**Software Architecture**

**Document**

Version 1.0

for

The Force Awakens

Prepared by

|  |  |  |
| --- | --- | --- |
| Nicholas Burdet | 29613773 |  |
| Georges Mathieu | 26863477 |  |
| Olivier Cameron-Chevrier | 27228805 |  |
| Stefano Pace | 27454716 | Stefano.pace12@gmail.com |
| Julian Ippolito | 27419112 |  |
| Adam Trudeau-Arcaro | 27459157 |  |
| Joey Tedeschi | 27513062 |  |
|  |  |  |

|  |  |
| --- | --- |
| Instructor: | Dr. Constantinos Constantinides, P.Eng. |
| Course: | SOEN343: Software Architecture and Design I.  Section H, Fall term, 2016. |
| Date: | November 25th, 2016 |

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| November 12th, 2016 | 1.0 | Preliminary Merging of Documentation Sections | Stefano Pace |
| November 13th, 2016 | 1.1 | Continuation of Merging Documentation Sections | Stefano Pace |
| November 14th, 2016 | 1.2 | Continuation of Merging Documentation Sections | Stefano Pace |
|  |  |  |  |

Table of contents

[1. Introduction 5](#_Toc437439092)

[Purpose 5](#_Toc437439093)

[Scope 5](#_Toc437439094)

[Definitions, acronyms, and abbreviations 5](#_Toc437439095)

[2. Architectural representation 6](#_Toc437439096)

[3. Architectural requirements: goals and constrains 8](#_Toc437439097)

[Functional requirements (Use case view) 8](#_Toc437439098)

[Non-functional requirements 8](#_Toc437439099)

[4. Use case view (Scenarios) 11](#_Toc437439100)

[5. Logical view 12](#_Toc437439101)

[Layers, tiers etc. 12](#_Toc437439102)

[Subsystems 12](#_Toc437439103)

[Use case realizations 12](#_Toc437439104)

[6. Development (Implementation) view 13](#_Toc437439105)

[Reuse of components and frameworks 13](#_Toc437439106)

[7. Process view 14](#_Toc437439107)

[8. Deployment (Physical) view 15](#_Toc437439108)

[9. Data view (optional) 16](#_Toc437439109)

[10. Size and performance 17](#_Toc437439110)

[11. Quality 17](#_Toc437439111)

**List of figures**

[Figure 1: Layered Architecture Design for the System. 6](#_Toc353551167)

# Introduction

## Purpose

This section defines the role or purpose of the Software Architecture Document and briefly describes the structure of the document. Identify the intended audience for the document is identified, with an indication of how they are expected to use the document.

This Software Architecture Document (or SAD) delivers a complete and well-structured overall architecture of the Room Reservation System. This document contains multiple different architectural representations in order to illustrate the different system components in addition to the different architectural choices that the user can exert on the system itself. This document is addressed to the stakeholders (the students and the staff), the developers, the professors and the correctors who will be evaluating the Room Reservation System. This SAD can be used as a reference in order to fully understand the system from the different architectural representations.

## Scope

A brief description of what the Software Architecture Document applies to; what is affected or influenced by this document.

The scope of this SAD document is to represent the different architectures of the web-based Room Reservation System. This document is entirely based on the Room Reservation System and depicts different architectural views of the system itself and on the system when the user is using its different functionalities.

## Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Architecture Document. This information may be provided by reference to the project’s Glossary. For example:

|  |  |
| --- | --- |
| **Word** | **Definition** |
| Corrector | Person responsible to evaluate the project and its components in order to assign a result. |
| Software Architecture Document | A document that captures the bigger structures of a software system, and it deals with how multiple software processes cooperate to carry out their tasks. |
| Database | Collection of all the information monitored by this system. |
| Interface | Computer hardware or software designed to communicate information between hardware devices, between software programs, between devices and programs, or between a device and a user. |
| Stakeholder | A person or group that has an investment, share, or interest in something, as a business or industry. |
| User | Person who uses the system. |

|  |  |
| --- | --- |
| **Word** | **Acronym** |
| Software Architecture Document | SAD |
| Unified Modeling Language | UML |

## References

Provide a list of all documents referenced in the SRS.

[1] C. Constantinides, "SAD", 2016.

[2] "The Definition of Stakeholder", *Dictionary.com*, 2016. [Online]. Available: http://www.dictionary.com/browse/stakeholder?s=t. [Accessed: 01- Nov- 2016].

