**Software Architecture**

**Document**

Version 1.3

for

The Force Awakens

Prepared by

|  |  |  |
| --- | --- | --- |
| Nicholas Burdet | 29613773 | nicholas\_burdet@hotmail.com |
| Georges Mathieu | 26863477 | gmathieu6@hotmail.com |
| Olivier Cameron-Chevrier | 27228805 | olivier.cc@hotmail.com |
| Stefano Pace | 27454716 | Stefano.pace12@gmail.com |
| Julian Ippolito | 27419112 | julian.ippolito@hotmail.com |
| Adam Trudeau-Arcaro | 27459157 | adamarcaro@gmail.com |
| Joey Tedeschi | 27513062 | joeyg.tedeschi@gmail.com |

|  |  |
| --- | --- |
| 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 |
| November 20th, 2016 | Final | Final Editing of Documentation | Stefano Pace |

\*\*\*\*\*\*\*Please Note:

Identity map was planned to be integrated into the system in order to ease the amount of loading from the database by creating a cache for lazy loading (loading information only when it was needed). The problem encountered was with the system's design at the time identity maps were decided to be used. The class structure to be used that was negotiated at the beginning of the project conflicted with the core ideas of identity mapping. Since the Room Reservation System makes various different calls to the database to obtain reservations by different groupings (by week/month/per student/waitlist/etc), identifying future queries into the identity map would be difficult to know whether another type of grouping needed by the program was complete or missing entries from the database. For that reason, one of the only determined fixes was to load the entirety of the database into the cache upon first user's loading. However, this plan contradicts the basis of the lazy loading carried out by the identity maps. For this reason, on top of the amount of refactoring required to implement this feature into the system, Identity mapping was scoped out of the final product.

Table of contents

[1. Introduction 4](#_Toc467416135)

[Purpose 4](#_Toc467416136)

[Scope 4](#_Toc467416137)

[Definitions, acronyms, and abbreviations 4](#_Toc467416138)

[References 5](#_Toc467416139)

[2. Architectural representation 6](#_Toc467416140)

[3. Architectural requirements: goals and constrains 8](#_Toc467416141)

[Functional requirements (Use case view) 9](#_Toc467416142)

[Non-functional requirements 10](#_Toc467416143)

[4. Use case view (Scenarios) 11](#_Toc467416144)

[5. Logical view 11](#_Toc467416145)

[Layers, tiers etc. 11](#_Toc467416146)

[Subsystems 12](#_Toc467416147)

[Use case realizations 16](#_Toc467416148)

[6. Size and performance 19](#_Toc467416149)

[7. Quality 20](#_Toc467416150)

**List of figures**

[Figure 1: Layered Architecture Design for the System 6](#_Toc467416151)

[Figure 2: Architectural View Breakdown 7](#_Toc467416152)

[Figure 3: Class Diagram 12](#_Toc467416153)

[Figure 4: Significant Design Architecture 14](#_Toc467416154)

[Figure 5: Application Logic 15](#_Toc467416155)

[Figure 6: Database Access 15](#_Toc467416156)

[Figure 7: Adding a Reservation Sequence Diagram 16](#_Toc467416157)

[Figure 8: Modifying a Reservation 16](#_Toc467416158)

[Figure 9: Deleting a Reservation 17](#_Toc467416159)

[Figure 10: Editing Profile Information 18](#_Toc467416160)

# 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 react when users make reservations, view reservations, edit profile information, as well as once they are placed on a waitlist (should the room happen 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 |
| Table Date Gateway | TDG |

## 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.

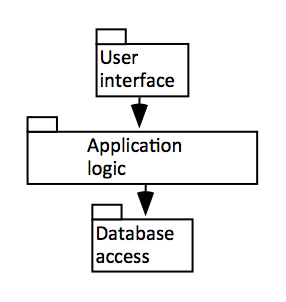


Figure 1: Layered Architecture Design for the System

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 incorporated: Logical, Development, Process, Physical, and Use Case. Many of these views are outside the scope of this course, however two views will be covered: Logical view, and Use Case view. Logical view is concerned with the functionality that the system provides to end-users, and its main audience is the designers of the software. Use Case view is presented to all stakeholders of the systems (even end-users), and is a small set of scenarios/use cases describing a series of interactions between objects and processes.

