Software Architecture Document

Version 1.6

for

Room Booking System

Prepared by

Robert McAuley	27195788	mcauley.robert@live.com
David Dubé	27525036	dav_dube@encs.concordia.ca
Harrison Andriamanantena	27435282	Harrison.a101@hotmail.com
Joshua Chan-Fee	26301959	joshuachanfee@gmail.com
Ioan Cioca	27526601	ioancioca@yahoo.ca
Ali Douch	27578253	alidouch@hotmail.com
George Kolampas	27192233	gkolampas@gmail.com
Darrel-Day Guerrero	27352409	ddayy.guerrero@hotmail.com

Instructor: Dr. Constantinides

Course: SOEN 343

Date: November 12th 2016

Document history

Date	Version	Description	Author
October 25th 2016	1.1	Introduction and architectural Representation	George Kolampas
October 29th 2016	1.2	Size and Performance, and Quality	Ioan Cioca
October 29th 2016	1.3	Logical View	Ali Douch / Robert McAuley
November 5th 2016	1.4	Use Case View	Joshua Chan-Fee
November 6th 2016	1.5	Minor corrections to ensure consistency	David Dubé
November 12th, 2016	1.6	Updated class diagram	Ali Douch

Table of contents

List of figures

1. Introduction

The introduction of the Software Architecture Document provides an overview of the entire document.

The Software Architecture Document introduction describes a summary of the document in its entirety.

Purpose

This document provides a comprehensive architectural overview of the system by use of different views. It aims to describe all architectural decisions made on the system. This document is intended for developers to create the system and for other stakeholders to understand the design decisions of the system.

Scope

This document applies to the room booking system created by the entire team. The coding was done by following the guidelines shown in this document.

2. Architectural representation

This document presents the architectural style as two views: the logical view and use case view, also known as scenarios. These two views are taken from the 4+1 view. It can therefore be said that this is an incomplete 4+1 view that many businesses use.

The two views are described below:

1. **Logical view**: Audience: Designers. The logical view is concerned with the functionality that the system provides to end-users. A class diagram and sequence diagrams for the critical use cases are used. Related Artifacts: Class Diagram and Sequence Diagram.

2. **Use case view**: 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. 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 Artifacts: Use Case Diagram.

3. Architectural requirements: goals and constraints

Functional requirements (Use case view)

The architecturally significant use cases for this system are login and make reservation. Login is vital as only logged in users can use the system. Make reservation, seeing as it's the main function of the system and is what many other use cases are based on, is a significant use case.

The overview below refers to architecturally relevant Use Cases from the Use Case Model.

Source	Name	Architectural relevance	Addressed in:
Login	Login	This use case is relevant to the architecture as it is crucial to the functioning of the system. No further functional requirements can be met without the user logging in.	Section 4.1
Create Reservation	Create Reservation	This use case is relevant to the architecture as it is the main function of the system. Many other use cases are based off of it.	Section 4.2

4. Use case view (Scenarios)

The use case view describes the set of architecturally significant use cases. These represent critical system behaviours as seen by its actors. Use case scenarios describe interactions between actors and the system viewed as a black box. We use the UML use case diagram to represent this.