[3] "Software Architecture and Design Tutorial", *www.tutorialspoint.com*, 2016. [Online]. Available: https://www.tutorialspoint.com/software\_architecture\_design/. [Accessed: 02- Nov- 2016].

# Architectural representation

Describe the top-level architectural style of the system and the view model you will adopt. Additionally, describe what each individual view will provide. Many enterprise software systems are modelled using the 4+1 view illustrated in Figure 1.

The top-level architectural style being used for this system is “Layered Architecture”. In layered architecture, the system is separated into several levels, in which related functionalities are grouped together and associated to a single layer. Each layer provides its services to the layers above it, thus the lowest level would represent core services likely to be used throughout the entire system. In the system being created, 3 main layers were defined: The user interface, application logic, and database access/network communication. The lowest layer, Database access, serves as a way to provide information to all layers above it (Application logic layer in order to access data and perform calculations, User interface layer to display information to the user). The next layer, the application logic level, uses data pulled from the database in order to fulfill request generated by the user (e.g.: Viewing their reservations, making a new reservation). Additionally, any manipulation of information would occur at this level. The topmost layer, the user interface level, serves as the primary method for interacting with the system as a whole. No logic occurs at this level, but instead allows for the generation of signals. These signals notify the application logic layer of what needs to be done, which then accesses the database layer to pull the required information.

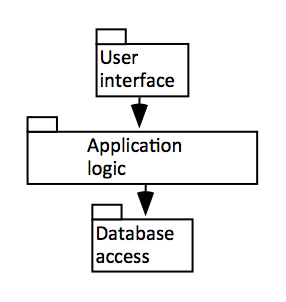


Figure 1: Layered Architecture Design for the System

The view model being adopted is the 4+1 view, in which the system is described from the point of view of multiple different stakeholders. 5 main views are presented below: Logical, Development, Process, Physical, and Use Case. In addition to their appropriate visual representations (in the form of various diagrams), each view will include a description of the purpose it serves with regards to the system as a whole, and who they are appropriate for.

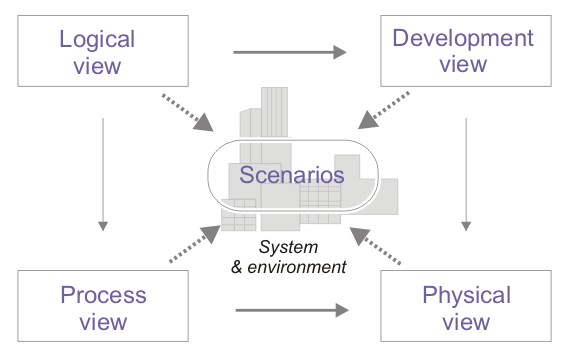


Figure 2: From SAD doc

# Architectural requirements: goals and constrains

Requirements are already described in SRS. In this section describe *key* requirements and constraints that have a significant impact on the architecture.

The key goals of the architecture:

* Mutual Exclusion: only one user can make a reservation for a room at a specific time. Any amount of people can be viewing reservations.
* Safety: Reservations and user info is not overwritten or modified by other users.
* Fairness: Users can only make three reservations a week, for a time limit of three hours each. Have a waiting queue for people who want to make reservations that does not prioritize any user.

Key constraints to the architecture are:

* Coupling: The system must not make too many calls to the database. Units of work will be used to group calls and execute them all at the same time.

## Functional requirements (Use case view)

Refer to Use Cases or Use Case scenarios which are relevant with respect to the software architecture. The Use Cases referred to should contain central functionality, many architectural elements or specific delicate parts of the architecture.

The overview below refers to architecturally relevant Use Cases from the Use Case Model (see references).

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |

## Non-functional requirements

Describe the architecturally relevant non-functional requirements, i.e. those which are important for developing the software architecture. Think of security, privacy, third-party products, system dependencies, distribution and reuse. Also environmental factors such as context, design, implementation strategy, team composition, development tools, time to market, use of legacy code may be addressed.

Usually, the non-functional requirements are already in place and can be referenced here. This document is not meant to be the source of non-functional requirements, but to address them. Provide a reference per requirement, and where the requirement is addressed.

