Software Architecture Document

Version 1.0

for

GetARoom

Prepared by

Student Name	**Student ID**	**Email**
Mihai Damsachin	27177895	mi_damas@encs.concordia.ca
Jesus Imery	27174276	j_imery@encs.concordia.ca
Meetaz Alshbli	25558840	mshbli75@hotmail.com
Samuel Dufresne	26992633	samuel.dufresne@live.com
Samuel Campbell	26457959	samuel.pcampbell@gmail.com
Sylvain Czyzewski	27066333	Sylvain.Czyzewski@gmail.com
Mark Snidal	26864317	mark.snidal@gmail.com

Instructor:	Dr. Constantinos Constantinides
University	Concordia University
Course:	SOEN 343: Software Architecture and Design I
Date:	November 23, 2016

Document history

Date	Version	Description	Author
11/14/2016	1.0	Part 1	Meetaz Alshbli
11/17/2016	1.0	Communication diagrams	Meetaz Alshbli Samuel Dufresne
11/18/2016	1.0	Parts #2- #3	Samuel Campbell
11/20/2016	1.0	Revision of SAD part 1	Mihai Damaschin
11/20/2016	1.0	Functional and non-functional requirements, size and performance, quality	Meetaz Alshbli Jesus Imery Samuel Dufresne
11/20/2016	1.0	Revision of all document Did parts 7, 8 ,10	Samuel Campbell
11/21/2016	1.0	Completed Data view Section	Samuel Campbell

Table of contents

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] C. Constantinides, An introduction to software development, 1st ed. 2016.

[2] C. Constantinides, Requirements analysis, 1st ed. 2016.

[3] C. Constantinides, Object - oriented design I, 1st ed. 2016.

[4] C. Constantinides, Object - oriented design II, 1st ed. 2016.

[5] C. Constantinides, Architectural styles, 1st ed. 2016.

[6] C. Constantinides, Architectural patterns, 1st ed. 2016.

[7] 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.

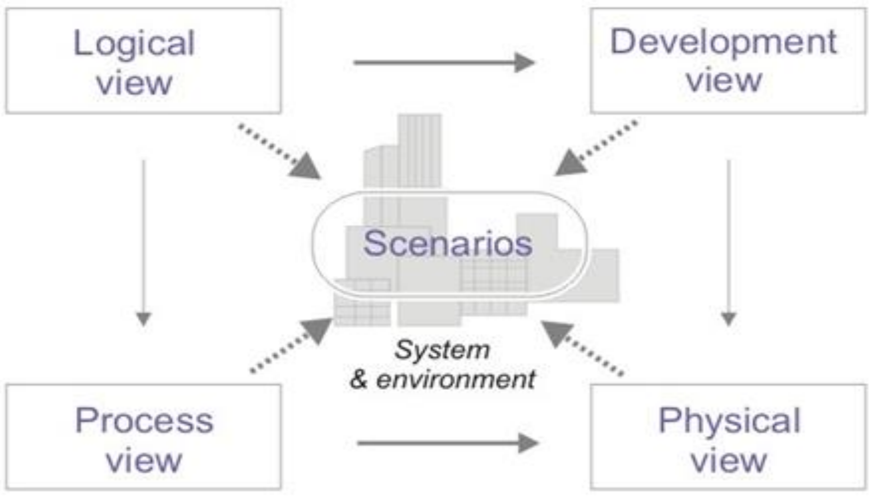


Figure 1 – 4 +1 Architectural view model

Reference: Ibryam, Bilgin. 4+1 View Model. 2016. Web. 23 Nov. 2016.

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 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

2.4 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

2.5 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	SAD -> 4.2.3

		multiple time slots	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

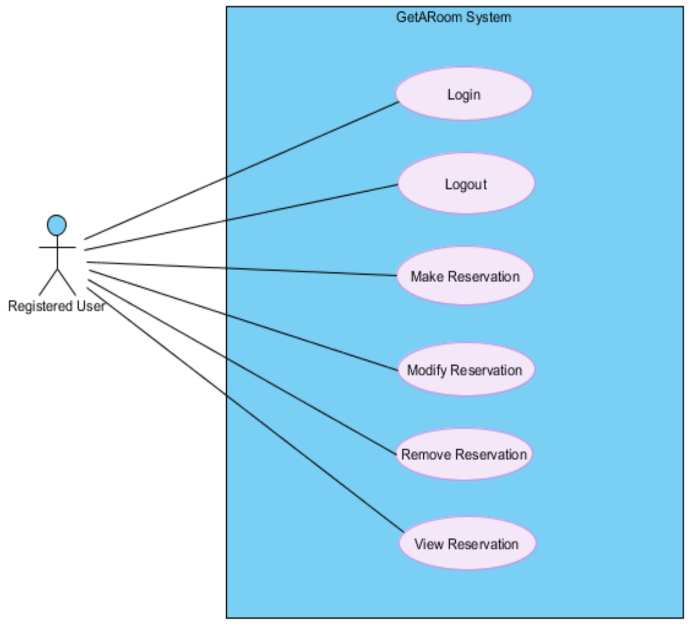


Figure 2 – Use Case Diagram

<u>4.1 Actors</u>

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.

<u>4.2 Brief Use Case Description</u>

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 all reservations of a room.

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.

