**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 |  |

|  |  |
| --- | --- |
| 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 |

**Table of contents**

[1. Introduction](#_30j0zll)

[Purpose](#_1fob9te)

[Scope](#_3znysh7)

[Definitions, acronyms, and abbreviations](#_2et92p0)

[2. Architectural representation](#_tyjcwt)

[3. Architectural requirements: goals and constrains](#_4d34og8)

[Functional requirements (Use case view)](#_2s8eyo1)

[Non-functional requirements](#_17dp8vu)

[4. Use case view (Scenarios)](#_3rdcrjn)

[5. Logical view](#_26in1rg)

[Layers, tiers etc.](#_lnxbz9)

[Subsystems](#_35nkun2)

[Use case realizations](#_1ksv4uv)

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

[Reuse of components and frameworks](#_2jxsxqh)

[7. Process view](#_z337ya)

[12. Deployment (Physical) view](#_3j2qqm3)

[13. Data view (optional)](#_1y810tw)

[14. Size and performance](#_4i7ojhp)

[15. Quality](#_2xcytpi)

**List of figures**

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

# Introduction

This SAD document describes the high level and low level designs of the GetARoom system. It illustrates the conceptual instantiations of the objects in the system through class and interaction diagrams, provides a guideline on how to create and maintain the system and provides an overview of what the hardware and software platforms on which the system will be deployed as well as their essential interactions.

## 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. It is directed to software developers, using language and diagram conventions that this audience is prepared to interpret.

The document is separated in the following sections: a logical view to illustrate the object-oriented classes and the states that comprise our system, and a development view that includes a “4+1” architecture view by using a high level class diagram as well as all the interaction diagrams.

## 1.2 Scope

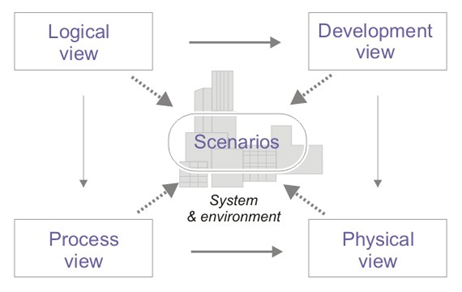
This software architecture document applies to the whole system design. It represents all the architectural designs along with the database structure. This illustration provides a good detailed understanding to the developers of the system dynamic behaviour, as well as a clear technical representation of the system to the stakeholders. The Software Architecture Diagram covers significant architectural decisions which influences the entirety of the web application.

## 1.3 Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | A registered Individual who 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 |

# Architectural representation

This document details the architecture using the views defined in the “4+1” architecture model [KRU41], but using the RUP naming convention. The views used to document the DTCPII tool application are:



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

1. **Development view**

Audience : Programmers

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

Related Artifacts: Implementation model, components

1. **Process view** :

Audience : Integrators

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

Related Artifacts: No specific artifacts

1. **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

1. **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

1. **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 Security**

The system shall be secured to protect users’ identity while creating reservations for rooms.

Basic security behavior:

* 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 .

**3.3 Persistence**

Data persistence shall be implemented using relational databases.

**3.4 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.

**3.5 Performance**

All operations related to room reservations shall be executed within 5 seconds.

## 4. Functional requirements (Use case view)

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| Use case(s) or scenario(s). | Name of case(s) or scenario(s). | Description on why this use case or scenario is relevant to the architecture. | Section number where this use case or scenario is addressed in this document. |
| Use Case UC1 | Login | Any user who wishes to utilize the website must login first to access any of its functions |  |
| Use Case UC3 | Make Reservation | User shall be able to reserve a room on a multiple time slots |  |
| Use Case UC4 | Remove Reservation | User shall be able to remove any of his existing reservation(s) from the database |  |
| Use Case UC6 | Modify Reservation | User shall be able to modify the time slot for any of his existing reservation(s) in the database |  |
| Use Case UC5 | View Reservation | User shall be able to view all of his existing reservation per room |  |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## Non-functional requirements

