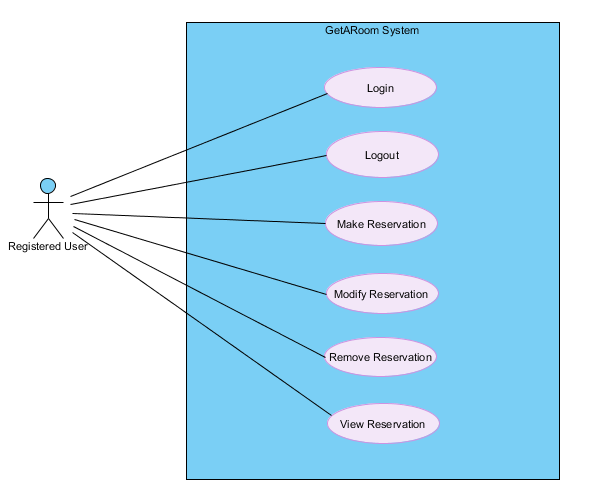
# 

# 4. Use case view (Scenarios)

This section describes the use case view of the architecture. This view represents a collection of significant use cases and scenarios that give the system its core functionality.

The GetARoom use cases are:

* Login
* Logout
* Make Reservation
* Modify Reservation
* Remove Reservation
* View Reservation



## 4.1 Actors

### 4.1.1 User

The user may create, delete, modify, and view room reservations

### 4.1.2 System

The system handles all logical processes in the software. The system defines the behaviour of the software as a black box.

### 4.2 Brief Use Case Description

Further explanation regarding the use may be found in the SRS.

*Note : All actors in the following use cases are students from a college institution.*

#### 4.2.1 Login

This use case allows a registered user to log into their account.

#### 4.2.2 Logout

This use case allows a currently logged in user to log out of their accounts and end their sessions.

#### 4.2.3 Make Reservation

This use case allows a user to make a room reservation.

#### 4.2.4 Remove Reservation

This use case allows a user to remove one of their existing reservations.

#### 4.2.5 View Reservation

This use case allows a user to view their room reservations.

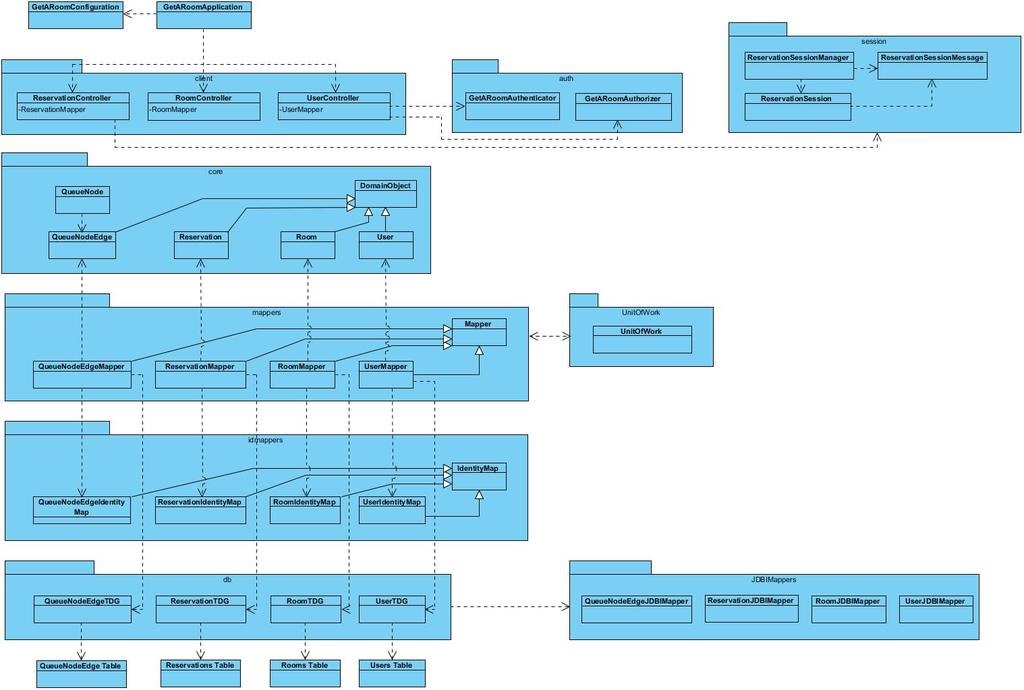
#### 4.2.6 Modify Reservation

This use case allows a user to modify one of their existing room reservations.