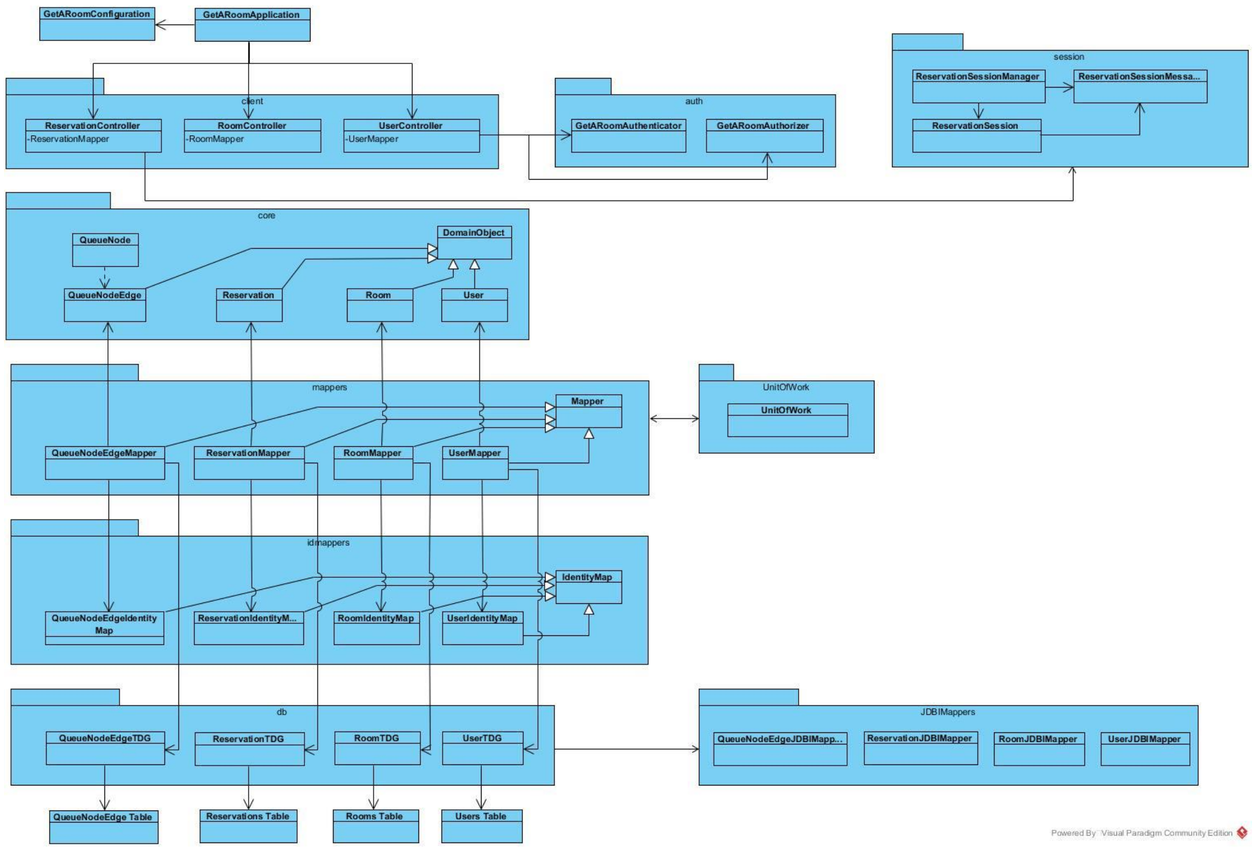


Figure 3 - Class Diagram

5.1 Tier diagram

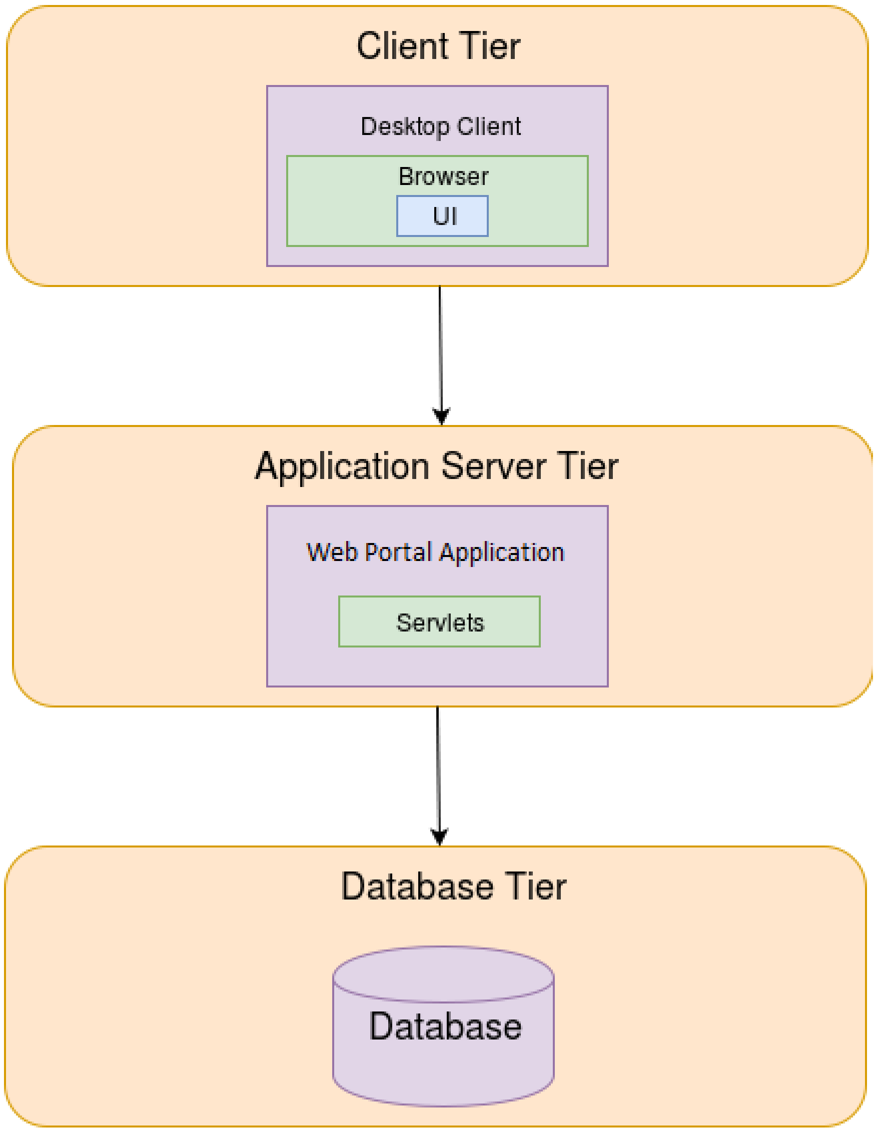


Figure 4 – Tier Diagram