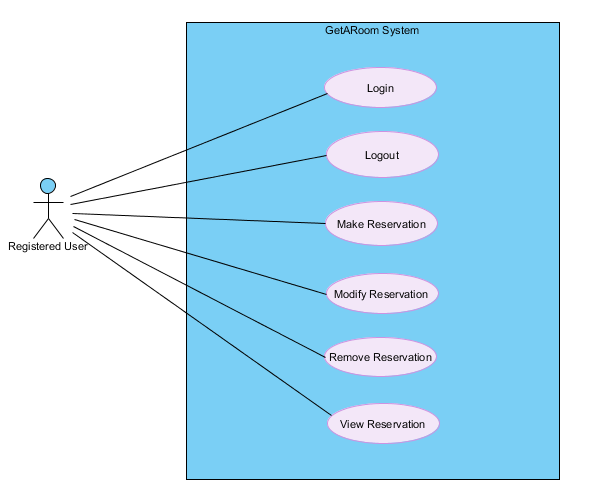
|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| e.g. Vision, SRS. |  | Description on why this requirement is relevant to the software architecture. | Section number where this requirement is addressed in this document. |
|  | Performance | The system average response time shall be lower than two seconds |  |
|  | Security | The system shall be secured to protect users’ identity while creating reservations for rooms.  Basic security behavior:   * 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 . |  |
|  | 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. |  |

# 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, make, 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**

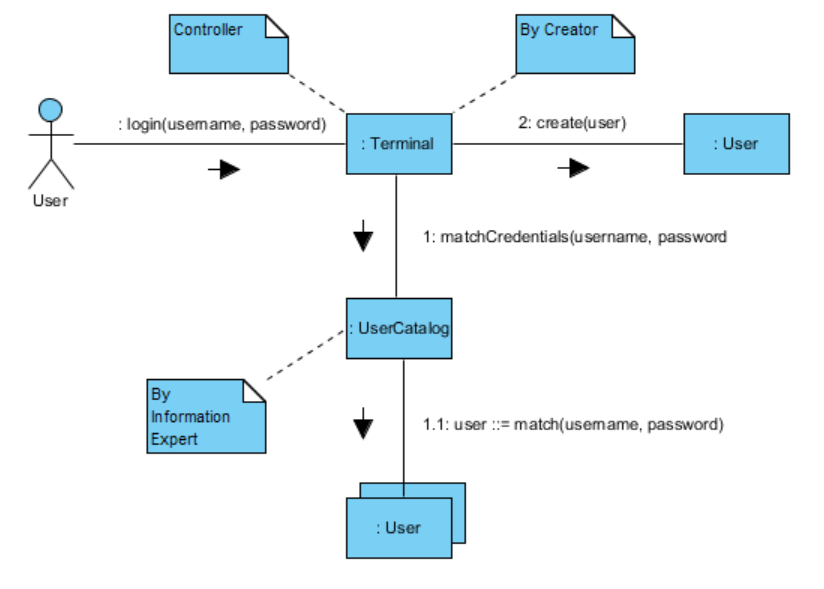
*Note : All actors in the following use cases are students from a college institution.*