# 5. Logical view

The GetARoom application is designed based on the 3-tier client-server architecture. This model assigns different responsibilities to each layer. This strategy has been chosen due to its scalability and performance, good database security, and good maintainability.

Class diagram



## 5.1 Tier diagram



Each layer has specific responsibilities:

1. The Client Tier Layer is the top most layer of the application and also a layer supertype. Its function is to display all the system’s information into concrete information for the user.
2. The Application Server Tier is the medium by which the surrounding layers may communicate. The role of this layer supertype is to perform all logical decisions, evaluations and operations.
3. The layer supertype, Database Tier, is in charge of storing information which can be accessed by the application server tier. This data is then manipulated by the Application Server tier and then eventually sent to the user.

## 5.2 Subsystems

The system consists of the CoreElements subsystem. This subsystem contains 2 components: Authentication and Reservations.

1. The Authentication component allows users to login and logout of the system.
2. The Reservations component provides create, modify and delete operations to manage Reservation objects.

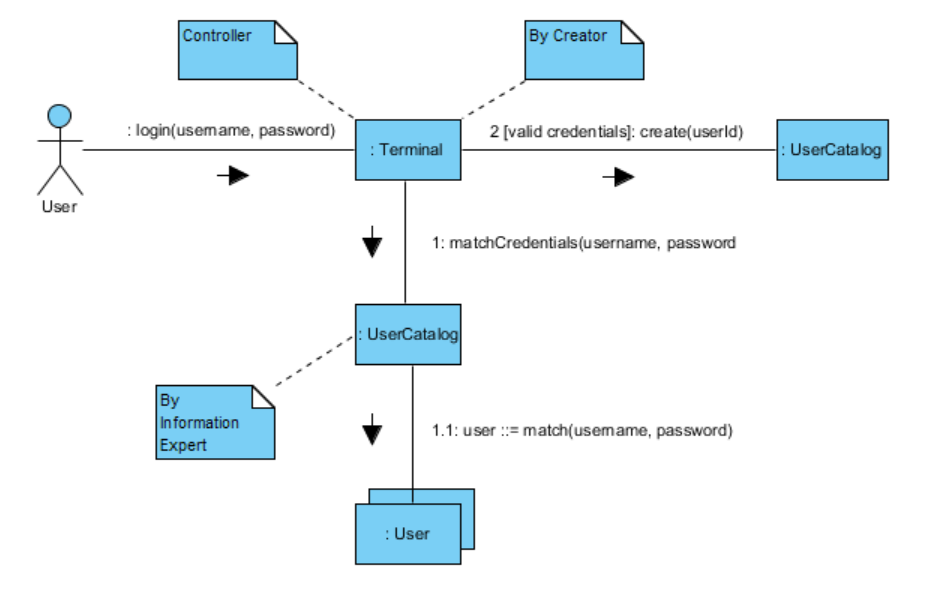
## 5.3 Use case realizations

The use realization diagram below shows how design elements provide functionalities identified in the critical use-cases. Refer to section #4 in the SRS to view the operation contracts, the system sequence diagams and the system operations.