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 diagrams and the system operations.

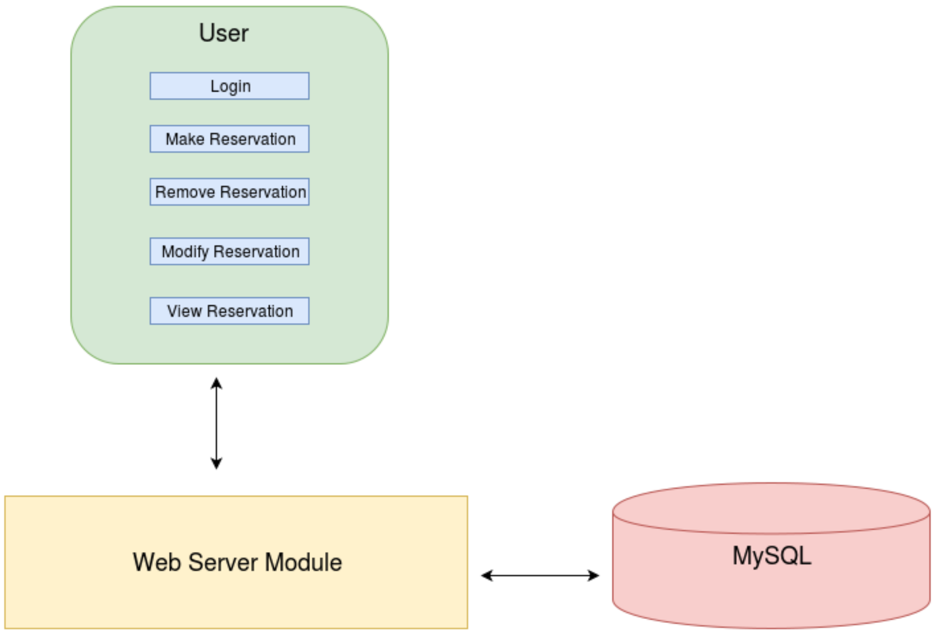
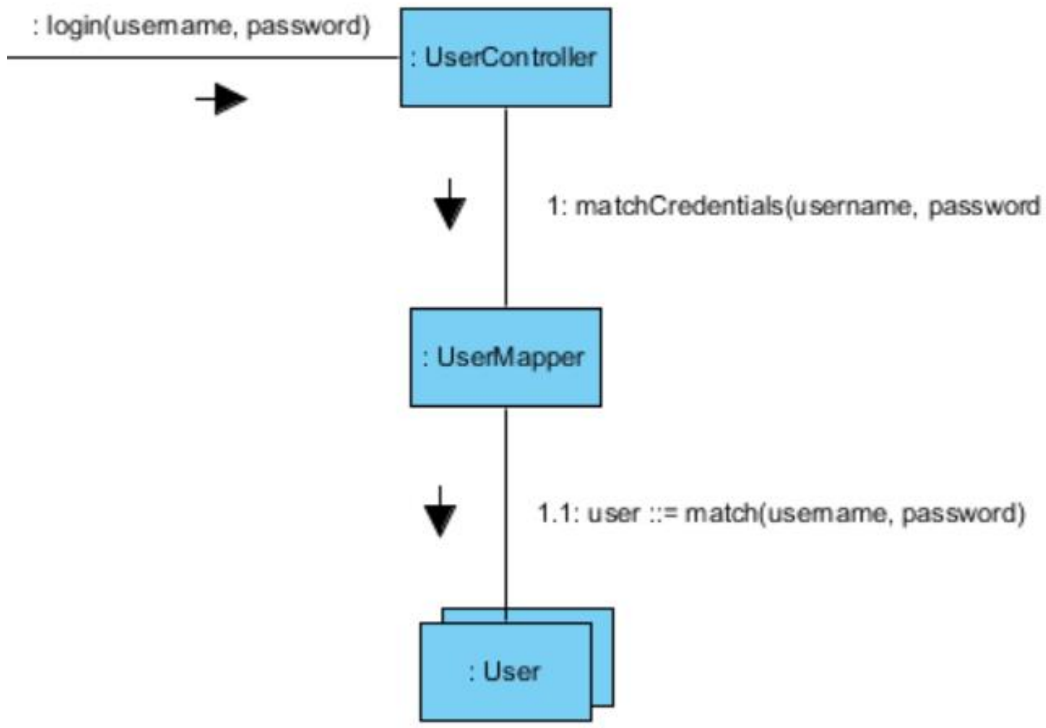