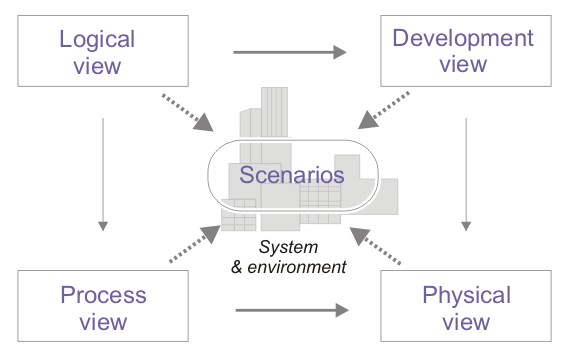


Figure 2: Architectural View Breakdown

# Architectural requirements: goals and constrains

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. Incorporate a waiting queue, for people who want to make reservations, which does not prioritize any user. This queue implies that, even if they don’t get the room they want, their action was acknowledged and an action was taken as a result.
* Liveness: Liveness is incorporated through the use of timers when making or modifying a reservation. Each user is given a time limit within the room, ensuring that, eventually, users waiting for a room will be able to do what they want to do (once they are given a chance).

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)

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| SRS Use Case UC1 | Check Username and Password | Must implement the Unit of Work | SAD Unit of Work Importance. |
| SRS Use Case UC2 | Add Reservation | Must implement the Unit of Work | SAD Unit of Work Importance. |
| SRS Use Case UC3 | Update Password | Must implement the Unit of Work | SAD Unit of Work Importance. |
| SRS Use Case UC4 | Update Email Address | Must implement the Unit of Work | SAD Unit of Work Importance. |
| SRS Use Case UC5 | Modify Reservation | Must implement the Unit of Work | SAD Unit of Work Importance. |
| SRS Use Case UC6 | Delete Reservation | Must implement the Unit of Work | SAD Unit of Work Importance. |

Importance of Unit of Work

When pulling data in and out of a database, it is important to keep track of what you have changed to maintain the integrity of the database. If a new object must be inserted, then that object must be inserted and if an object me be removed then you delete that object. To avoid having multiple database calls when manipulating multiple object, you can use the Unit of work to implement all updates at the end of a sequence.

A unit of work will keep all information in one place, therefore keeping track of everything you do during a transaction that can affect the database. When it is time to commit all create, changes or deletes in a database the Unit of work the unit of work compiles everything and writes the changes to the database in one connection.

In the case of the Room Reservation System, it is important to include this design pattern for all of our functionality, as all functionality in the system relies on the integrity of the data stored in the database. For example, If data was not created/deleted/modified at appropriate times this would cause our system to accept overlapping reservations and allow users to go over their maximum numbers of reservations for a week. To meet all constraints, as well as functional requirements, we must include a Unit of Work.

## Non-functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Name** | **Architectural relevance** | **Addressed in:** |
| SRS Document | Fairness | This is to prevent users from constantly attempting to make reservations and locking Rooms for an indefinite amount of time. | SRS Accessibility |
| SRS Document | Efficiency | This sets a Limit on the time it will take for a functionality to be performed for a user | SRS Efficiency |
| Vision | Password Protection | Passwords for all users will be hashed when stored in the database to prevent malicious attempts at retrieving them. | MySQL provides default hashing functions which will be integrated with all database calls. |

# Use case view (Scenarios)

Refer to the SRS Analysis Models Section

# Logical view

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 selected “Layered Architecture” as its top-level architecture style. In this style, each layer provides its services to the layer above it, such that the lowest layer would imply services to be used by the system as a whole (e.g.: Database services). Refer to section 2, Architectural Representations, for a more robust explanation of the architectural style as a whole.