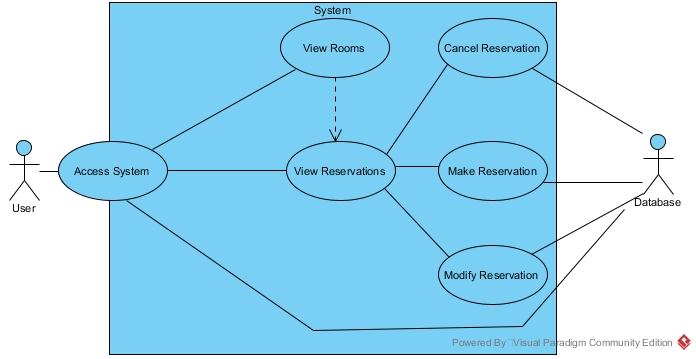
|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| e.g. Vision, SRS. | Name of requirement. | Description on why this requirement is relevant to the software architecture. | Section number where this requirement is addressed in this document. |

# Use case view (Scenarios)

**Use case view** (also known as Scenarios): Audience: all the stakeholders of the system, including the end-users. The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. Related Artefacts: **Use-Case Model**.

The scenarios (or functional view) represent the behaviour of the system as seen by its actors. Use case scenarios describe sequences of interactions between actors and the system (seen as a black box) as well as between the system and external systems the *UML use case diagram* is used to capture this view.

|  |  |
| --- | --- |
| Actor | Goal |
| User/Student | -Access system  -View directory of rooms  -Make a reservation  -Cancel a reservation  -Modify a reservation  -View list of reservations in the registry (view availability of rooms) |



* The site will be privately available to only selected Students. These users will be sent the link directly to access the website.
* The reservation system must be available to use from any commonly used web browser application (and relatively recent version).
* Users will only be allowed to make a reservation in another room after already making one, once every 5 minutes.
* Users will only be allowed to add themselves to a single waiting list for a specific room.
* Users cannot exist in a single waiting list more than once.
* Users cannot maintain more than 3 reservations for a single week period.
* Users may hold a position in any number of waiting lists for reservations, so long as they have not already reached the maximum number of reservations for the week (currently 3).
* Users holding the top position in a waiting list queue will be given the reservation for that room and timeslot if the current holder cancels their reservation.
* Users given a reservation for a room will be removed from all other waiting lists if they have reached the maximum allotted reservations for the week.

*Note: Timeframe and numbers are subject to change upon negotiations*

*Note: We will only be supporting the following commonly used web browsers: Google Chrome, Firefox, Internet Explorer, Opera, Microsoft Edge.*

# Logical view

**Logical view**: Audience: Designers. The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include **Class diagram**, and **interaction diagrams** (**communication diagrams**, or **sequence diagrams**).

The logical view captures the functionality provided by the system; it illustrates the collaborations between system components in order to realize the system's use cases. Describe the architecturally significant logical structure of the system. Think of decomposition in tiers and subsystem. Also describe the way in which, in view of the decomposition, Use Cases are technically translated into Use Case Realizations.

## Layers, tiers etc.

Describe the top-level architecture style. Deploy a *UML class diagram*.

## Subsystems

Describe the decomposition of the system in subsystems and show their relation.

**Architecturally significant design packages**

Desribe packages of individual subsystems that are architecturally significant. For each package nclude a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

## Use case realizations

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.

# Development (Implementation) view

**Development view** (also known as Implementation view): Audience: Programmers. The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the **Package diagram**.

The development (or implementation) view describes the components used to assemble the system. Use a *UML component diagram* to capture this view.

## Reuse of components and frameworks

Describe any third-party or home-made components and frameworks that will be reused.

The physical, or deployment view, presents the system from the perspective of a system engineer. It is concerned namely with the software components used, and their physical connections. As the system being designed is constructed according to Layered architecture, there are 3 mainly components to consider: The user themselves (user interface), the Reservation system and its’ sub-components (application logic), and the database (database access).

At the user level, one can provide a request to add, remove, or modify an existing reservation. These options are all instances of user control in action, and lead to said calls being received by the application logic layer. Within this layer, the main software component, the reservation management system, is found. Within this system, the three components Student, Room, and Reservation serve to process and apply all changes requested by the User. These components contain all pertinent classes and attributes relating to their namesake, and as such all logic take places through their usage (normally through the manipulation of objects). This will require a call to the database layer either to access stored information found within said database, or otherwise to simply make changes to the data currently being stored. At the database layer, all logical changes made are stored for further access. Upon being updated, this change is propagated through to the upper levels when required (i.e.: When another update is required, such as to update the user’s view or make further logic changes).