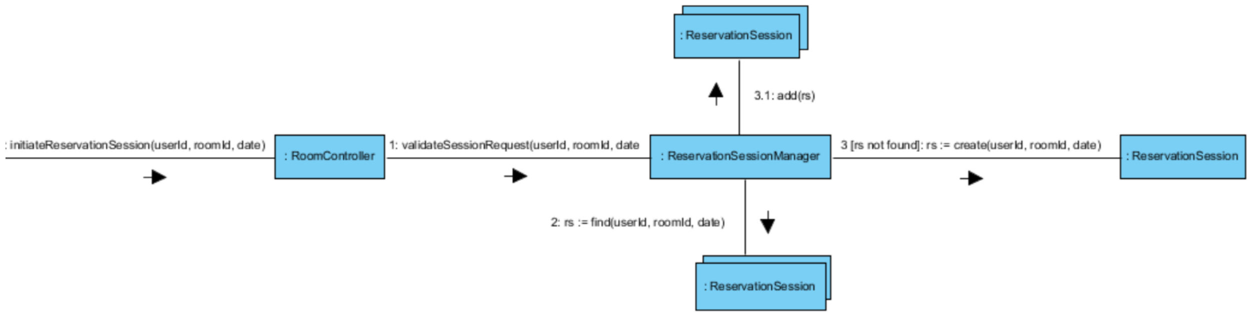
Figure 5 – Realization Diagram

5.4 Communication Diagrams



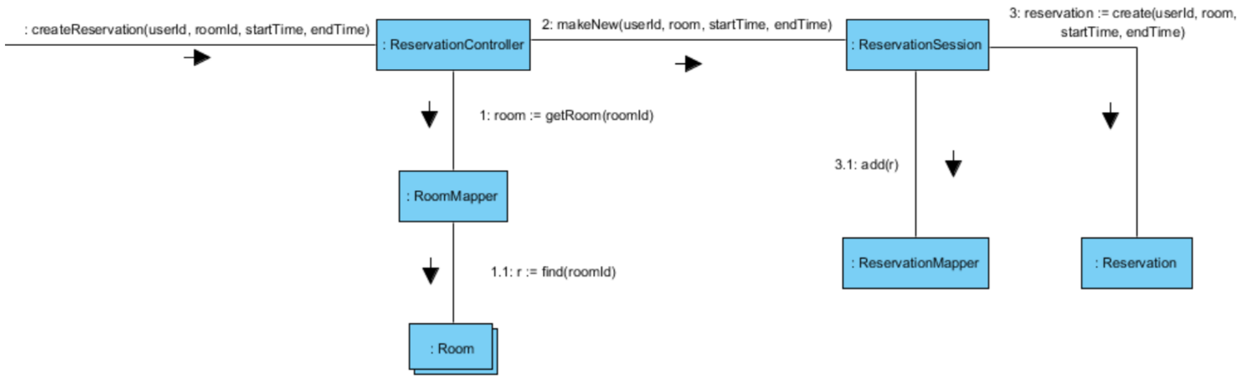
: login(usemame, password)
: UserController
1: matchCredentials(username, password
: UserMapper
1.1: user ::= match(usemame, password)
: User

Figure 6 - Communication Diagram (Login)



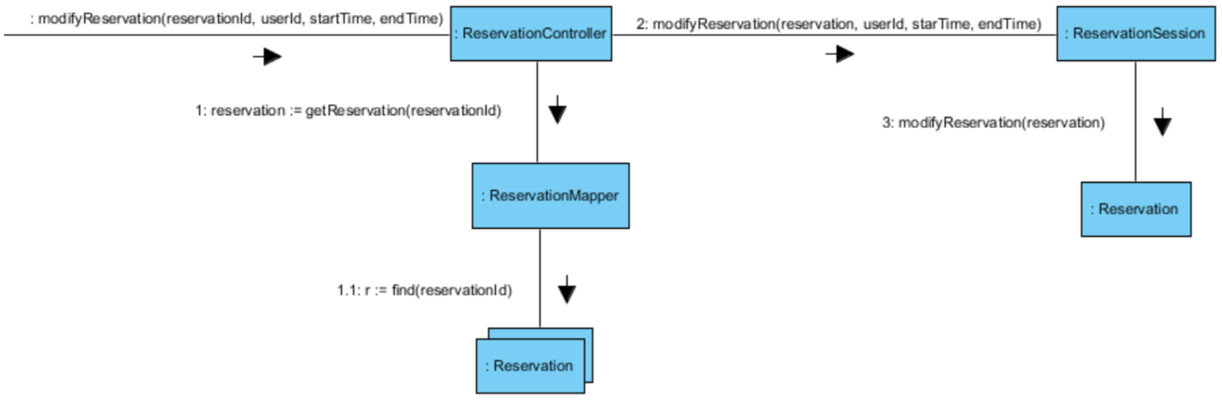
: ReservationSession
3.1: add(rs)
: initiateReservationSession(userId, roomId, date)
: RoomController
1: validateSessionRequest(userId, roomId, date
: ReservationSessionManager
3 [rs not found]: rs := create(userId, roomId, date)
: ReservationSession
2: rs := find(userId, roomId, date)
: ReservationSession

Figure 7 - Communication Diagram (Initiate Reservation Session)



: createReservation(userId, roomId, startTime, endTime)
: ReservationController
2: makeNew(userId, room, startTime, endTime)
: ReservationSession
3: reservation := create(userId, room, startTime, endTime)
1: room := getRoom(roomId)
: RoomMapper
3.1: add(r)
: ReservationMapper
: Reservation
1.1: r := find(roomId)
: Room