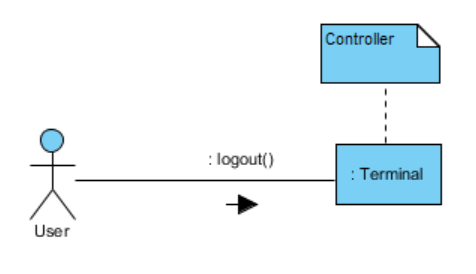


### 5.4.1 Communication Diagrams

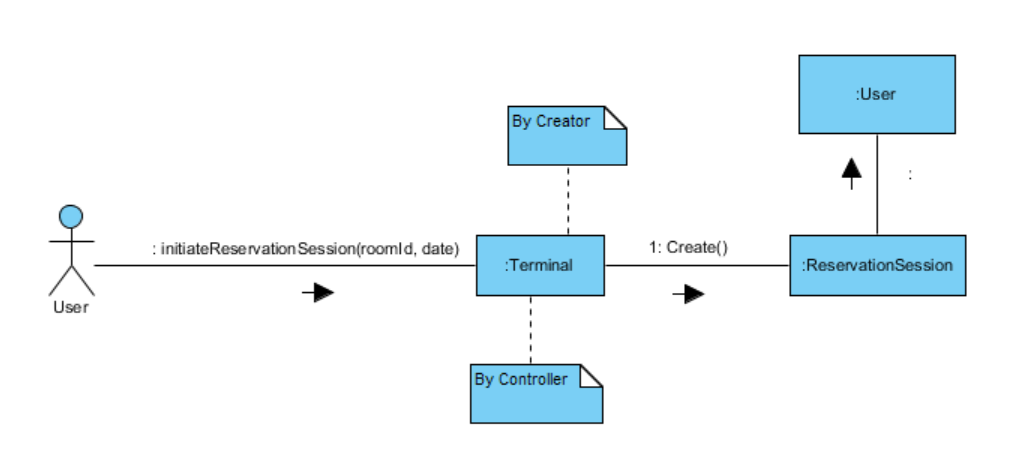
Login



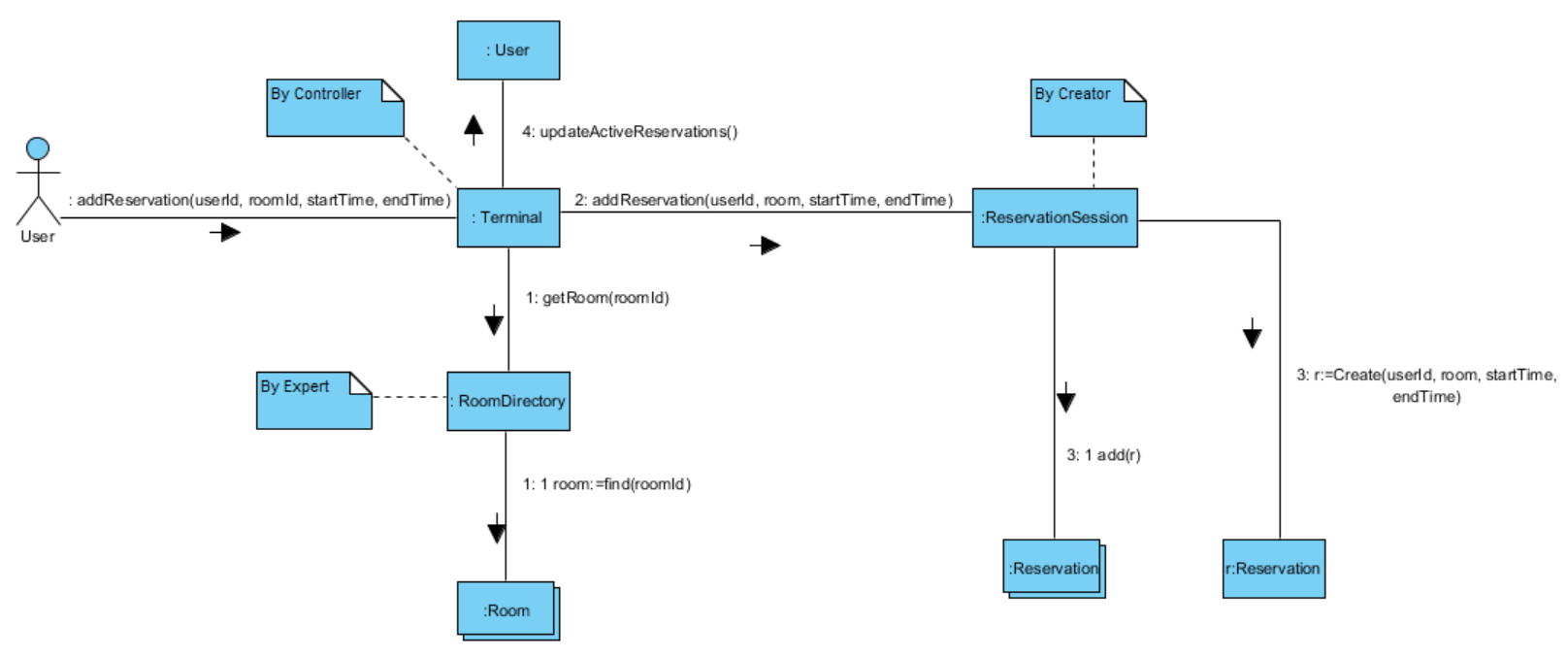
Logout



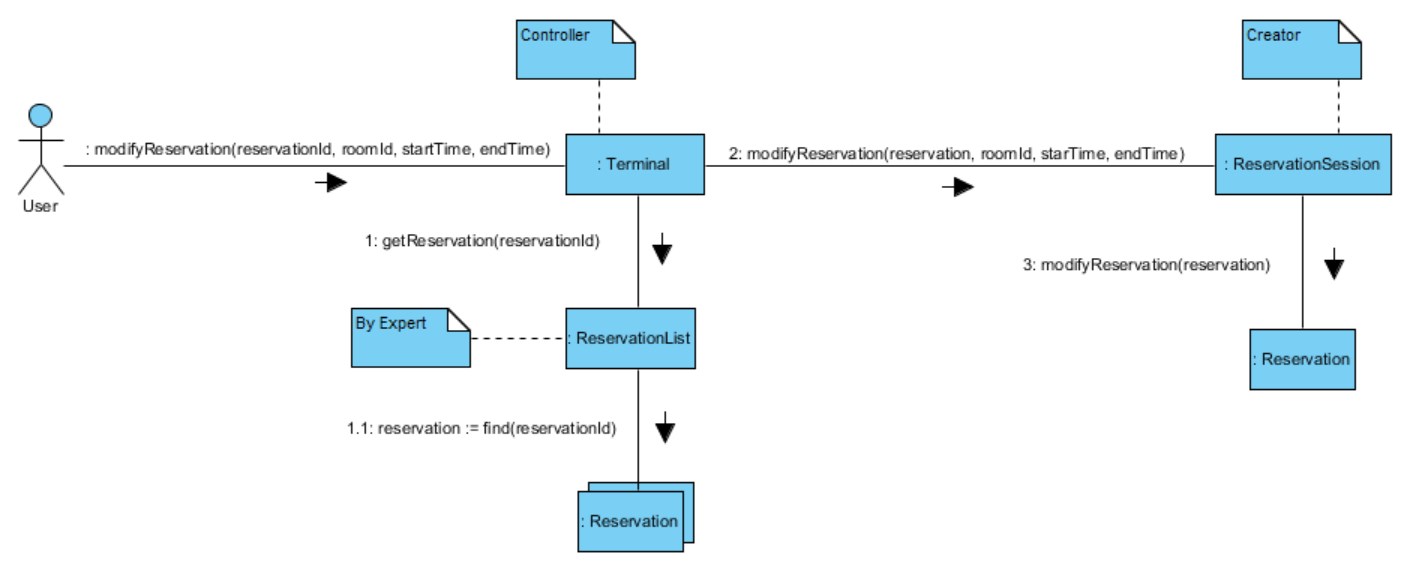
Initiate Reservation Session



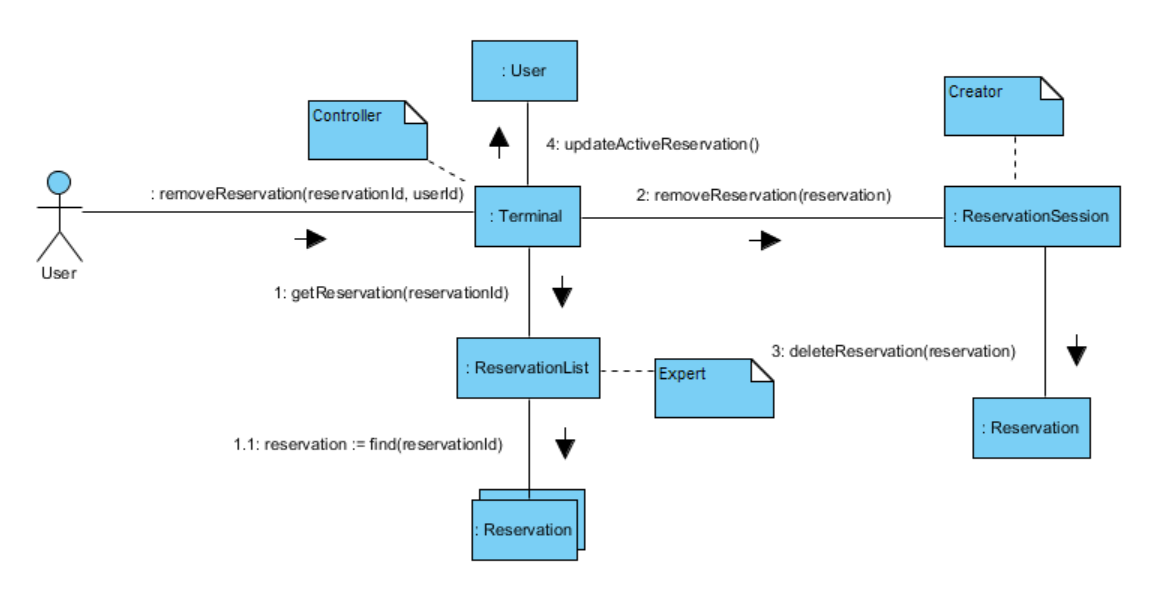
Add Reservation