A deployment diagram of the connections between the various components found in the system can be seen below.

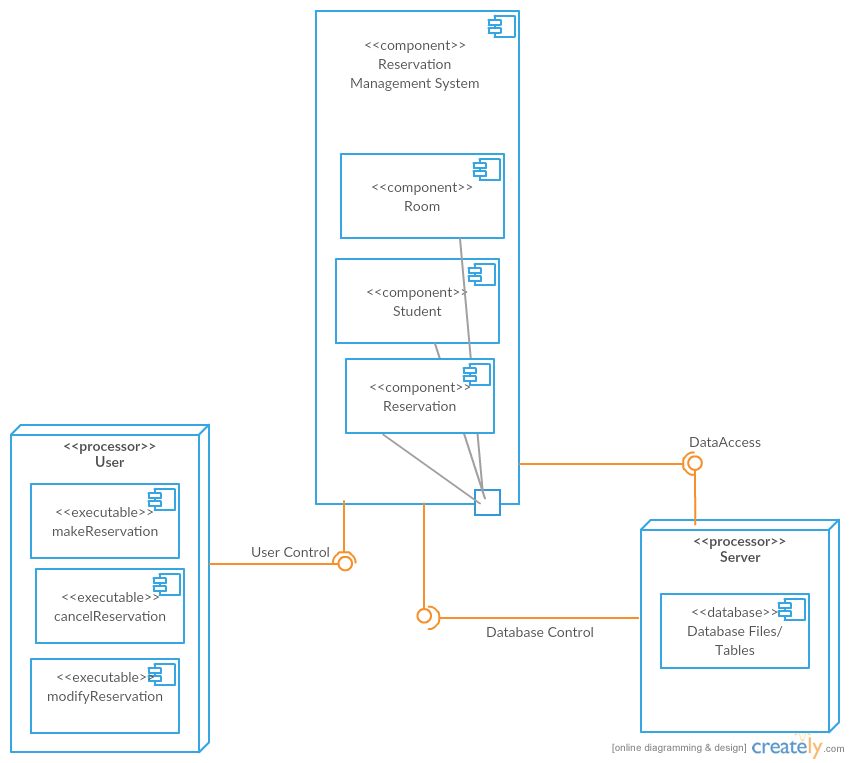


Figure 2: Deployment Diagram of Room Reservation System

# Process view

**Process view**: Audience: Integrators. The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the **Activity diagram**.

The process view illustrates the system's processes, focusing on the runtime behavior of the system. The view illustrates parallelism and concurrency. Deploy a *UML activity diagram* to capture this view.

The goal of a process view is to visually represent the critical scenarios as a sequence of events through the use of UML Activity Diagrams. These events include both user and system operations, and can contain conditional paths that lead to situational events. The critical scenarios of the room reservation system are as follows:

1. Logging in to the system (Figure 1)
2. Creating a reservation (Figure 2)
3. Cancelling a reservation (Figure 3)
4. Modifying a reservation (Figure 4)
5. Changing profile details (Figure 5)

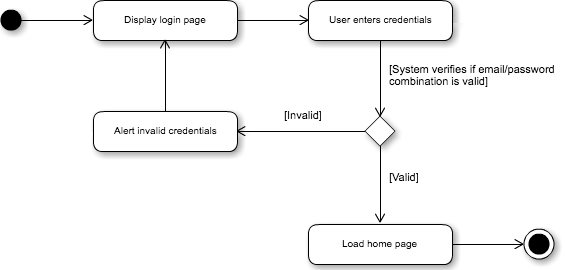


Figure 3: Authentication

This activity diagram represents the path for logging into the room reservation system. When the site is first accessed, the user will be greeted with a login page where they are prompted to enter their credentials (email address and password). The user enters their credentials, after which the system verifies the database to ensure that the credentials correspond to an existing account. If the credentials are valid, then the user is logged in and redirected to the site’s home page. On the other hand, if the credentials are invalid, then a message will appear to inform the user, after which they will be prompted to enter their credentials again.