With regards to the Room Reservation System as a whole, the layered architecture style can be presented in the form of a UML class diagram, as seen below (Note: Not all functions are included in each class, as it would be redundant and fails to demonstrate the top-level architecture style). The breakdown of each subsystem will be seen in the following section.

It is important to add that, as the Room Reservation System in the case of this group is a Web-based application, there is no software class associated with the user interface. Thus, the class diagram below will highlight the application logic and database accesses done by the system. All this information is then taken and presented to the user visually, but this is not a concern for the class diagram. Additionally, mutator and accessor methods are not shown for private variables (in the interest of space and redundancy), however they are implied.

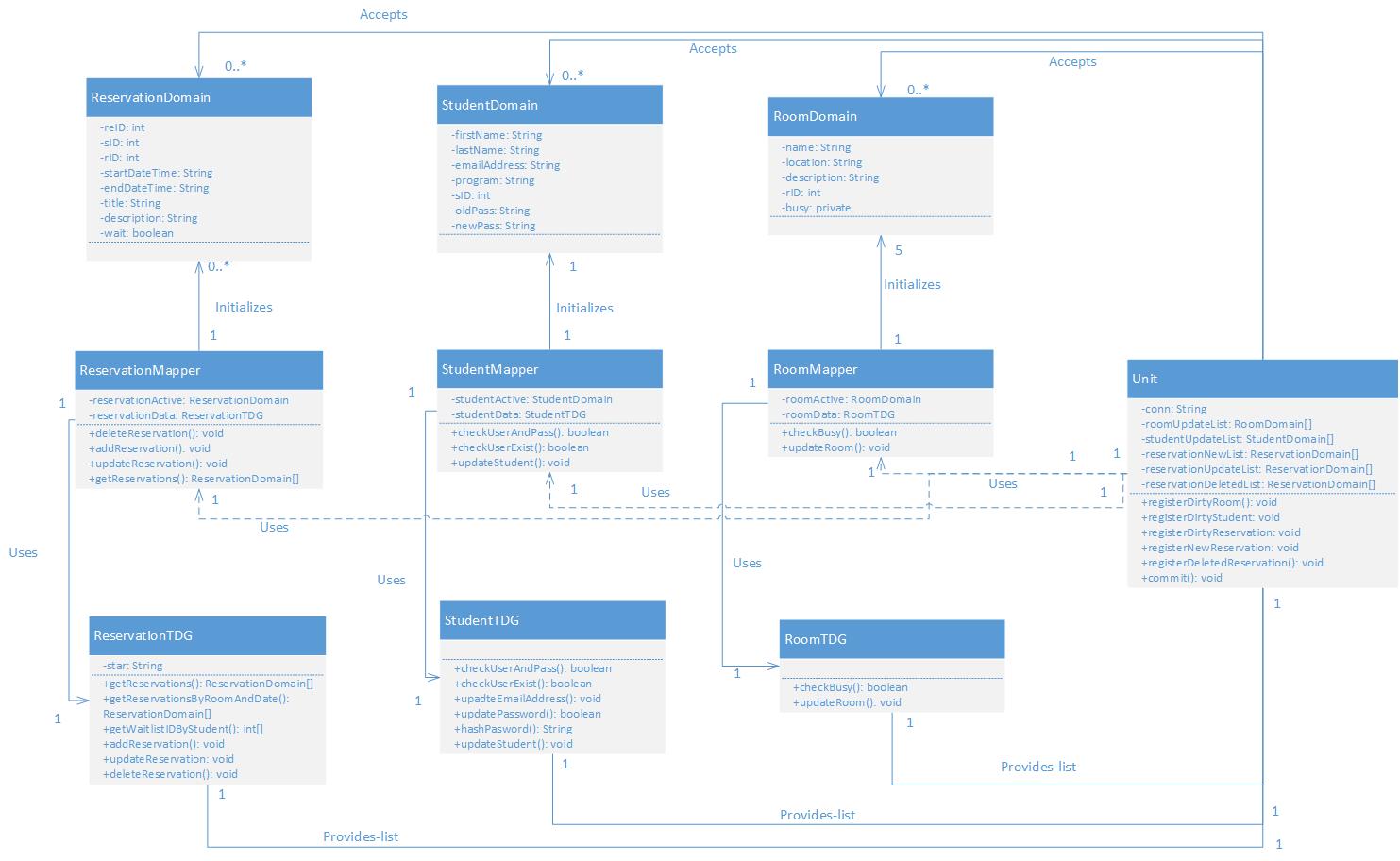


Figure 3: Class Diagram

## Subsystems

The Room Reservation System can be broken down into 3 major subsystems: User Interface, Application Logic, and Database Access.