Figure 8 - Communication Diagram (Create Reservation)



: modifyReservation(reservationId, userId, startTime, endTime)
: ReservationController
2: modifyReservation(reservation, userId, starTime, endTime)
: ReservationSession
1: reservation := getReservation(reservationId)
3: modifyReservation(reservation)
: ReservationMapper
: Reservation
1.1: r := find(reservationId)
: Reservation

Figure 9 - Communication Diagram (Modify Reservation)

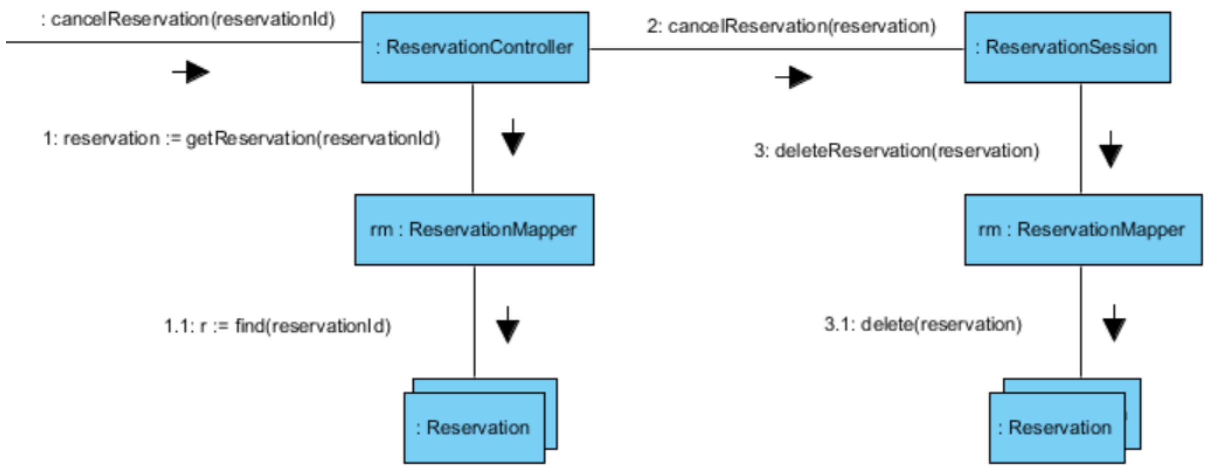

: cancelReservation(reservationId)
: ReservationController
2: cancelReservation(reservation)
: ReservationSession
1: reservation := getReservation(reservationId)
3: deleteReservation(reservation)
rm : ReservationMapper
rm : ReservationMapper
1.1: r := find(reservationId)
3.1: delete(reservation)
: Reservation
: Reservation

Figure 10 - Communication Diagram (Cancel Reservation)

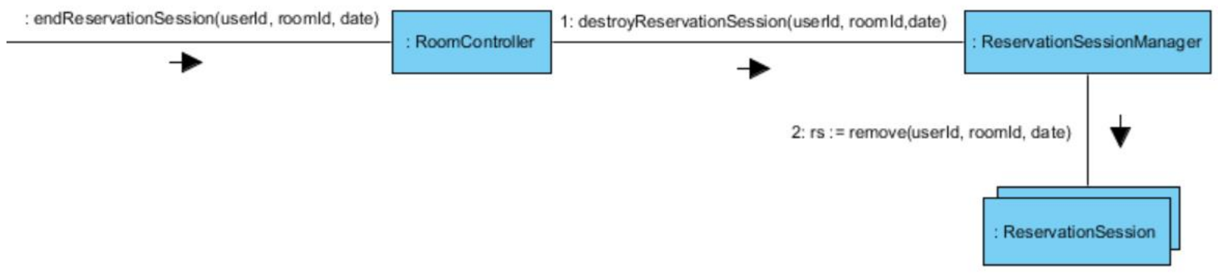

: endReservationSession(userId, roomId, date)
: RoomController
1: destroyReservationSession(userId, roomId,date)
: ReservationSessionManager
2: rs := remove(userId, roomId, date)
: ReservationSession

Figure 11 - Communication Diagram (End Reservation Session)

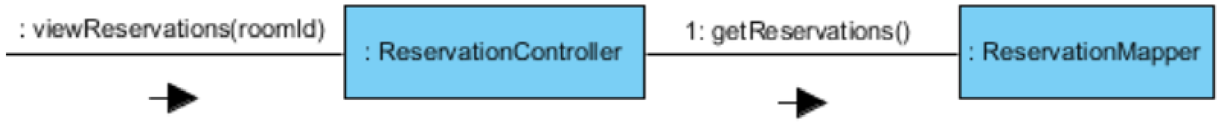


Figure 12 - Communication Diagram (View Reservations)