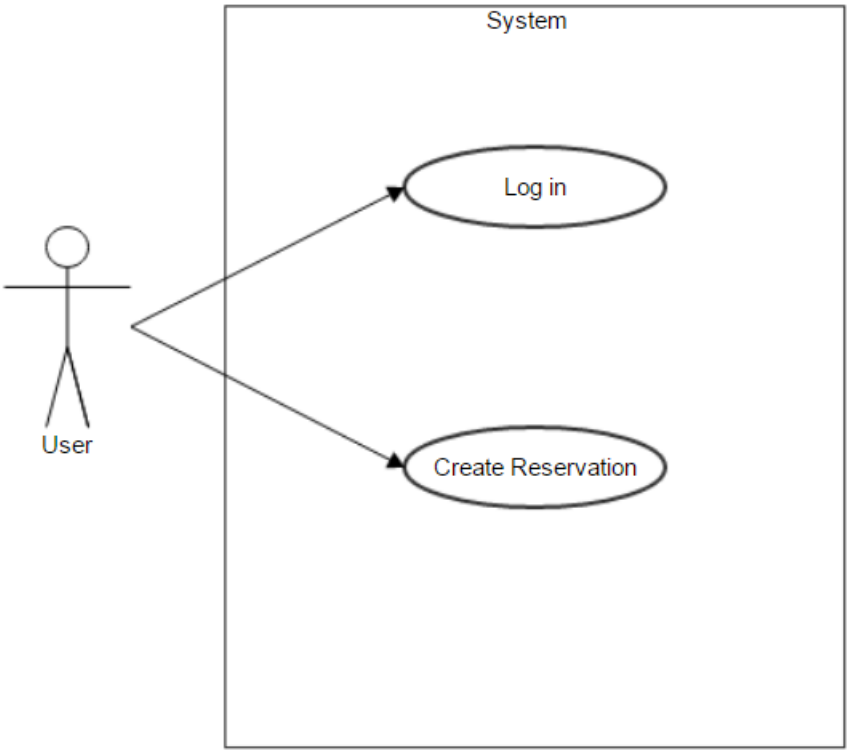

Figure 1: Architecturally Relevant Use Cases

4.1 Login

Brief Description: This use case describes how a user logs into the system.

4.2 Create reservation

Brief Description: This use case describes how a user creates a reservation for a specific room.

5. Logical view

The logical view captures the functionality provided by the system; it illustrates the collaborations between system components to realize the system's use cases.
We use a UML class diagram and UML sequence diagrams to illustrate these logical views.