User Interface is a subsystem considered with displaying data and information to the users of the system. It is the topmost layer, and thus no logic is found at this level. Instead, users are able to perform actions and generate requests which the system will then attempt to handle by moving to the second layer: The Application Logic layer.

Application Logic is the second tier of the system, and is a subsystem represented by the Mapper and Domain classes of the system (refer to class diagram above). Based on user request, objects defined by the Domain classes will become “dirty” (i.e.: Edited/changed), and upon compiling all user requests, these dirty objects will be passed via the Mappers to the lowest layer of the system. As such, the application logic layer serves as a “middle ground” of sorts, receiving user requests (and processing them then acting accordingly), but also receiving information from the database in order to present any changes the user might desire.

The final, and lowest layer, the Database Access layer, is represented in the class diagram above by the TDG classes. Once a user request has been processed, their request is passed to the TDG, which then allows access to the external database, where the appropriate information can be accessed or mutated. By definition, the TDG serves as a gateway between domain objects and the database. The TDGs receive messages, which then define the type of interaction they will have with the database.

Referring to Figure 1, one can see that the relation between these subsystems is a quite linear one: They all interact vertically with one another (up and down), and without a single layer, the entire system would fall apart. By separating all responsibilities into layers, a clear distinction in services can be seen, allowing each individual level to provide services to the level above it, and act as a client (receiver of services) to the level below it. User Interface sends HTTP requests to the Application logic layer, where the logic is applied to the request and then sends it to the Database Access level. From here, the database server executes the request (using MySQL), and sends the result back to the Application logic level. From here, the response is sent back to the User Interface (and thus the web server), and displays the information for the user.

**Architecturally significant design packages**

Overall Architecture Package

The architecture of our system was split into 2 main packages. They are the application Logic and Database Access (See Subsystem section for details)

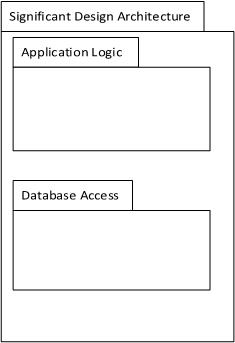


Figure 4: Significant Design Architecture

Application Logic Package

Within the application logic package, there are 2 sub packages. They are the Domain package and the Mapper package. The domain package contains all the Domain objects necessary for system functionality (as previously described in the subsystems section). The Mapper package contains all the Mapper objects necessary for system functionality (as previously described in the subsystems section).

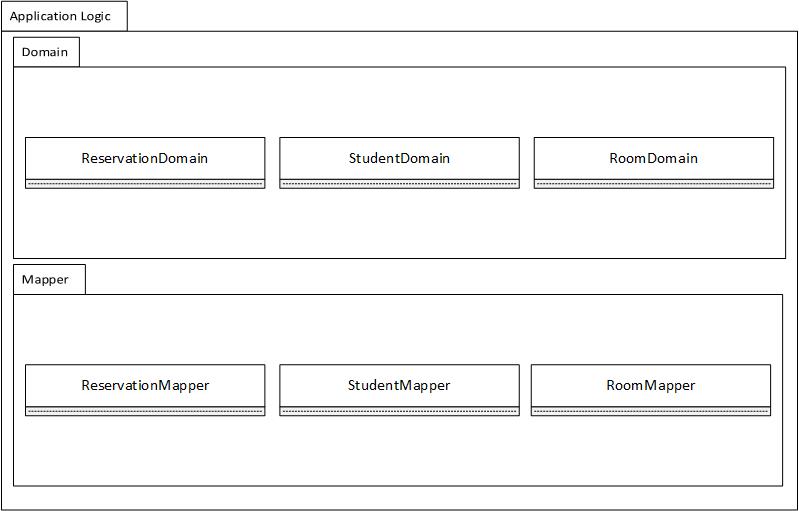


Figure 5: Application Logic

Database Access

The Database Access package contains only 1 sub package; Table Data Gateway (TDG). Within the TDG package, all the TDG objects necessary for manipulating the central system database can be found.

