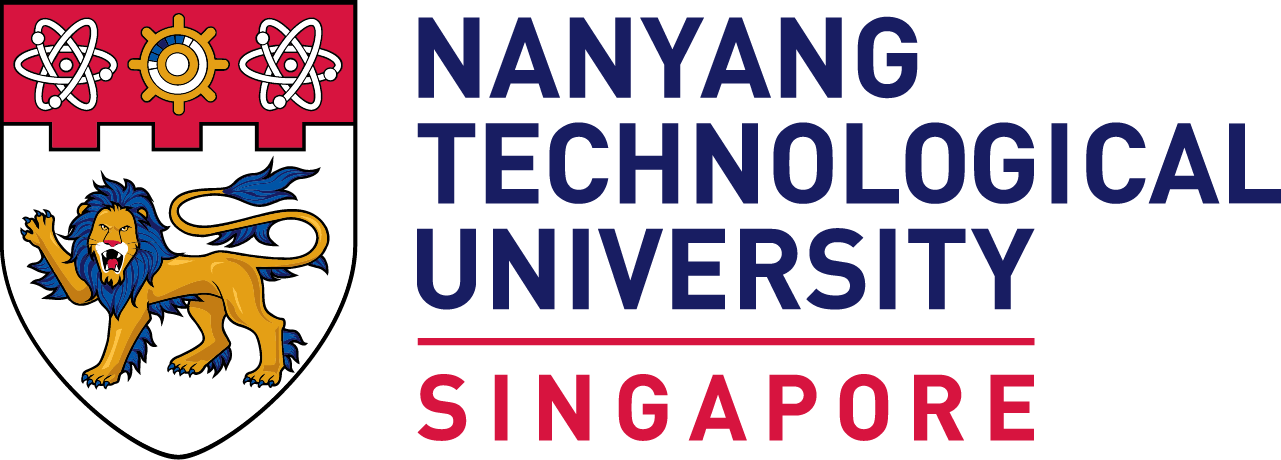
# 



**CZ2002 Object - Oriented Design & Programming**

**Group Assignment Report**

**Lab group SSP1**

**Group 5**

| Group Members | |
| --- | --- |
| Name | Matriculation No. |
| Teo Jun Yi, Austin | U2022547L |
| Evan Sean Sainani | U2022976B |
| Liew Yew Loong, Jefferson | U2021690C |
| Ryan Wong Jing Heng | U2022472J |
| Sebastian Cheong Yew Ho | U2023660D |

# **Table of Contents**

[**Table of Contents**](#_o0yhmqxuniqj) **2**

[**Declaration of Original Work for CE/CZ2002 Assignment**](#_2yuoyd5u381c) **3**

[**Introduction**](#_lijzc65r6w76) **4**

[**Design Considerations and OO Concept**](#_me5vq7kk0urt) **4**

[Apache Maven Java Project](#_2763p3uhvmyh) 4

[Minimal Configuration Database / File Storage System](#_424d36okqhfz) 4

[Clearly Defined Codebase Structure](#_yzzu9ygct4z) 5

[Modular Code Design (Separation of concern)](#_xvv7c2jdmieg) 5

[Single Responsibility Principle (SRP)](#_lwcqms59dp1q) 6

[**Class Diagram**](#_cwa8s2k9b968) **7**

[**Sequence Diagram for Checking/Removing Reservation**](#_p9uazq5uo6bz) **8**

[Important/notable parts](#_31lxygwwhywp) 8

[**Screen Captures**](#_coyhs427reee) **10**

[Test Case 1](#_809ff6jiuz1s) 10

[Result](#_bl4xq2esc2e0) 10

[Test Case 2](#_1p61qu44cdii) **10**

[Result](#_8iskbo6ch3rx) 10

[Test Case 3](#_x0thrvmeqjv6) 11

[Result](#_220c00wmb21v) 11

[Test Case 4](#_454tuxmz6dlw) 11

[**Video Demonstration Link**](#_rtfum1ow1vgn) **11**

[**Javadoc**](#_98meoa794lh7) **11**

# Declaration of Original Work for CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

| Name | Course  (CE2002 or CZ2002) | Lab  Group | Signature /Date |
| --- | --- | --- | --- |
| Teo Jun Yi, Austin | CZ2002 | SSP1 | Austin 14/11/21 |
| Evan Sean Sainani | CZ2002 | SSP1 | Evan 14/11/21 |
| Jefferson Liew Yew Loong | CZ2002 | SSP1 | 14/11/21 |
| Ryan Wong Jing Heng | CZ2002 | SSP1 | 14/11/21 |
| Sebastian Cheong Yew Ho | CZ2002 | SSP1 | Sebastian 14/11/21 |

# 

# 

# Introduction

We are tasked to design and develop a Restaurant Reservation and Point of Sale System (RRPSS), an application that will be used by restaurant staff. The RRPSS aims to be an application that improves the workflow and time efficiency of the restaurants’ day-to-day operations. It does so by computerizing the processes of managing reservations, orders and the generation of sales reports. It will also have the capability of editing the items on the menu, creating and editing promotional packages, all with ease. With this implementation, the restaurant staff will be able to produce their best performance in providing quality table service without the stress of performing laborious tasks.

