**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 |
| November 19th, 2016 | 1.3 | Editing of Document | Stefano Pace |

Table of contents

[1. Introduction 4](#_Toc467354657)

[Purpose 4](#_Toc467354658)

[Scope 4](#_Toc467354659)

[Definitions, acronyms, and abbreviations 4](#_Toc467354660)

[References 5](#_Toc467354661)

[2. Architectural representation 6](#_Toc467354662)

[3. Architectural requirements: goals and constrains 8](#_Toc467354663)

[Functional requirements (Use case view) 8](#_Toc467354664)

[Non-functional requirements 9](#_Toc467354665)

[4. Use case view (Scenarios) 10](#_Toc467354666)

[5. Logical view 10](#_Toc467354667)

[Layers, tiers etc. 10](#_Toc467354668)

[Subsystems 11](#_Toc467354669)

[Use case realizations 12](#_Toc467354670)

[6. Size and performance 13](#_Toc467354671)

[7. Quality 14](#_Toc467354672)

**List of figures**

[Figure 1: Layered Architecture Design for the System 6](#_Toc467351740)

[Figure 2: From SAD doc 7](#_Toc467351741)

# Introduction

## Purpose

This Software Architecture Document (or SAD) delivers a well-structured description of the overall architecture for the Room Reservation System. This document contains multiple architectural views in order to illustrate the different system components in the system. This document is addressed to the stakeholders (the students and the staff), the developers and the professors of Concordia University. They are expected to use this document to fully understand the system from an architectural perspective.

## Scope

This Software Architecture document for the Room Reservation System will depict different architectural views to provide the reader with an overall architecture of the system. It will go on to show how the system will reach when users can make reservations, view reservations, edit profile information, and be placed on a waitlist if the room happens to be full at the selected time slot.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **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 |
| User Acceptance Testing Environment | UAT |
| Production Environment | PRD |

## References

Provide a list of all documents referenced in the SRS.

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

[2] C. Constantinides, "Object Oriented Design I", 2016.

[3] C. Constantinides, "Object Oriented Design II", 2016.

[4] C. Constantinides, "Object Constraint Language", 2016.

[5] C. Constantinides, "Architectural Style", 2016.

[6] C. Constantinides, "Architectural Views", 2016.

[7] C. Constantinides, "Architectural Patterns", 2016.

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

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

# Architectural representation

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.

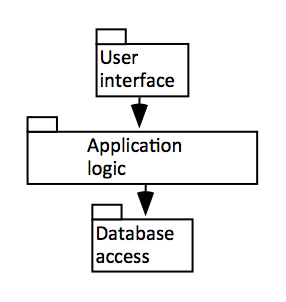


Figure 1: Layered Architecture Design for the System

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.

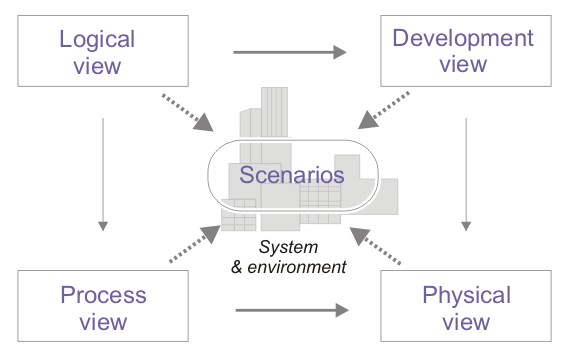


Figure 2: From SAD doc

# 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.

*Note: For all constraints related to functional and non-functional requirements, please view the “Constraints” section of the Software Requirements Specification.*

## 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.

Refer to the Functionality section of the SRS

Refer to the Analysis Model section of the SRS

# Logical view

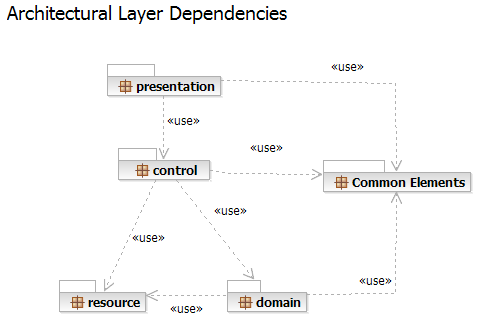
**Logical view**: Audience: Designers. The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include **Class diagram**, and **interaction diagrams** (**communication diagrams**, or **sequence diagrams**).

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.

The goal of the logical view is to present the functionality that will be available to end-users. This section will illustrate the interactions/dependencies between various system components as well as describe important logical structure of the system.

## Layers, tiers etc.

The Online Reservation System is divided into layers based on the N-Tier architecture.



The layering model of the Online Reservation System is based on a responsibility layering strategy which will associate each layer with a particular responsibility.

## Subsystems

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

* The **presentation layer** deals with the presentation logic and the pages rendering
* The **control layer** manages the access to the data layer
* The **resource layer** (integration layer) is responsible for the access to the enterprise information system (databases or other sources of information)
* The **domain layer** is related to the business logic and manages the accesses to the resource layer.
* The **Common Elements** **layer** gathers the common objects reused through all the layers
  + Unit of work will be included here

**Architecturally significant design packages**

Describe packages of individual subsystems that are architecturally significant. For each package include 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.

# Size and performance

Volumes:

* Estimated reservations per Week (Maximum): 250
  + 200 total accounts with 50 daily active users will produce roughly 250 reservations in a week.
* 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 1 minute required (Maximum time).
* Time to process a modification of a reservation: less than 1 minute required (Maximum time).
* Time to process a deletion of a reservation: less than 1 minute required (Maximum time).
* Time to process a view of reservation information: less than 1 minute required (Maximum time).
* Time to process a view of profile information: less than 1 minute required (Maximum time).
* Time to process an edit of profile information: less than 1 minute required (Maximum time).

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

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.

*Note: Please also refer to the respective sections in the SRS for Reliability, Portability, Scalability*