**4.2.1 Login**

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

4.2.1.1 Communication Diagrams

4.2.1.1.1 Login

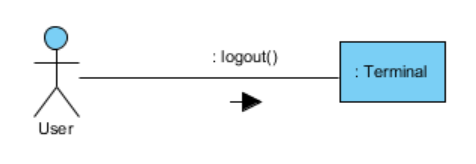


## 4.2.2 Logout

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

4.2.2.1 Communication Diagram

4.2.2.1.1 Logout

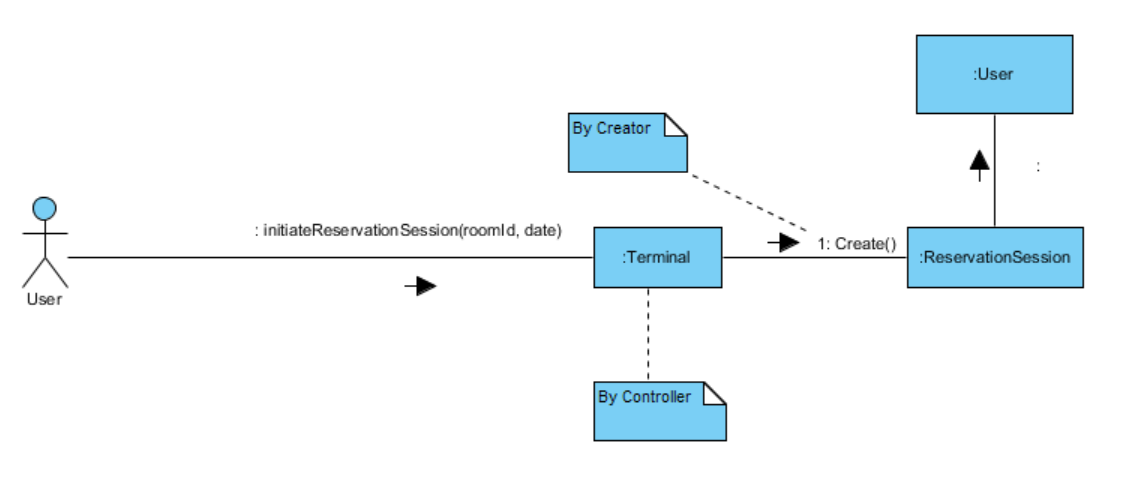


## 4.2.3 Make Reservation

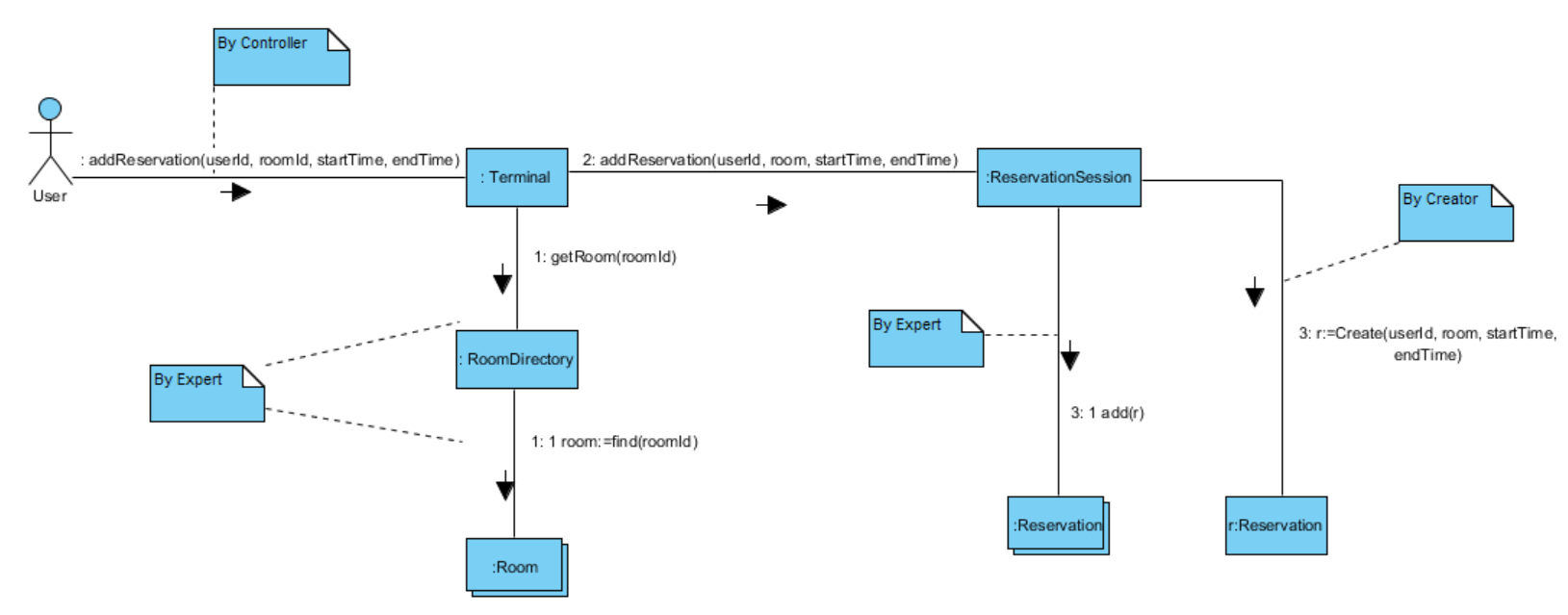
This use case allows a user to make a room reservation.