# Design Considerations and OO Concept

## Apache Maven Java Project

We have opted to use an Apache Maven project as Maven allows for external dependencies or external packages to be added or removed in a way that is least intrusive for all project members.

Some external dependencies we are utilizing are Simple Logging Facade for Java (SLF4J) and Google GSON. These external dependencies reduces redundancy by ensuring that our development process can be focused on implementing our business logic and progressing quickly in the development lifecycle instead of constantly “reinventing the wheel”

## Minimal Configuration Database / File Storage System

As we are restricted from using SQL based database solutions, we are assuming NoSQL based database solutions such as MongoDB are not allowed too and we are required to come up with our own solution for data storage.

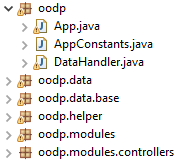
It is imperative to have a robust data access and data storage system that requires minimal interaction while providing maximum control, hence we have developed a data storage solution that requires zero effort from the developers other than inheriting from a ‘BaseData’ class

This solution relies on a DataHandler class which invokes a series of methods that acts on the classes that are inheriting ‘BaseData’, with the intention of serializing the superclasses into flat JSON strings through inheritance.

Having a simple and robust system would mean that there is little to no learning curve for developers to utilize the system.

Data access and storage is as simple as writing 1 line to create/retrieve/update/delete data.

## Clearly Defined Codebase Structure



The code structure is clearly defined with boundaries, controllers and entities clearly separated.

This allows for an “easy to learn” codebase implementation as the functionality of each individual class can be inferred from its package path and class name.

Helper classes are introduced to reduce code redundancy and promote clean code alongside error-free code as exceptions are handled by the same method calls.

The helper classes (IOHelper, DataHandler, SeedHelper) are accessible by all modules and unique to none, this allows for maximum code compatibility and reusability. These helper classes contain many instances of overloading to also help maximize their flexibility and readability leading to seamless integration with the rest of the programme.

## Modular Code Design (Separation of concern)

The codebase is developed with a structure that minimizes overlaps of code intention. A modular system is used; where each individual work area is separated into their respective module classes.

Each module is clearly defined to have implementations for a specific work area that can be inferred by their class name in a sensible manner.

Having a modular code design ensures a clearly defined separation of concern pattern and minimizes code conflicts to improve work efficiency, code cleanliness and to reduce development frustration.

## Single Responsibility Principle (SRP)

Each class was created with specific responsibilities in mind so that there is never more than one reason for it to change. By following this principle, our classes will adhere to one functionality, having their data and methods to serve their sole purpose. This results in high cohesion and reliability, which ultimately reduces error.

