**Software Design Document**

**Reservation System Accommodation Management Library**

**Sherwin Price**

**SWEN 646 9040: Software Design and Implementation**

**Course: ITEC-646-9040**

**Date: 09/23/2025**

Software Detail Design

# Table of Contents

# 1 Introduction

## 1.1 Purpose

This document describe the design and build of prototype software. The developed code operates as a stand-alone application, loaded on a business's local computer to manage reserved accommodations for accounts. This document is intended for Project Managers, Software Engineers, and anyone else who will be involved in the implementation of the system.

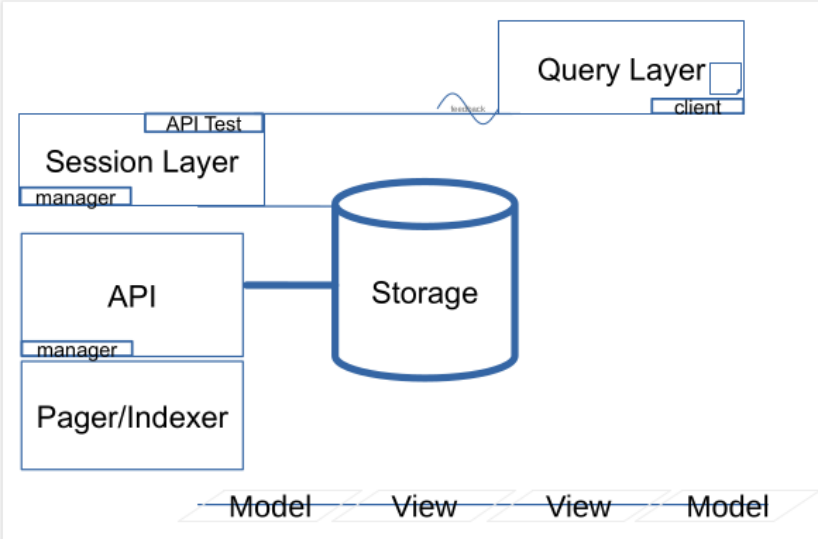
## 1.2 Scope

This software will be used and called by an User Interface (UI) as an API layer to exchange accounts and reservation messages. The UI screens will be touched on in this software design document (SDD), but will not be implemented as this project delivery.

## 1.3 Overview

This design uses the file-system for storage. System architecture will be based on Java coded modules as libraries, and follow Model-View-ViewModel pattern represented as:

The data-repository store on the file-system represents the storage component with modeled accounts and reservations stored in JSON formatted files. A single configured directory on host file-system contains all the account and reservation files.




Client driver Code will implement the Query Layer and UI View Layer.

Each managed account is saved in a separate directory named as the account's number. The account information is saved in one file and then each reservation that belongs to that account in separate file. The account file has all the reservation numbers associated with the account. Reservation file data is similarly saved in a separate file, with the associated account number. Accounts and Reservations are stored by the API manager class using the Java package library code and the data is stored on the file system.

### 1.4 Reference Materials

To validate that the data repository API works as intended, test code has been written and provided with the deployment to evaluate the deployed system. A single test class ‘TestReservations’ contains a main function that drives test functions. Each test case scenario executes as a separate method called by the main method. All methods should be able to run at once or select a single method to run by commenting out calls to the other methods. Comments in the source file document what each test case is verifying and doing. Each test case output a brief description of what is tested and then some output.


- Dea, C. (2023). *cognitive.* GitHub. https://github.com/carldea/cognitive
- JSON Lines. (2025). J*SON Lines: A convenient format for structured data.* https://jsonlines.org/
- Gluon. (2025). *OpenJFX 24 release notes*. Gluon. https://gluonhq.com/products/javafx/openjfx-24-release-notes/
- JFX Central. (2025). *JFX Central: JavaFX community hub*. JFX Central. https://www.jfx-central.com/
- OpenJDK. (2025)**.** *OpenJDK: JDK 24.* https://openjdk.org/projects/jdk/24/


### 1.5 Definitions Acronyms

- JSONL text-based data format where each line is a self-contained, valid JSON object.
- MVVM Model–view–viewmodel is an architectural pattern in computer software that facilitates the separation graphical user interface (GUI; the view) code from the development of the business logic.
- NA. Not Applicable.