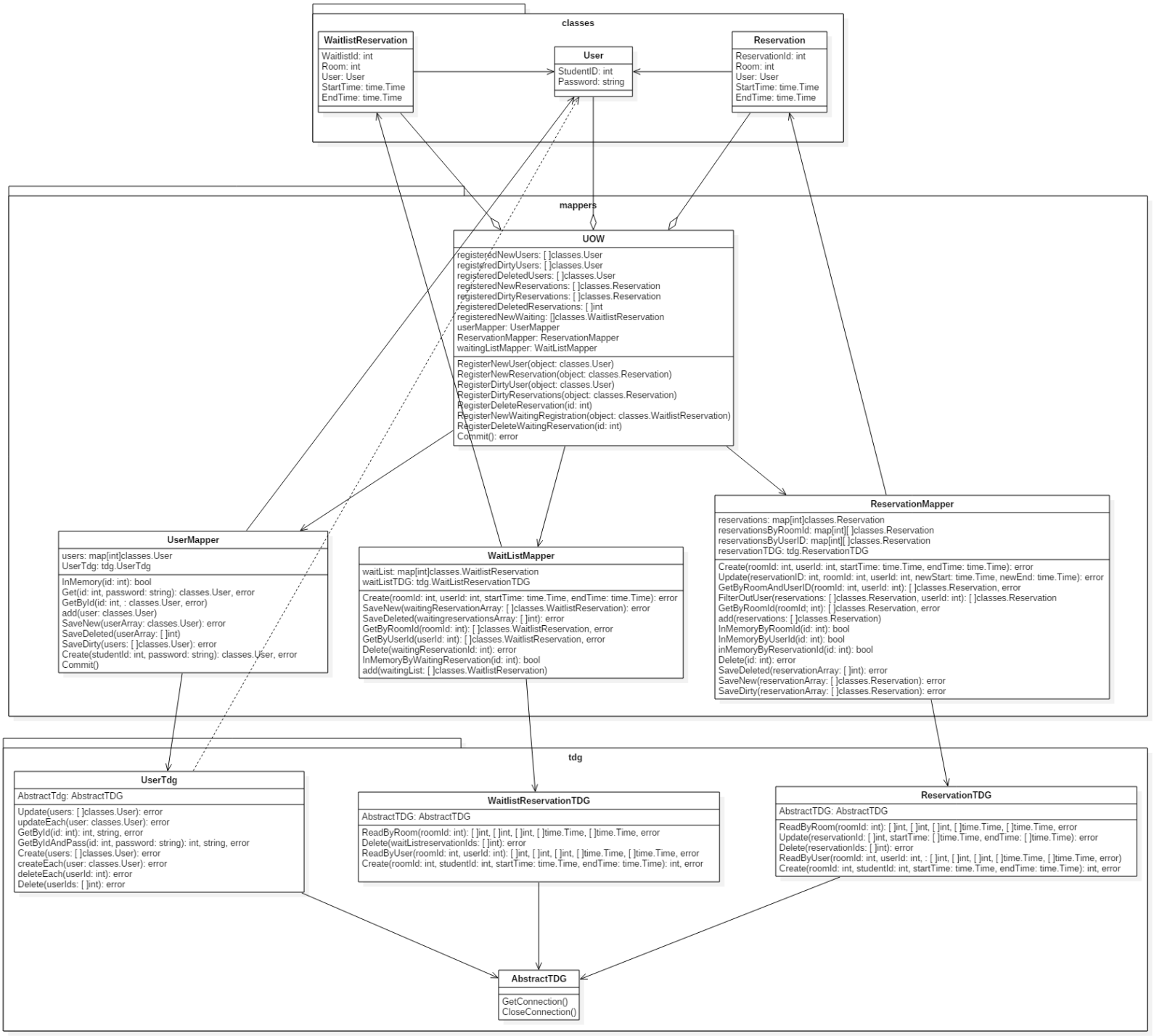

Figure 2: Class Diagram

5.1 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.

Class package: Represents the different domain model entities. For diagram see above.
Mapper package: Design patterns used to communicate between domain model entities and the data layer. For diagram see above.
Table Data Gateway: Design pattern to abstract interaction with the database regardless of database implementation. For diagram see above.

5.2 Use case realizations

This section describes, using sequence diagrams, how the use case scenarios are executed.