4.2.3.1 Communication Diagrams

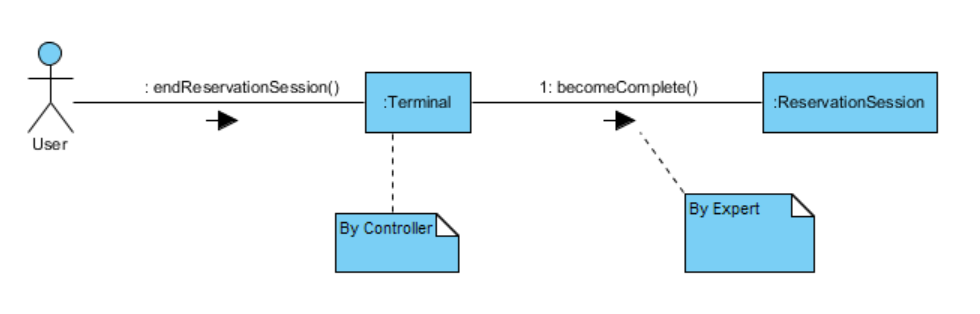
4.2.3.1.1 Initiate Reservation Session



4.2.3.1.2 Add Reservation



4.2.3.1.3 End Reservation Session

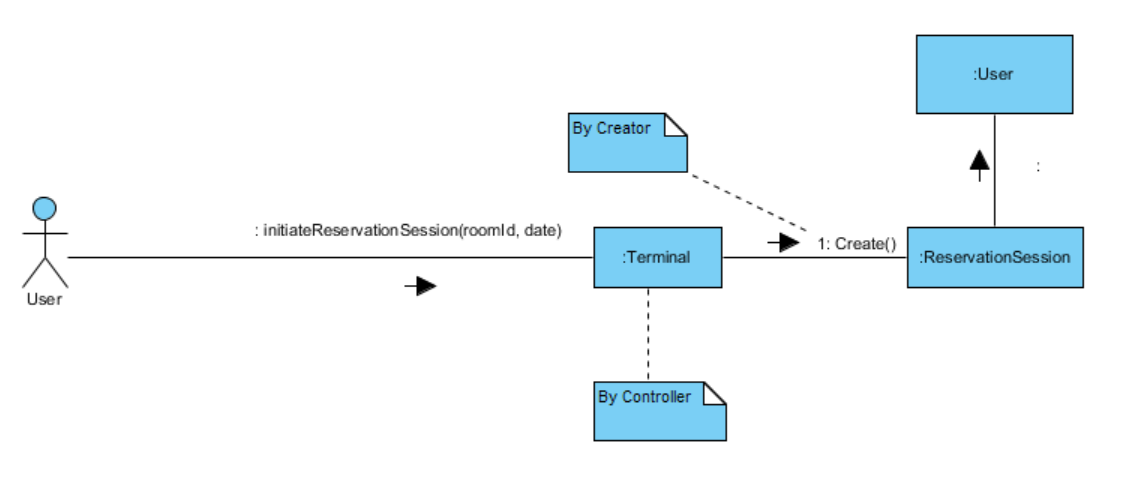


## 4.2.4 Remove Reservation

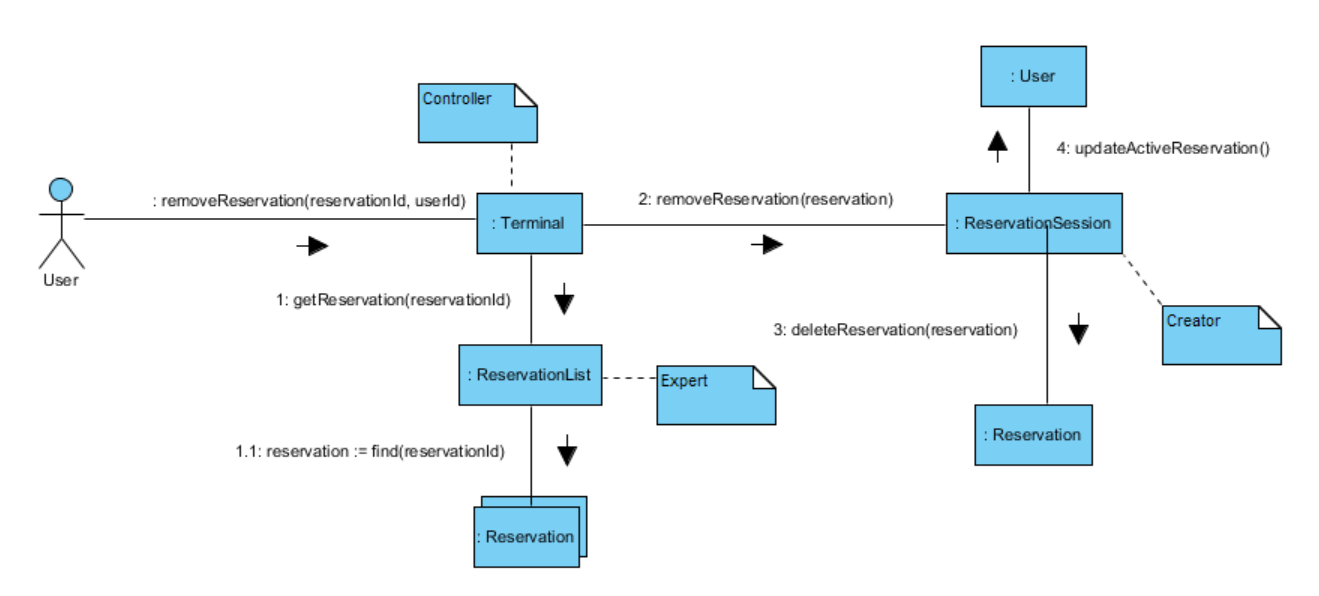
This use case allows a user to remove one of their existing reservations.