## 2 System Overview

The AccomodationManager class is the api library manager and loads and updates storage for all the existing accounts and their associated reservations. API Manager feature is AccomodationManager class which is in the reservationsystem java archive(jar).

The manager is initialized with a directory to configure as a repository for the application, data will be loaded and saved in the repository through use of the AccomodationManager class.

```
public final class TestAccountLoad {
        // Request that Manager updates specific account's files with data stored in
        static void Test_AddAccount(AccomodationManager mgr, Account acct) throws Exception {
                mgr.AddAccount(acct);

                // 4. Request that Manager updates specific account's files with data stored in
                // memory
                mgr.UpdateAccount(acct);
        }
        Run | Debug
        public static void main(String[] args) throws Exception {

                // Configure data repository
                AccomodationManager mgr = new AccomodationManager(AccomodationManager.getRepositoryConfig.getPath());

                // 3. Add new account object to the list managed by Manager (if account object
                // already exists on add action with the same account number, it is considered
                // an error)
                Test_AddAccount(mgr, mgr.newAccount(phone_number:"701-456-7890",
                                new Address(street:"10 wilco ave", city:"wilco", state:"WY", zip:"82801"),
                                new EmailAddress(email_address:"wilco@wyommin.net")));

                Test_AddAccount(mgr, mgr.newAccount(phone_number:"701-456-7890",
                                new Address(street:"10 wilco ave", city:"wilco", state:"WY", zip:"82801"),
                                new EmailAddress(email_address:"wilco@wyommin.net")));

                mgr.showAccountList();
                System.out.println("Program Completed.");
        }
```

*Results:*
lodge.TestAccountLoad
Account A45098176 exists, duplicates not allowed.
Account A45098176
Program Completed.

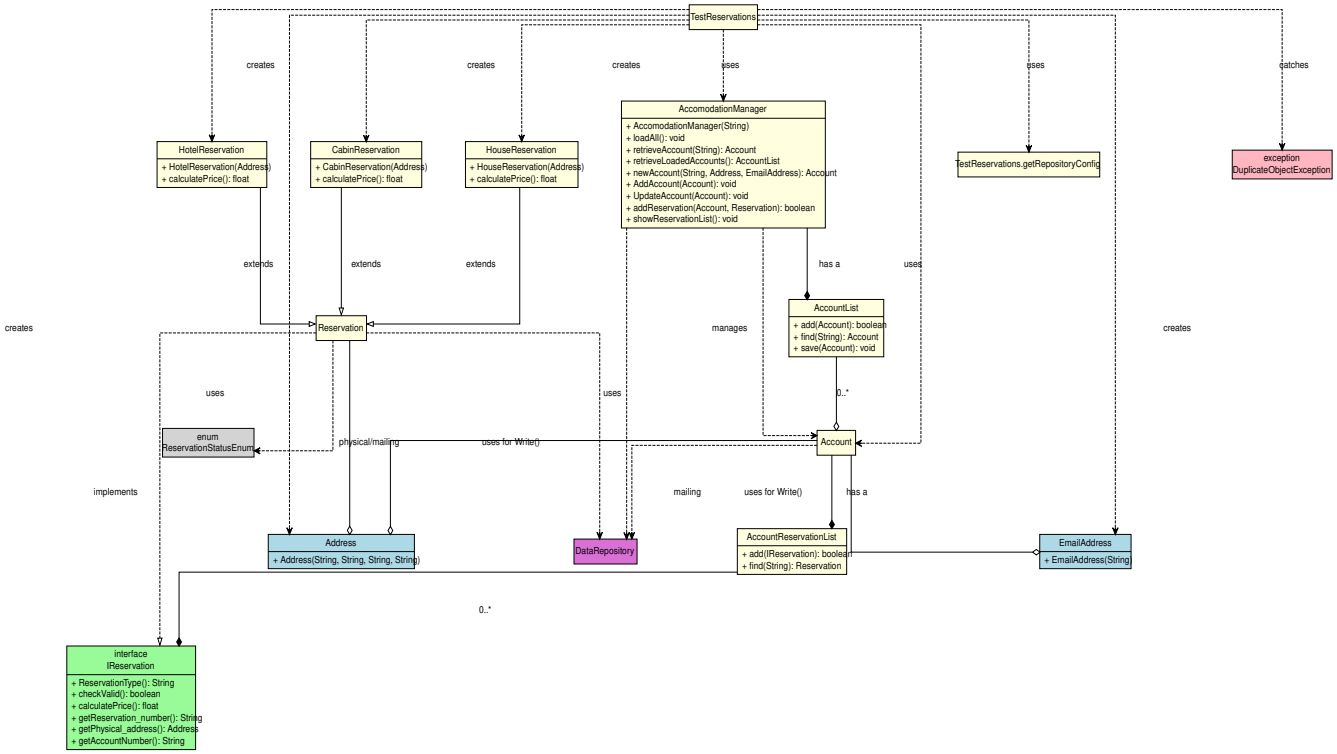
Modeled data files in the repository are formatted using a Line Delimited JSON format JSONL.