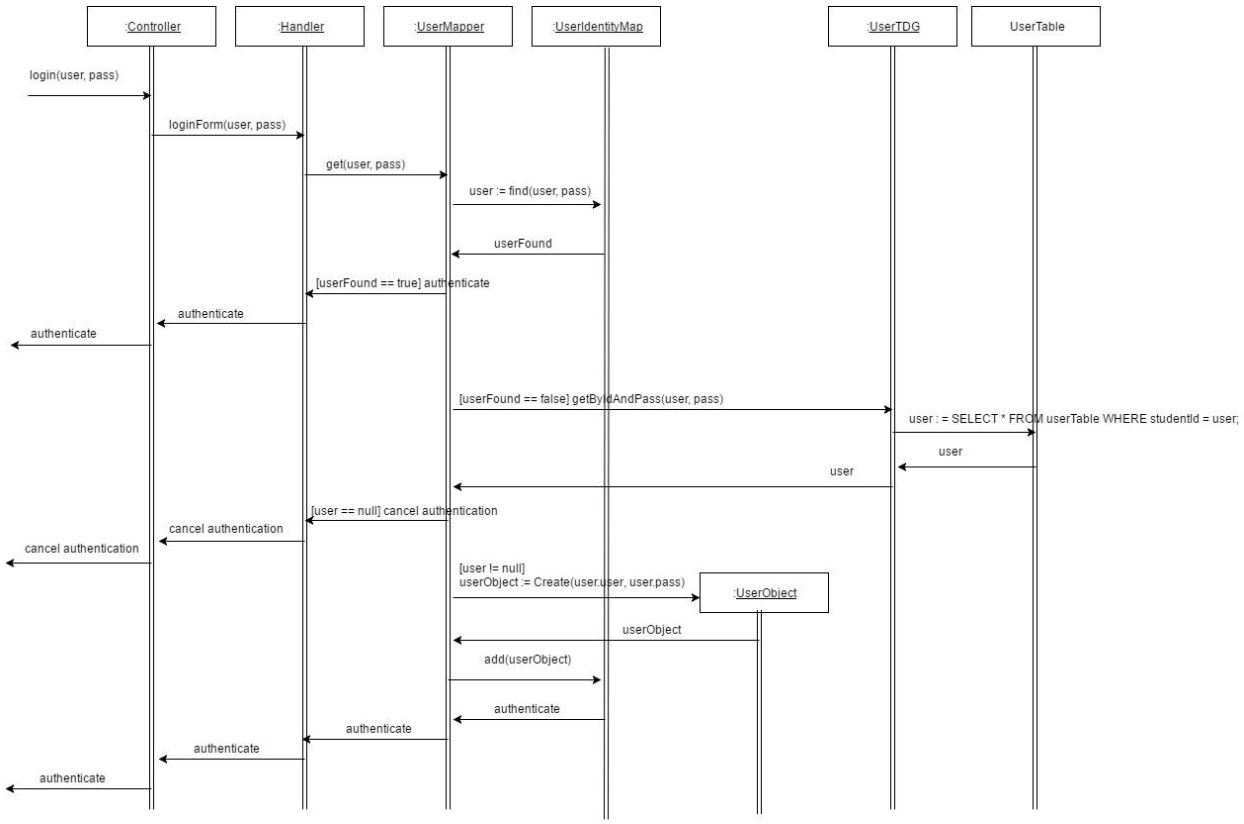

Figure 3: Login Sequence Diagram

The diagram on the previous page depicts the login method. The message is directed from the controller to the handler, user mapper, user identity map, User Table Data Gateway, and User Table. Upon successful login, a user object is created.