6. Development view

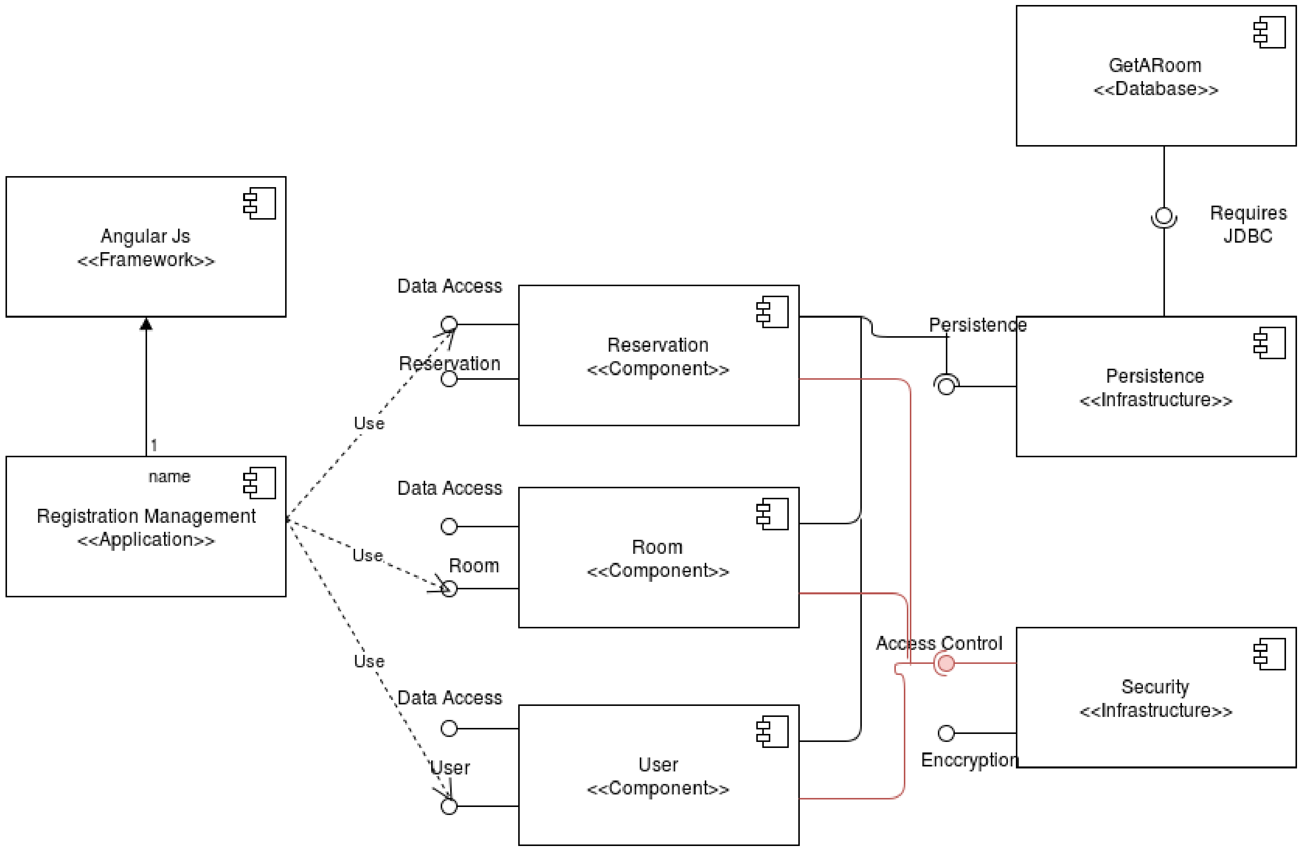


Figure 13 - Component Diagram

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.

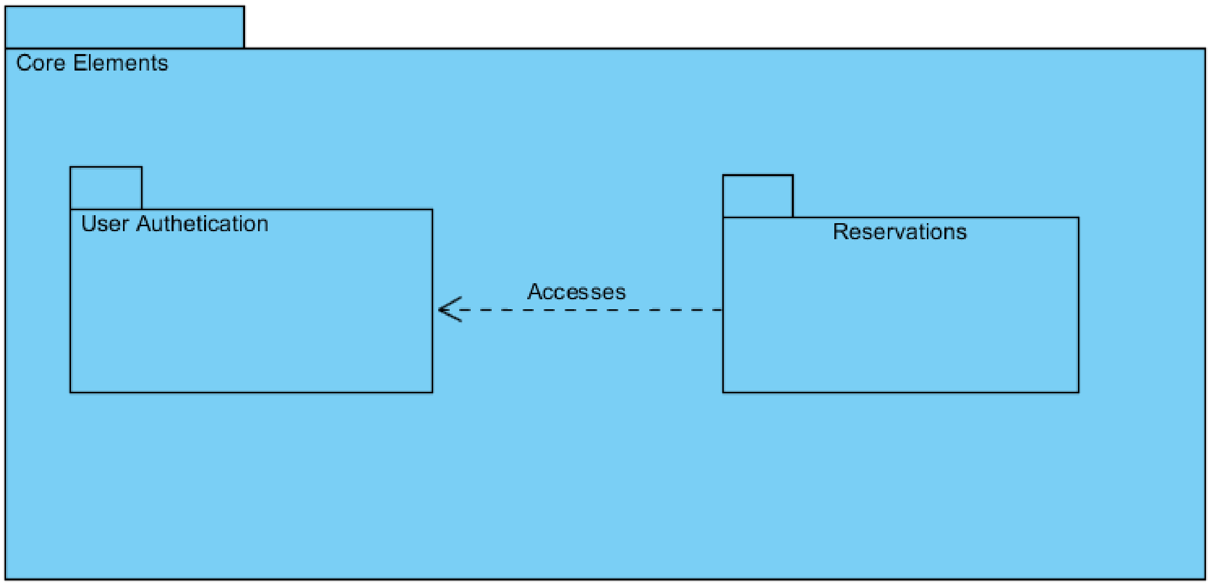


Figure 14 - Package Diagram

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.