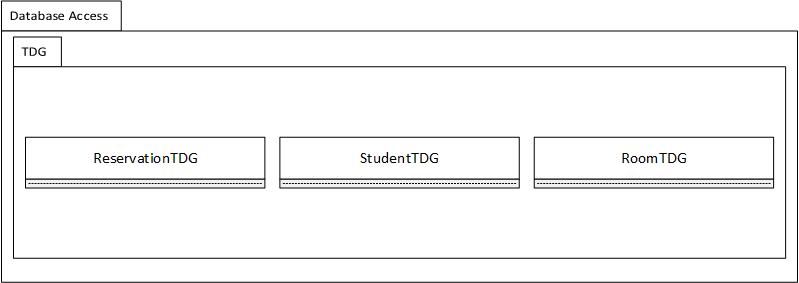


Figure 6: Database Access

## Use case realizations

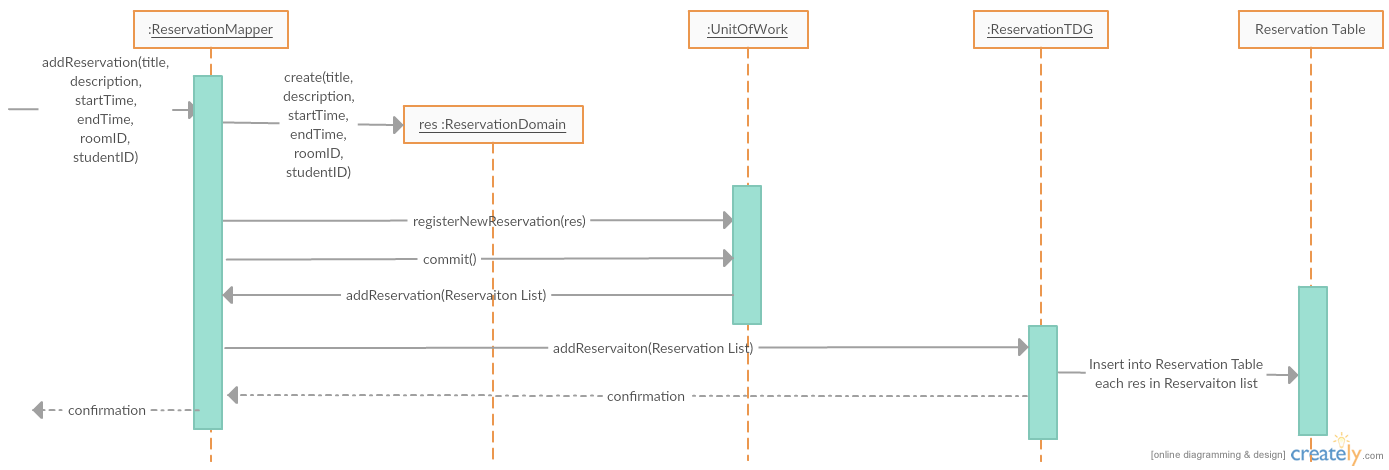


Figure 7: Adding a Reservation Sequence Diagram

Once the user initiates the creation of a new Reservation, the system initiates a new instance of type Reservation Domain. This newly created instance is then passed into the unit of work and registered as a New Reservation to be added into the database at the time of commit.

When the unit of work initiates a commit, the list of all newly added reservations is passed through the Reservation Mapper class to the Reservation TDG. Once in the TDG, the INSERT statements occur in the database. Upon successful completion the user will receive a confirmation message.