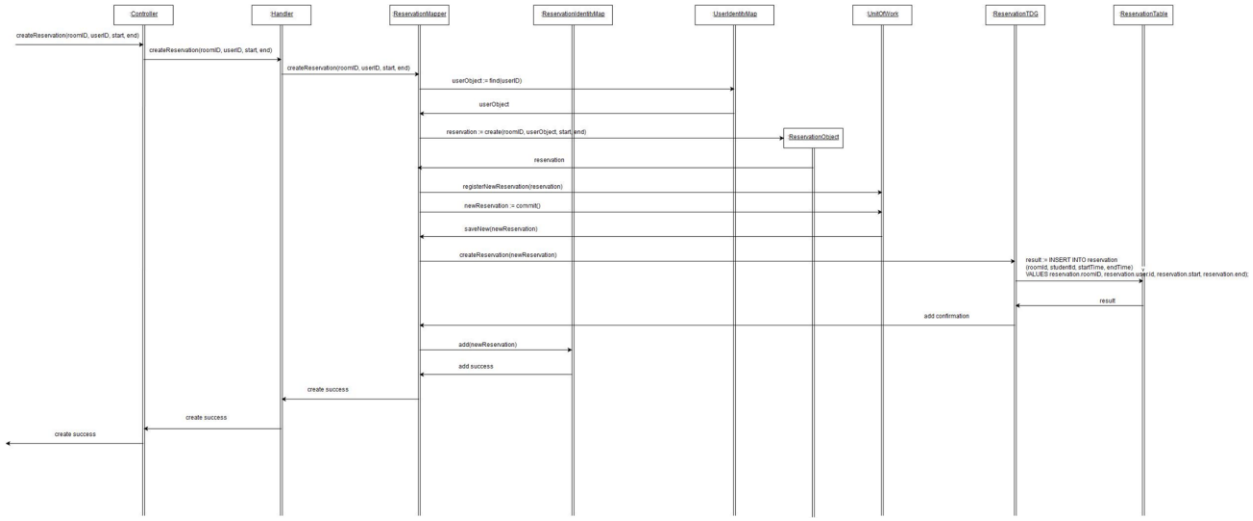

Figure 4: Create Reservation Sequence Diagram

This diagram describes the steps needed to create a reservation. The user sends the message to the controller, which sends the message to the handler who then sends it to the mapper. The mapper retrieves the user object from the user identity map by passing in the student ID. The mapper proceeds to create a reservation object and then passes it on to the unit of work. The reservation mapper sends a commit message which instructs the unit of work to save the reservation. The mapper calls the table data gateway to insert the new reservation into the reservation table. The reservation mapper then saves the created reservation into the reservation identity map and sends a success message to the handler which then sends it to the controller which displays it to the user.

6. Size and performance

This system is designed to handle users through one server. It is designed to run both the database and application on one server.

Volumes:

- Estimated online request: 10 per day.
- Registered individual customers: 8.

Performance:

- Time to log in: less than 5 seconds.
- Time for reservation creation, modification, and deletion: less than 5 seconds required. Benchmarks were created to measure and validate performance within the specification.

Unit Test Results:

```
$ go test -v
=== RUN   TestLoginForm
Found studentId:1111111
Found password:1111111
Found User in db
Found studentId:2222222
Found password:2222222
Found User in db
Found studentId:3333333
Found password:3333333
Found User in db
Found studentId:4444444
Found password:4444444
Found User in db
Found studentId:5555555
Found password:5555555
Found User in db
Found studentId:6666666
Found password:6666666
Found User in db
Found studentId:7777777
Found password:7777777
Found User in db
--- PASS: TestLoginForm (0.30s)
        loginForm_test.go:41: case:1111111
```

```
    loginForm_test.go:41: case:2222222
    loginForm_test.go:41: case:3333333
    loginForm_test.go:41: case:4444444
    loginForm_test.go:41: case:5555555
    loginForm_test.go:41: case:6666666
    loginForm_test.go:41: case:7777777
    loginForm_test.go:41: case:kjdnfgoi
    loginForm_test.go:41: case:
    loginForm_test.go:41: case:124512
=== RUN   TestLoginGet
--- PASS: TestLoginGet (0.00s)
=== RUN   TestReservation
--- PASS: TestReservation (0.10s)
PASS
```

This log shows a successful login and a successful reservation creation.

Benchmark Results:

```
$ go test -bench=.
Found studentId:1111111
Found password:1111111
Found User in db
Found studentId:2222222
Found password:2222222
Found User in db
Found studentId:3333333
Found password:3333333
Found User in db
Found studentId:4444444
Found password:4444444
Found User in db
Found studentId:5555555
Found password:5555555
Found User in db
Found studentId:6666666
Found password:6666666
Found User in db
Found studentId:7777777
Found password:7777777
Found User in db
BenchmarkReservation-4          50          35221086 ns/op
PASS
```

This log means that one operation, on average, (one read and one create) takes 0.03 seconds.

7. Quality

A description of how the software architecture contributes to the quality attributes of the system as described in the ISO-9126 (I) standard.

Scalability:

- Description: System's reaction when user demands increase
- Solution: This system is designed to handle 8 users initially but subsequent registered users can be added up to a few hundred at once on basic hardware.

Reliability, Availability:

- Description: Transparent failover mechanism, mean-time-between-failure
- Solution: A crashed server can be quickly rebooted. Interruption of service will still occur during server reboot and session information will be lost.

Portability:

- Description: Ability to be reused in another environment
- Solution: The system can be run on any of the major operating systems if the database and the go programming language have been installed.

Security:

- Description: Authentication and authorization mechanisms
- Solution : The system only allows authenticated users to use it. This is done by redirecting anyone who attempts to use the system to the login page.