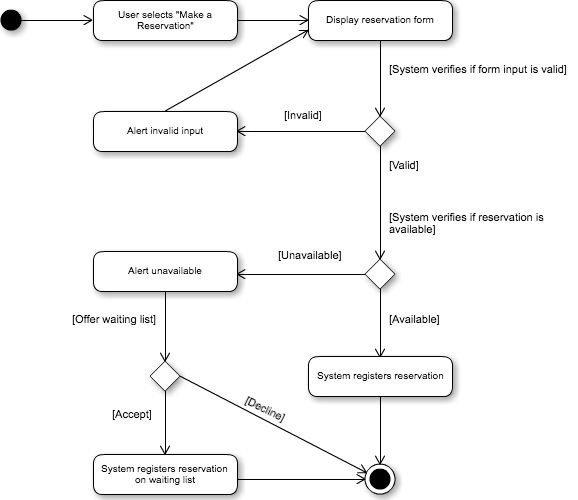


Figure 4: Creating a Reservation

This activity diagram represents the path for creating an available reservation or creating a reservation in the waiting list. On the home page, the user selects “Make a Reservation” from the toolbar. A form with all required information will appear, in which the user must define a title, description, start time/date, end time/date, and room. First, the system verifies if these inputs are valid. Examples of invalid inputs would be any blank field, or one that surpasses the character limit. If the input if invalid, then the user will be informed and told to try again.

Next, the system verifies if the selected time/date/room combination is available, or if the user already has 3 existing reservations for that week. If the combination is available, then the system registers the reservation associated with the student. However, if it is unavailable, the system alerts the user and offers to place this reservation on the waiting list. If the user accepts, then the system registers the reservation associated with the student and notes that it is waitlisted. If the user declines, then the reservation process is cancelled entirely.

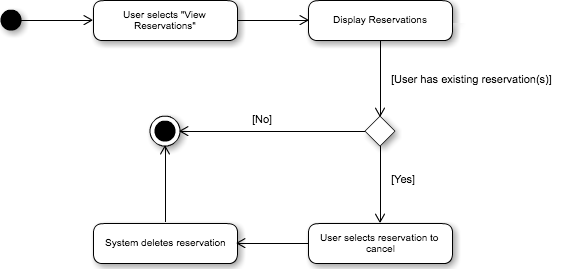


Figure 5: Cancel Reservation

This activity diagram represents the path for cancelling an existing reservation associated with the authenticated user. First, the user must select “View Reservations” from the toolbar. The system will display all existing reservations, providing the option to cancel them. Evidently, if the user has no existing reservations, this option will not be displayed. If the user selects a reservation to cancel, then it will be removed from the system and disassociated from the user.

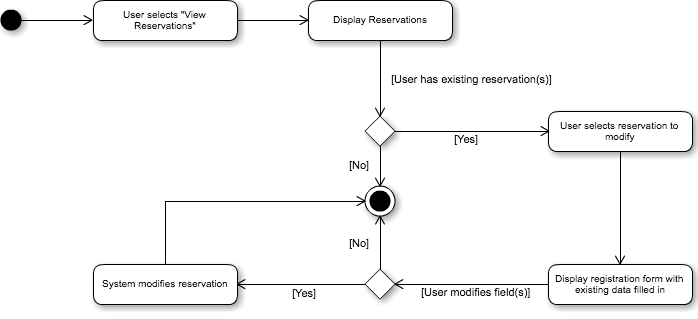


Figure 6: Modify Reservation

This activity diagram represents the path for modifying an existing reservation associated with the authenticated user. First, the user must select “View Reservations” from the toolbar. The system will display all existing reservations, providing the option to modify them. Evidently, if the user has no existing reservations, this option will not be displayed.

If the user selects a reservation to modify, then a form will appear with all the reservation information fields filled in. These fields are all modifiable, so the user can change any information as required. If no fields are modified, then the existing reservation is left alone. If any fields are modified, then the new reservation information replaces the respective old one in the system.