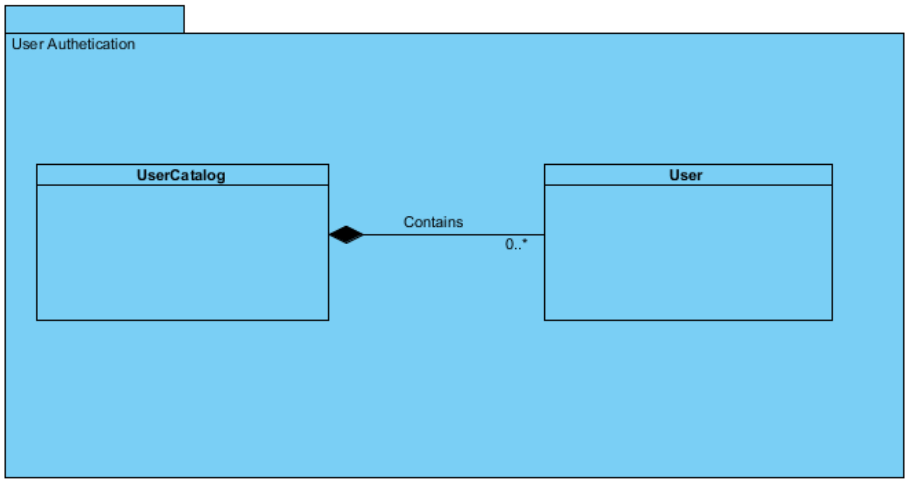


Figure 15

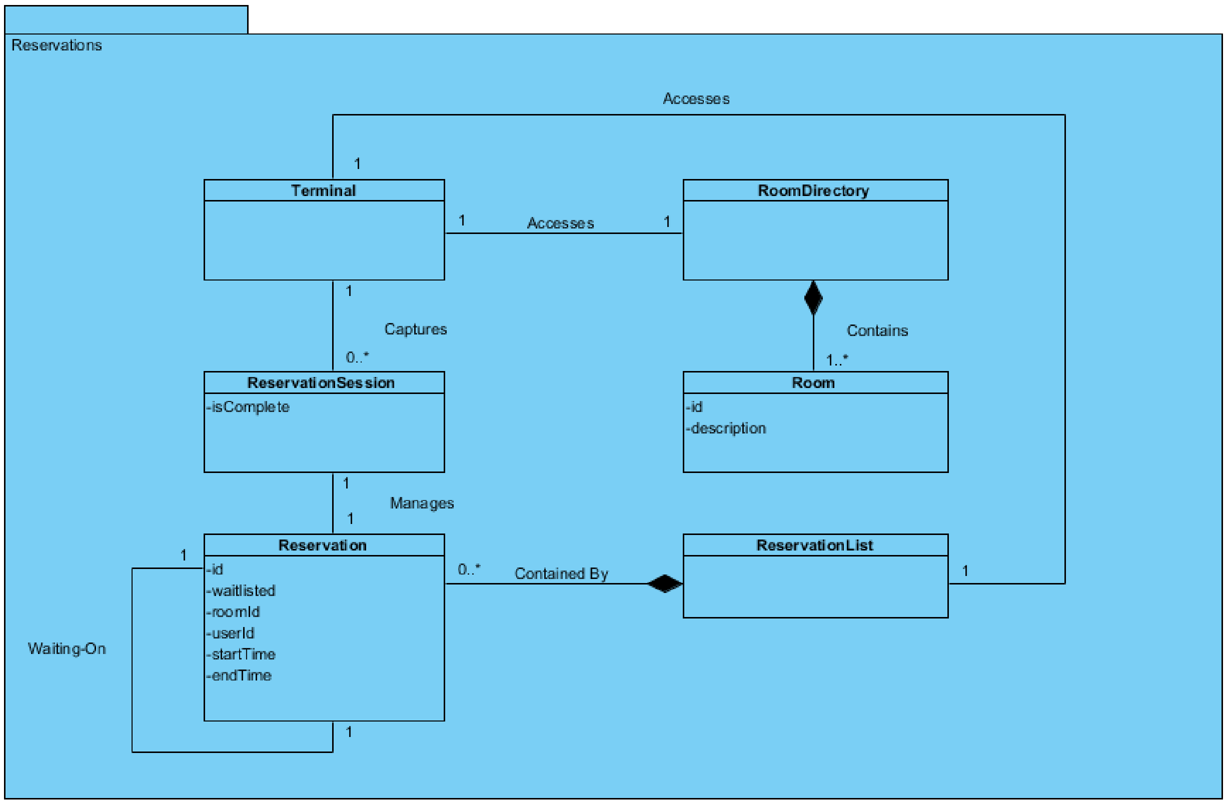


Figure 16

7. Physical view

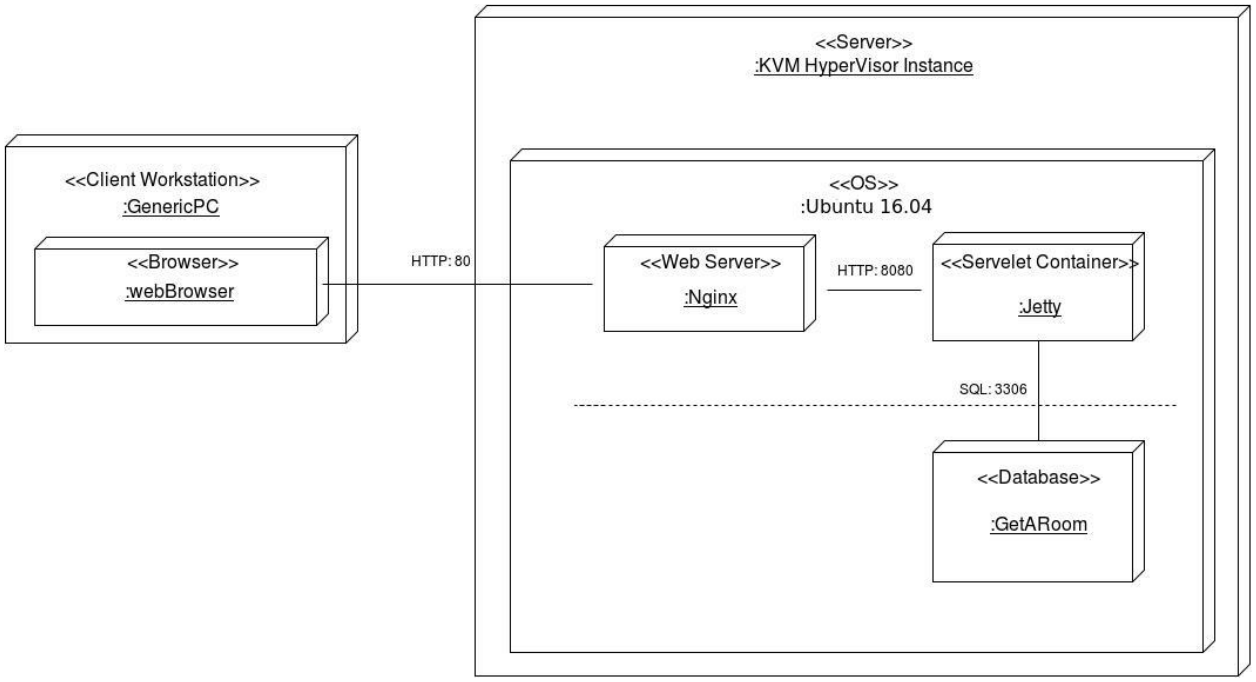


Figure 17 – Deployment Diagram

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.

8. Data View

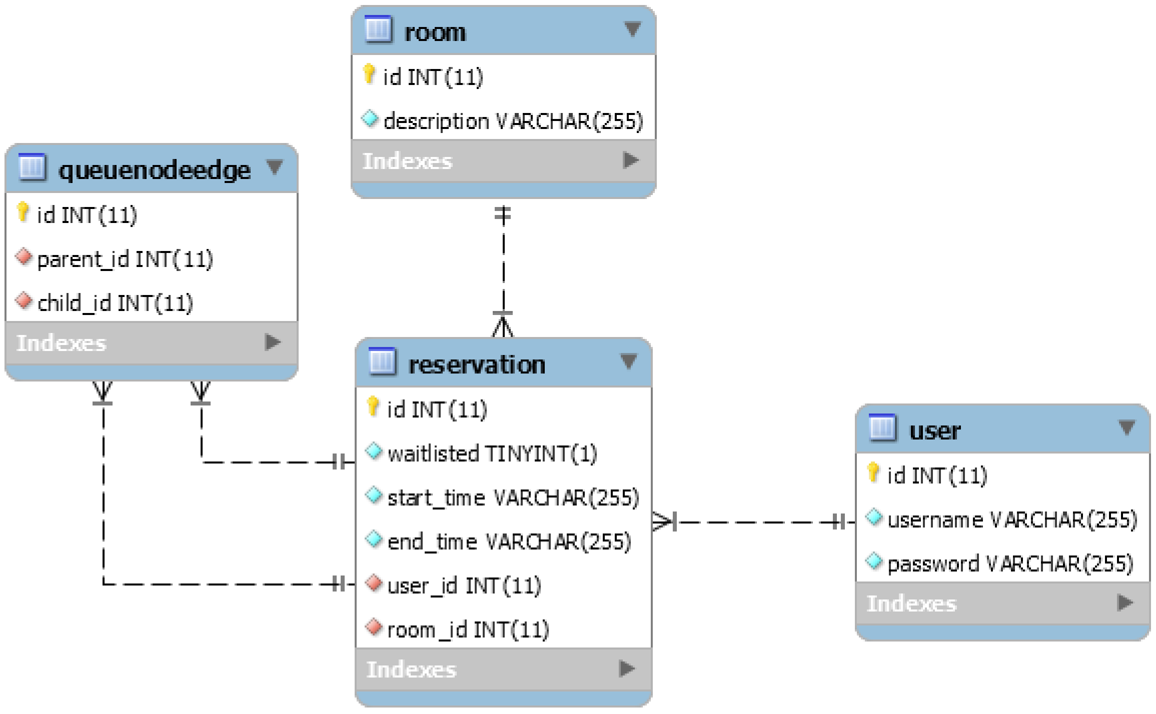


Figure 18 – ER Model

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.

9. Performance

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

10. Quality

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

10.1 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.

10.2 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.

10.3 Reliability, Availability:

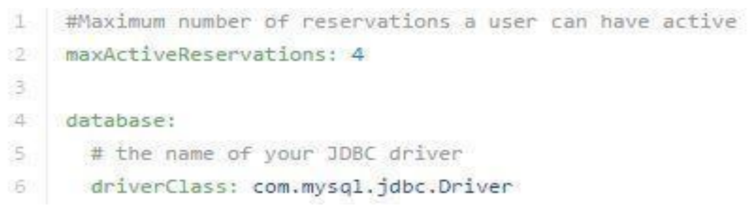
GetARoom system is hosted on Jetty server, which provides a load balancing through clusters as a solution for the failure mechanism.

10.4 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.

10.5 Configurability:

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



```
#Maximum number of reservations a user can have active
maxActiveReservations: 4

database:
  # the name of your JDBC driver
  driverClass: com.mysql.jdbc.Driver
```