4.2.4.1 Communication Diagrams

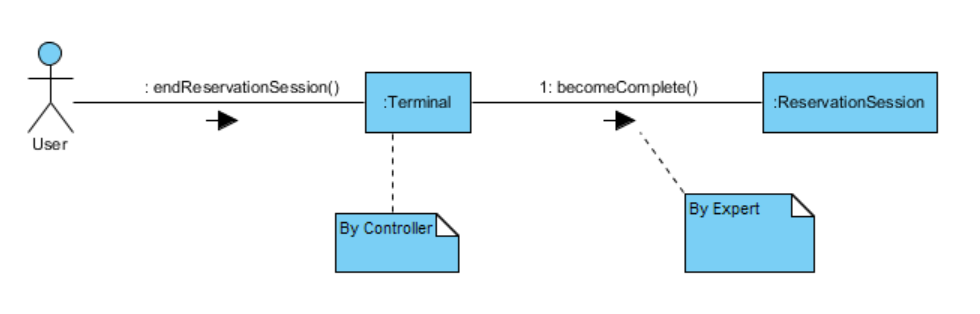
4.2.4.1.1 Initiate Reservation Session



4.2.4.1.2 Remove Reservation



4.2.4.1.3 End Reservation Session

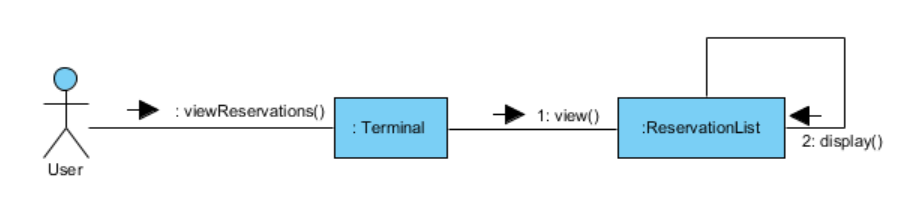


## 4.2.5 View Reservation

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

4.2.5.1 Communication Diagram

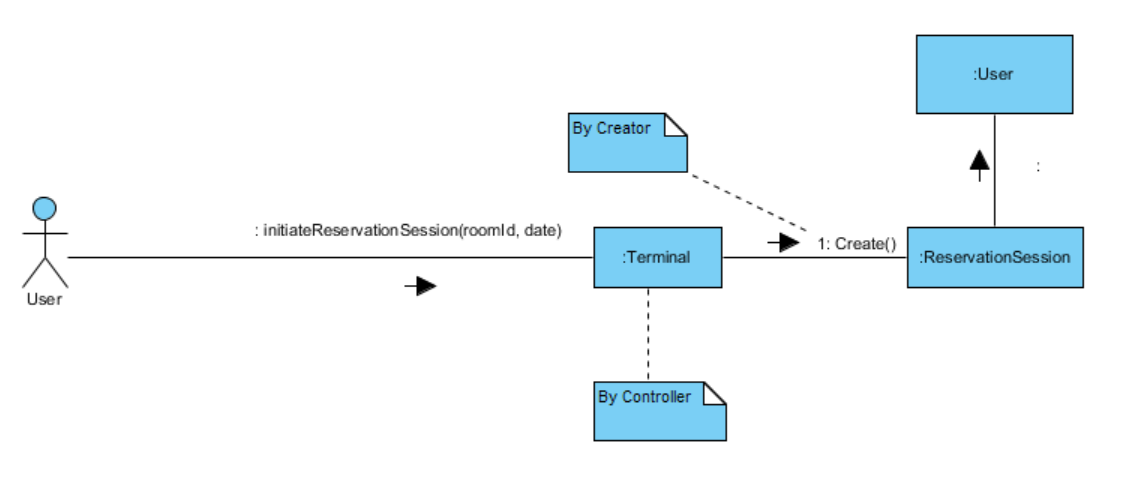
4.2.5.1.1 View Reservations



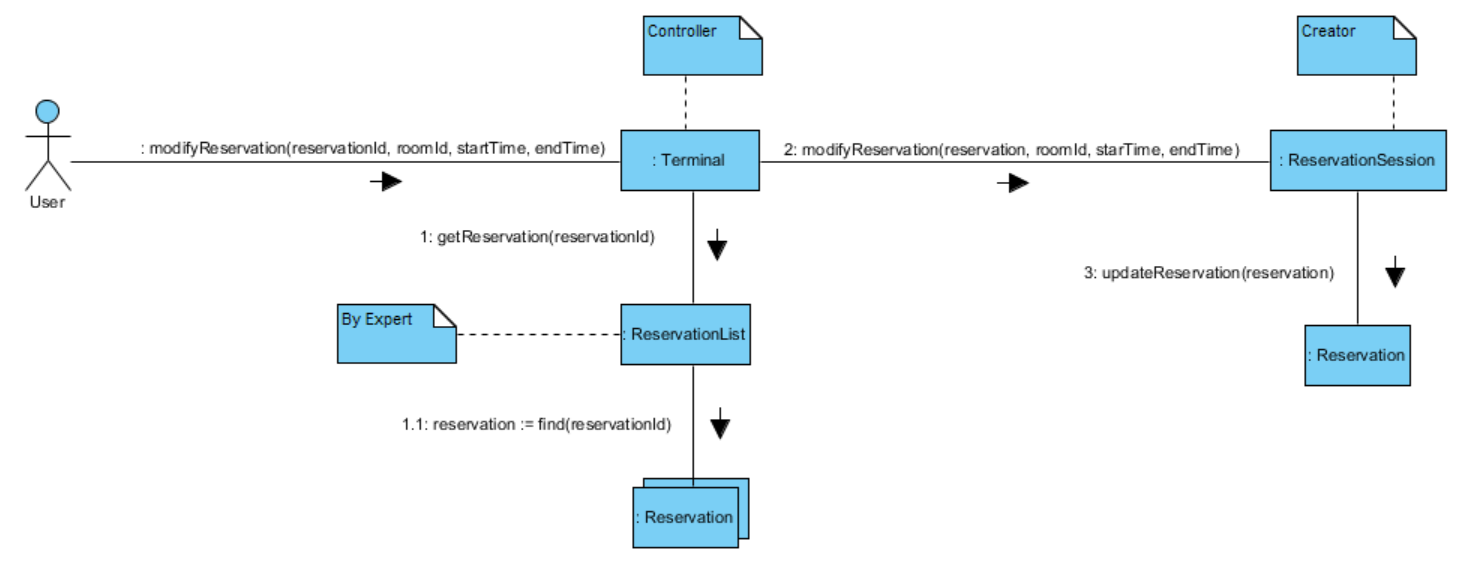
## 4.2.6 Modify Reservation

This use case allows a user to modify one of their existing room reservations.4.2.6.1 Communication Diagrams

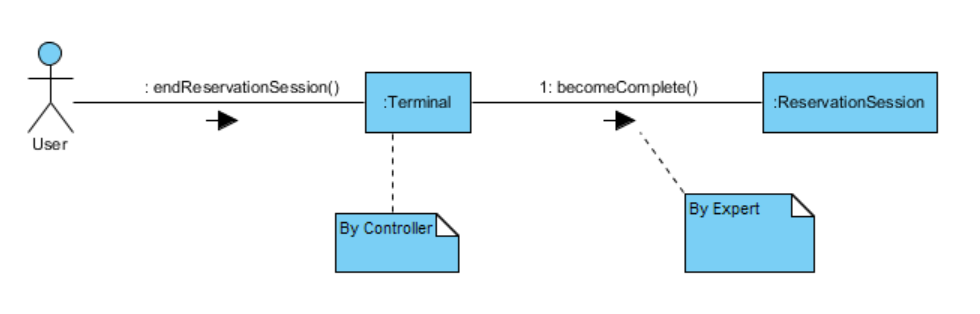
4.2.6.1.1 Initiate Reservation Session



4.2.6.1.2 Modify Reservation



4.2.6.1.3 End Reservation Session



# 8. Logical view

The GetARoom application is designed based of 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

## Layers, tiers etc.



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 translate 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 logic tier and then eventually sent to the user.

## Subsystems

The system consists of the CoreElements subsystem. The 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, delete operations to manage Reservation objects.

**Architecturally significant design packages**

## Use case realizations

The use realization diagram below shows how design elements provide functionalities identified in the critical use-cases.



<TEMPLATE>In this section you have to illustrate how use cases are translated into *UML interaction diagrams*. Give examples of the way in which the Use Case Specifications are technically translated into Use Case Realizations, for example, by providing a sequence-diagram. Explain how the tiers communicate and clarify how the components or objects used realize the functionality.</TEMPLATE>

# Development (Implementation) view



## Reuse of components and frameworks

**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.

**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.

**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.

# Process view

# Deployment (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. |

# Data view (optional)

# Size and performance

12.1 Size:

The system is expected to support a maximum of 100 concurrent users at one given time, with 500 users registered in the database. The system will support all of the 100 maximum expected concurrent users modifying their reservations, and subsequently the Reservations List data on the backend, as long as they are performing these modifications in different rooms, since each room can be modified by a single user at a time (a session is created that locks the room for that user).

12.2 Performance

A login operation will be processed in 3 seconds or less. A successful reservation operation will be processed in 10 seconds or less.

# Quality

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

**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, Sarafi). 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.

**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.

**Reliability, Availability:**

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

**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.

**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>