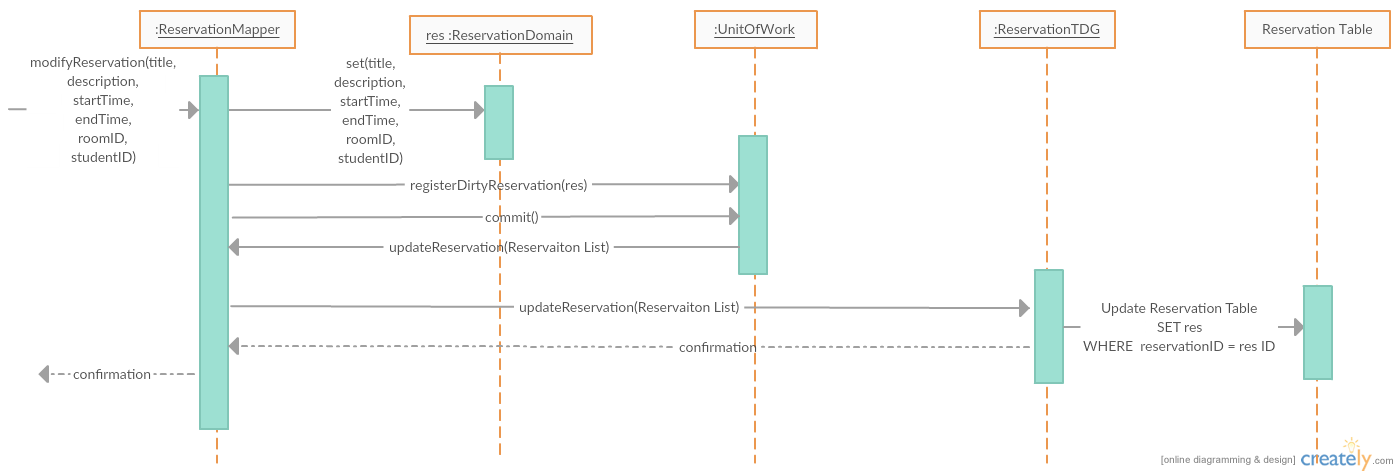


Figure 8: Modifying a Reservation

Once the user initiates the modification of a current Reservation, the system initiates a new instance of type Reservation Domain with the same Reservation ID as the previous version of the object. This modified instance is then passed into the unit of work and registered as a dirty Reservation to be modified in the database at the time of commit.

When the unit of work initiates a commit, the list of all dirty reservation objects is passed through the Reservation Mapper class to the Reservation TDG. Once in the TDG, the UPDATE statements occur in the database. Upon successful completion the user will receive a confirmation message.