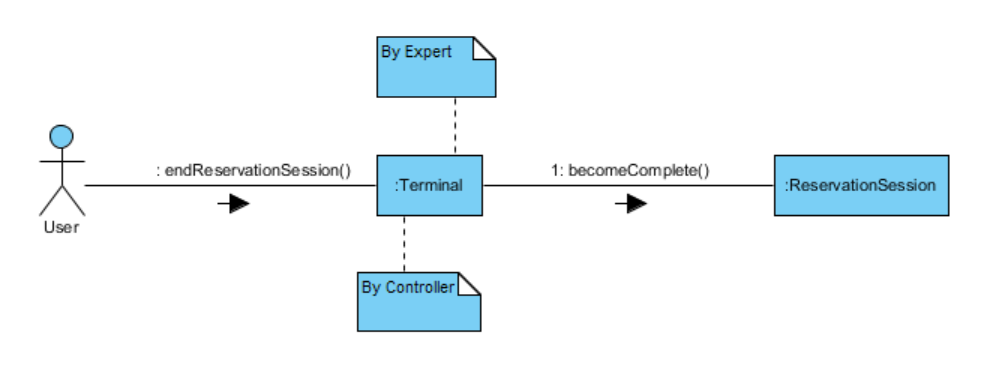
4.2.6.1.2 Modify Reservation



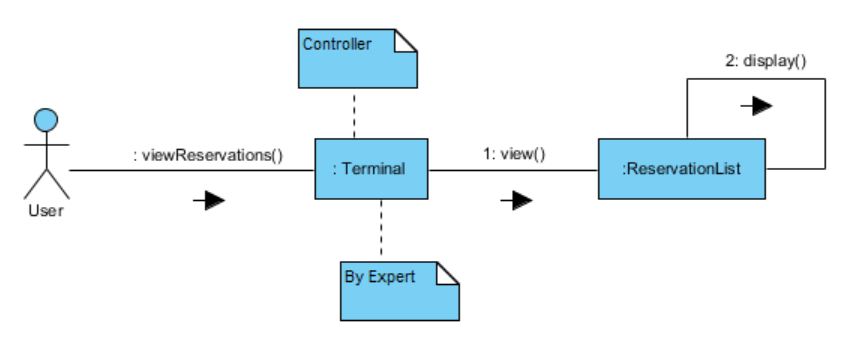
Remove Reservation



End Reservation Session



4.2.5.1.1 View Reservations



# 

# 

# 6. Development view



## 6.1 Reuse of components and frameworks

### 6.1.1 Angular JS:

Structural framework implemented with web applications that extends the HTML’s syntax to express a clear display of the program’s components. Data binding and dependency injection eliminate much of the code and promote reusability in the web pages. Angularjs eliminates repetition and enforces a stronger concept of the MVC infrastructure.