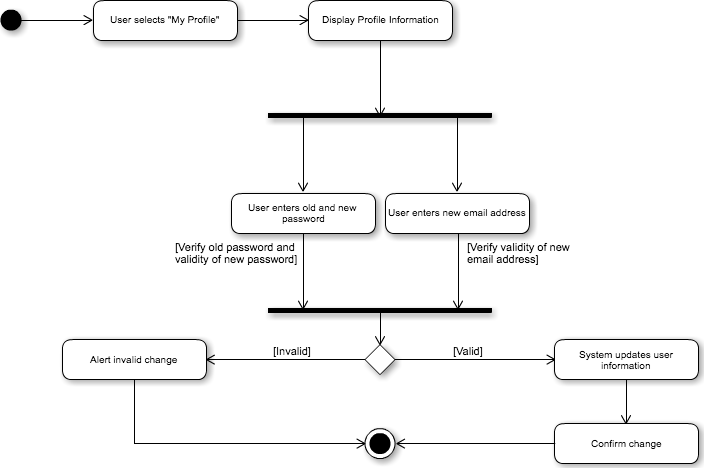


Figure 7: Change Profile Details

This activity diagram represents the path for changing the authenticated user’s profile information. The information that could be changed is limited to the email and password, since the student information is permanent in the student record. The user must first select “My Profile” from the toolbar. The user’s information is then displayed, with the email and password fields being modifiable. The user can then change either of the aforementioned, or both.

If the user wishes to change their password, then they must enter their old password and new password in those respective fields, and then press submit. The system then verifies if the inputted passwords are valid. That is, if the old password field matches the current one, and if the new password field contains a valid string.

If the user wishes to change their email, then they must simply enter their desired email address in the new email address field, and press submit. The system then verifies if the inputted email address is valid.

If all is valid, then the system updates the user information associated with the authenticated user. If the form input is invalid, then the user is alerted of their error, and no change is made in the system.

# Deployment (Physical) view

**Physical view** (also known as deployment view): Audience: Deployment managers. The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. UML Diagrams used to represent physical view include the **Deployment diagram**.

The deployment (or physical) view illustrates the physical components of the architecture, their connectors and their topology. Describe the physical network and hardware configurations on which the software will be deployed. This includes at least the various physical nodes (computers, CPUs), the interaction between (sub)systems and the connections between these nodes (bus, LAN, point-to-point, messaging, SOAP, http, http). Use a *UML deployment diagram* to capture this view.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Name of the node. | Node type. | Technical specifications. |

# Data view (optional)

**Data view** (optional): Audience: Data specialists, Database administrators. Describes the architecturally significant persistent elements in the data model . Related Artifacts: **Data model**.

An enterprise software system would additonally require a data view. The data view describes the data entities and their relationships. Deploy an *Entity-Relationship* (ER) *Model* to represent this view. Note that the ER model is not part of the UML specification. Additionally you can deploy a UML class diagram to represent the data view where classes would correspond to data entities.

# Size and performance

Describe how the architecture supports the key sizing and performance requirements, as described in the *Supplementary Specification*. For example:

Volumes:

* The server only requires a web server with PHP V7.0.9+ and MySQL version 5.6.26+ with 3 – 10 GBs depending on the size of the building and 1GB of ram
* The client side only requires one of the following web browsers: Google Chrome, Firefox, Internet Explorer, Opera, Microsoft Edge

Performance:

* Time to process a creation of reservation: less than 15 seconds required.
* Time to process a modification of a reservation: less than 15 seconds required.
* Time to process a deletion of a reservation: less than 15 seconds required.
* Time to process a view of reservation information: less than 15 seconds required.
* Time to process a view of profile information: less than 15 seconds required.
* Time to process an edit of profile information: less than 15 seconds required.

Because of the small time required to process requests, the system will be able to handle approximately 50 active users (with about 200 total accounts), as well as 50 reservations a day, with peaks in the morning.

# Quality

A description of how the software architecture contributes to the quality attributes of the system as described in the ISO-9126 (I) standard. For example: The following quality goals have been identified:

Scalability:

* Description: Increased system demands.
  + Solution: The system is given an easily extendible and flexible design which can handle the addition of more by simply adding them to the database.
* Description: Enforced fairness.
  + Solution: The system will give the user a time limit to finish submitting their reservation before another user can open a registration in the same room.

Reliability, Availability:

* Description: Mean-Time-Between-Failure
  + Solution: Mean time between failures is once a month.
* Description: Database recovery
  + Solution: The system shall provide SQL scripts and PHP files for replication to an off-site database location.

Portability:

* Description: Ability to be reused in another environment
  + Solution: The system shall provide SQL scripts and PHP files for installation on an off-site database location.

Security:

* Description: Authentication and authorization mechanisms
  + Solution: MySQL password encryption is employed.