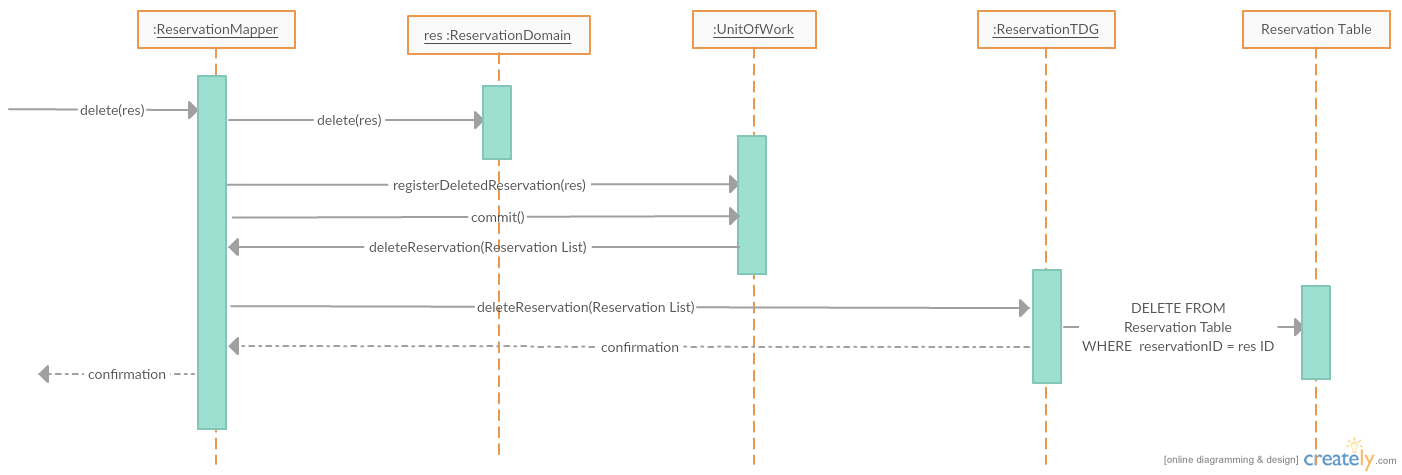


Figure 9: Deleting a Reservation

Once the user initiates the delete of a current Reservation, the system passes the reservation object to be deleted into the unit of work and registers it as a deleted Reservation to be removed from the database at the time of commit. The Domain Reservation for that instance is then also deleted.

When the unit of work initiates a commit, the list of all deleted reservation objects is passed through the Reservation Mapper class to the Reservation TDG. Once in the TDG, the DELETE statements occur in the database. Upon successful completion the user will receive a confirmation message.

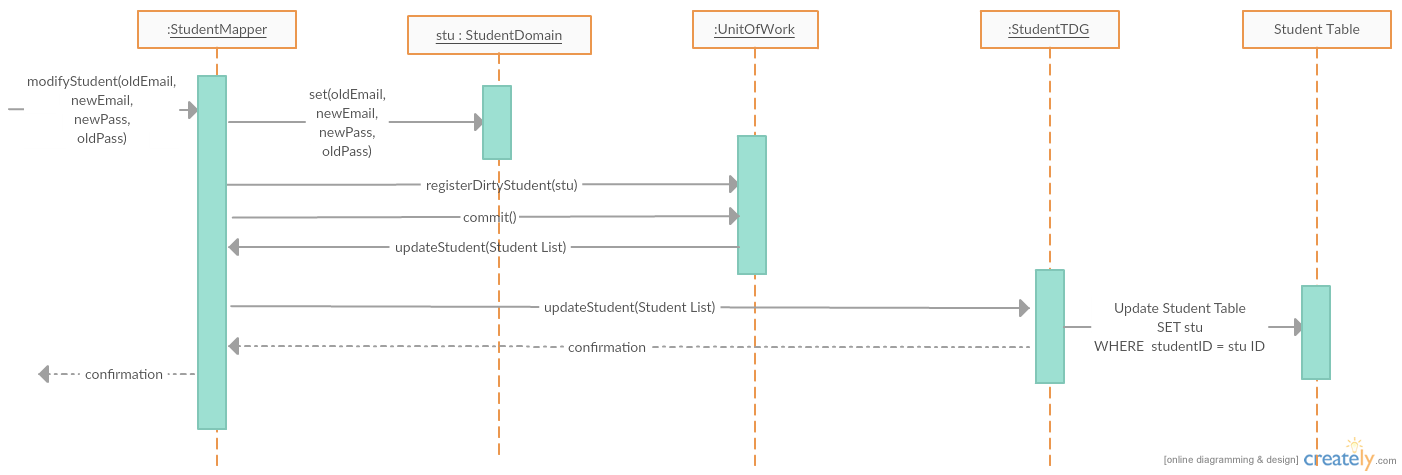


Figure 10: Editing Profile Information

Once the user initiates the modification of a current Student, the system modifies the current instance of Student Domain with the same student ID. This modified instance is then passed into the unit of work and registered as a dirty Student to be modified in the database at the time of commit.

When the unit of work initiates a commit, the list of all dirty Student objects is passed through the Student Mapper class to the Student TDG. Once in the TDG, the UPDATE statements occur in the database. Upon successful completion the user will receive a confirmation message.

# 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 more information on Reliability, Portability, Scalability*