### 6.1.2 DropWizard:

DropWizard is a Java framework which encompases serveral other frameworks and Java libraries needed for REST services. This framework includes the following packages:

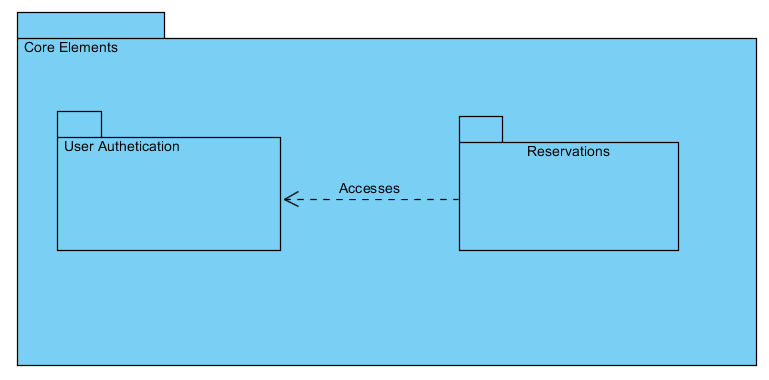
* Jetty: Jetty is a web server and a Java servlet container.
* JDBC: API which allows Java to access database management systems
* Jersey: Open source framework used to developing RESTful web services.
* Jackson: An API which serializes Java objects to JSON and JSON to Java objects.

### 6.1.3 MySQL:

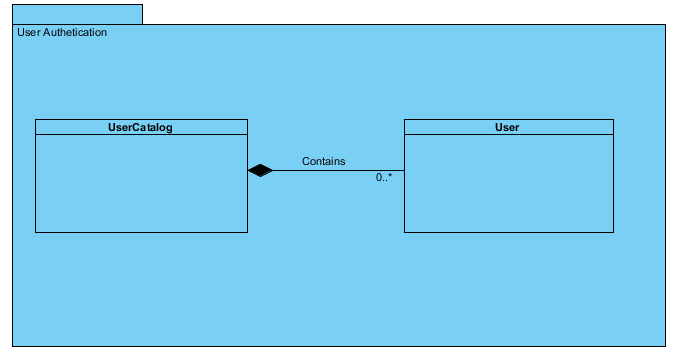
MySQL is an open source relational database system. This will be used to support GetARoom on the server side. MySQL is a framework for the SQL language.

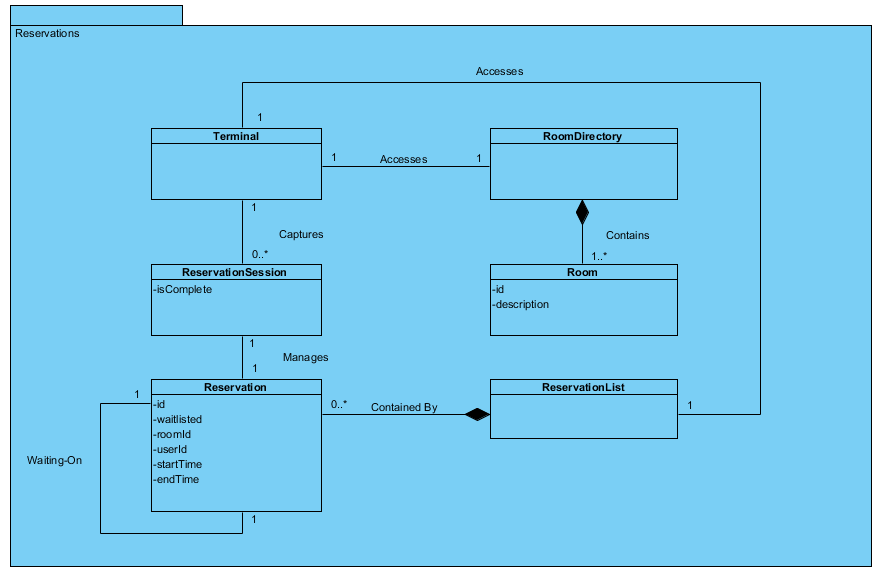
## 6.2 Architecturally significant design packages

In order to create the following package diagram, the DMO was partitioned into two separate elements that contain classes which share use cases and class hierarchy. Our system is not overly complicated, so it did not require much partitioning. We identified two main form of use cases: those that refer to the User and their credentials, and those that perform reservation operations on the room. Illustrated below are the core elements of our system.