An application user interface will be considered in this design document. The architecture to be considered for implementation is Model-View-ViewModel(MVVM) design pattern which aides in separating application business and presentation logic. In this design the Model would be our reservationsystem Java package entities hosted in Java SE runtime. The view on the client can be coded using a multi-platform model with OpenJFX(Dea, C. 2023). The user will see and interact with the JavaFX loaded UI html views and JavaFX will handle event-handler call-back from specific view controller code in another Java package.

## 3 System Architecture

### 3.1 Architectural Design

Class diagram showing the major subsystems and data repositories and their interconnections.



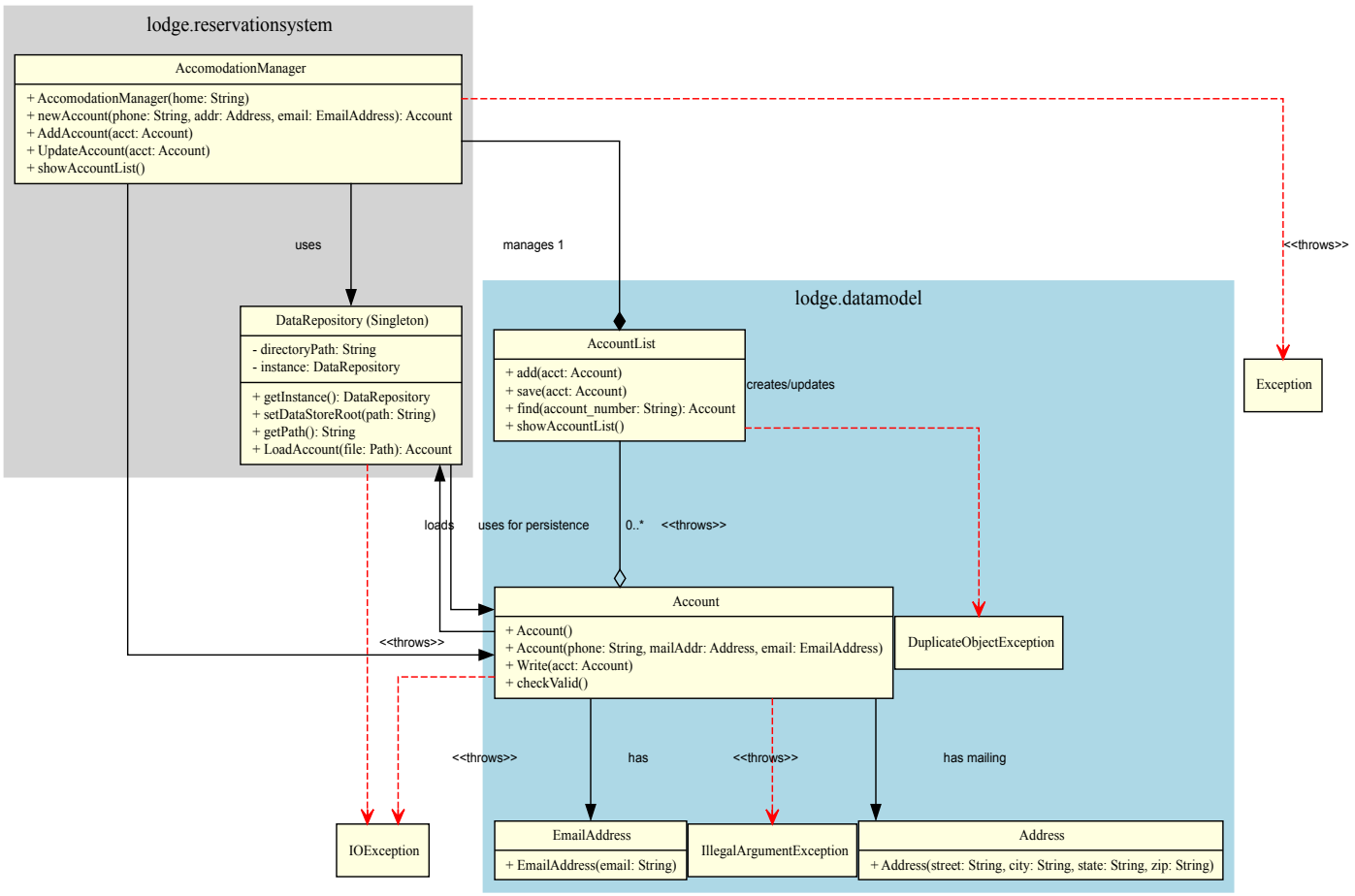

### 3.2 Decomposition Description

Above you see a decomposition of the subsystems in the design. A typical scenario to initialize the system a class would be written that instances the AccommodationManager class with it's constructor AccommodationManager(String) giving the storage folder path, then a list of accounts would be built. The account constructor requires that phone, address, and email be present so these will be entered one by one.

Data values for available accommodations would have to be collected with real property physical addresses. Then for each Account Reservation for accommodation at the physical location, records must be entered to the accounts(list) with the AccomodationManager class. The type polices for Cabins, Hotels, and Houses at the physical Address must be known. At this point the reservation system is loaded.

At this point the user could view the ui-layer and see the loaded views, and they would be able to set reservation dates, and complete reservations using the Command Menu to call Change on a Reservation selection, where the code would set one of the ReservationStatusEnum values on the reservation.

### 3.3 Exception Handling




Reservation System - Account Loading

Reservation System DataRepository and AccountList throw runtime exceptions as shown in the diagram. DuplicateObjectException can be thrown within the lodge.datamodel package code prior to add(Account) on AccountList. The error message set in the exception generation includes the causal data, such as account number that failed. This exception message information is displayed in the toString() output on the exception type. In addition IOException can be thrown during load of the application data files, in these cases the system will move on to the next file. Typically every account file in the directory will be attempted.

### 3.4 Design Rationale

The application is designed as a operational tool for booking of accomodation in a property with a physical address. Assuming that this sought of business is brisk, the user interface must load quickly and often, and be responsive to the business clerk information queries.
Therefore