# **Class Diagram**

# 

# 

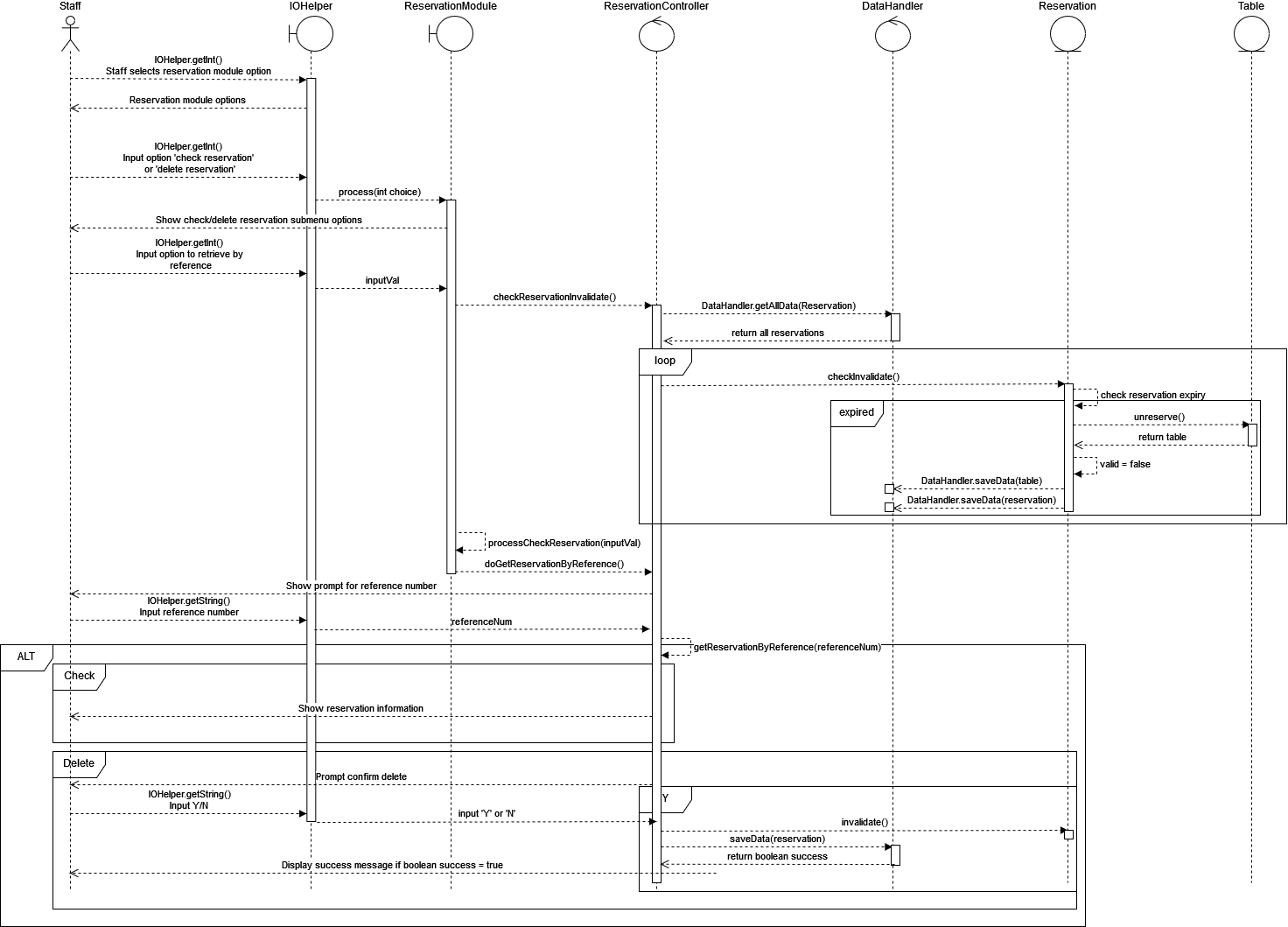
# 

# 

# 

# 

# Sequence Diagram for Checking/Removing Reservation



## Important/notable parts

**Removal of reservation that are past their expiry time**

Before any result is created,retrieved,updated or deleted from the data storage system for processing, ALL reservations in the system are retrieved and iterated. The sole purpose of this iteration is to invalidate all reservations that are past their expiry/decay time (fixed at 5 minutes after Reservation time in AppConstants.java). Hence every statement thereafter are free of reservations that are past their expiry date.

Invalidation of a reservation entry means the reservation’s ‘valid’ flag is set to false, and the update is persisted into the file system.

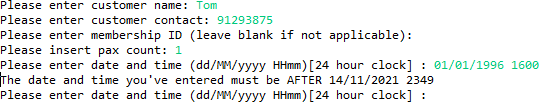
The general flow of invalidation is

1. Reservation flow triggered (check/create/delete reservation) from submenu
2. ReservationController.checkReservationInvalidate() is called
   1. All reservation is retrieved from data store
   2. Each reservation’s checkInvalidate() method is invoked
      1. If the reservation’s ‘reservationDecayDateTime’ is before the current date time, this means that it has expired and should be invalidated.
      2. From (i) if expired, valid is set to false. The related reserved table’s unreserve() method is invoked
      3. The changes to Table and Reservation are persisted to storage
3. The process only continues after the invalidation check has taken place to ensure expired data are not considered valid.

# Screen Captures

## Test Case 1

Making a reservation prior to current date

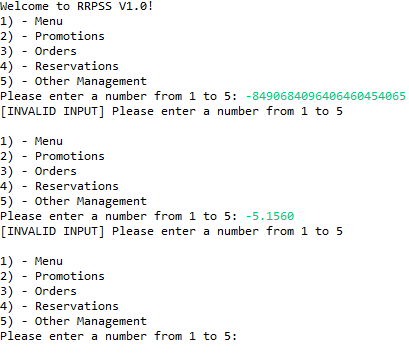


### Result

User is prompted over and over again until a valid input (advanced date and time) is provided.

## Test Case 2

Inserting unexpected value ranges as menu navigation options



### Result

User is informed of the invalid input and program loops without problems

## Test Case 3

Inserting blanks, and long string inputs

### **Result**

Prompt is repeated until a valid input is provided

## Test Case 4

Deleting the data folder and it’s contents

**Result**

Table data is seeded upon application restart, all other data produced at runtime does not encounter any error.

However, if the program is running and data is deleted, the program might encounter errors from data that is already loaded into memory.

# 

# Video Demonstration Link

<https://youtu.be/agjlU6_uLBM>

# Javadoc

All Javadoc files can be found in the {PROJECT}/javadoc folder