We then expand in detail what classes these elements contain. These are essentially snapshots of the DMO that encapsulate the conceptual classes involved in these use cases.





# 

# 

# 

# 8. Physical view

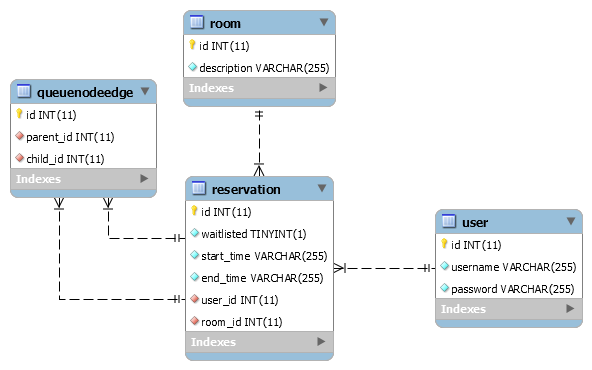


|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| :GenericPC | Device Node | This node is used to represent a user who may access the GetARoom program from a computer through a web browser |
| :webBrowser | Execution Environment Node | This node represents the web browser which will send HTTP requests on port 80 to nGinx on the KVM HyperVisor Instance server. |
| :KVM HyperVisor Instance | Device Node | The node represents the device which hosts the GetARoom application. |
| :Ubuntu 16.04 | Execution Environment Node | This node is the operating system of the server’s device. |
| :Nginx | Execution Environment Node | This node is the web server of GetARoom which receives HTTP requests on port 80. It will then redirect the HTTP request to Jetty, the servlet container. |
| :Jetty | Execution Environment Node | This node is the servlet container which receives HTTP requests on port 8080 from Nginx. |
| :GetARoom | Execution Environment Node | This node is the relational database of GetARoom. It receives SQL in the form (GET, FROM, WHERE…) on port 3306 of the server’s machine. |

# 

# 

# 9. Data View



This Diagram shows the tables in the database of the system. From this diagram a one and only one to many relationship is shown in between every table where the “id” represents a unique primary key of type integer.

The associations are given as follows:

* A unique user may have many reservations.
* A unique room may have many reservations.
* A unique reservation may have many users waiting in queue.

# 10. Performance

A login operation will be processed in 3 seconds or less. A successful reservation operation will be processed in 3 seconds or less.

# 11. Quality

The system is designed to meet the ISO 25010 quality standards which are defined as follow:

## 11.1 Portability

GetARoom system is developed using Java language which can be run on all kinds of platform. The system’s UI is accessible by any of the following web browsers (Chrome, Firefox, Chromium, Safari). Also it is possible to port it to any Mobile Platform with less than three person-week of effort. The system itself does not require installation for any of its functionalities.

## 11.2 Scalability

GetARoom system is hosted on a Jetty server which supports increasing its connection queue size for high incoming connections volumes and increasing the port range for high outgoing connection volumes. In case the expected maximum volume of users changes, we may modify these properties and update the server hardware accordingly.

## 11.3 Reliability, Availability:

GetARoom system is hosted on Jetty server, which provides a load balancing through clusters as a solution for the failure mechanism.

## 11.4 Security:

In terms of confidentiality, the user’s login ID will be public. When listing the reservations for a room, this user ID will be displayed in any reservations active for that room. This user ID will be provided to the user by the system (this cannot be modified), and will be said user’s student ID number.

## 11.5 Configurability:

GetARoom has a configurable option to set the number of allowable reservations per user.

# References

Jetty: <https://wiki.eclipse.org/Jetty/Howto/High_Load>