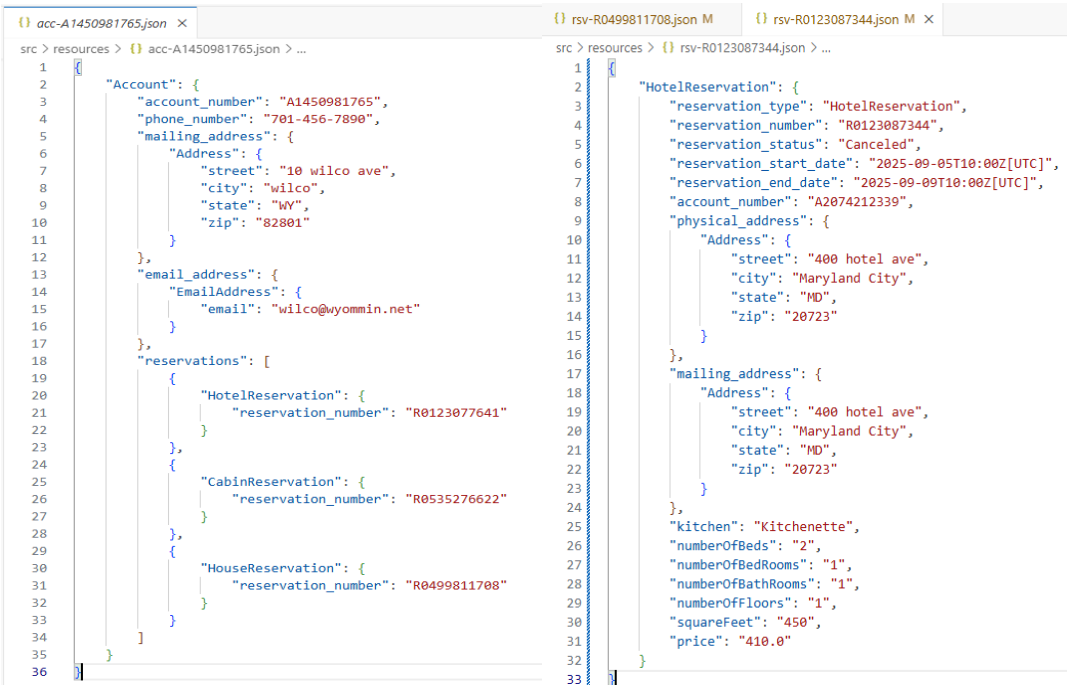
- storage is kept local to the host and data is immediately available to the user interface for queries, by keeping the data on the local file-system. This means data-queries can still occur when the system is not networked or disconnected.

- Language choice for the architecture uses Java to use the same code across multiple platforms such as Windows, Linux, Mac OS, IPhone and Android. In addition the long life-time of the language architecture preserves the original design coding investment. Along with these security features such as sand-boxing, secure class-loading, network security tooling and build tooling and local inter-processor communications(ipc) in the Java SE platform across Operating Systems make Java language a good choice.
- MVVM appear a viable choice because with in the JavaFX SDK there exist UI design tools that aide in the design of a modern app ui. This tool is called SceneBuilder by Gluon and allow all screens to be styled out. So the build tools already exits.
- In addition because we chose to build our storage layer with JSON, it is possible data-binding on the types in our data-layer-manager ( AccomodationManager class ) we can dual use the API in the server library and the client-side viewModel layer reducing the amount of code we write.

Together these points help us write the least amount of code to cover the requirements.

## 4 Data Design

Reservation System data-model entities are formatted as JSONL, and here we see there is a new line that separates JSON key/value pairs. This means the library content can be stream in new line blocks and when errors occur we will have a line number from the line in the file.



acc-A1450981765.json

src > resources > acc-A1450981765.json > ...

```
{
    "Account": {
        "account_number": "A1450981765",
        "phone_number": "701-456-7890",
        "mailing_address": {
            "Address": {
                "street": "10 wilco ave",
                "city": "wilco",
                "state": "WY",
                "zip": "82801"
            }
        },
        "email_address": {
            "EmailAddress": {
                "email": "wilco@wyommin.net"
            }
        },
        "reservations": [
            {
                "HotelReservation": {
                    "reservation_number": "R0123077641"
                }
            },
            {
                "CabinReservation": {
                    "reservation_number": "R0535276622"
                }
            },
            {
                "HouseReservation": {
                    "reservation_number": "R0499811708"
                }
            }
        ]
    }
}
```

rsv-R0499811708.json M  rsv-R0123087344.json M

src > resources > rsv-R0123087344.json > ...

```
{
    "HotelReservation": {
        "reservation_type": "HotelReservation",
        "reservation_number": "R0123087344",
        "reservation_status": "Canceled",
        "reservation_start_date": "2025-09-05T10:00Z[UTC]",
        "reservation_end_date": "2025-09-09T10:00Z[UTC]",
        "account_number": "A2074212339",
        "physical_address": {
            "Address": {
                "street": "400 hotel ave",
                "city": "Maryland City",
                "state": "MD",
                "zip": "20723"
            }
        },
        "mailing_address": {
            "Address": {
                "street": "400 hotel ave",
                "city": "Maryland City",
                "state": "MD",
                "zip": "20723"
            }
        },
        "kitchen": "Kitchenette",
        "numberOfBeds": "2",
        "numberOfBedRooms": "1",
        "numberOfBathRooms": "1",
        "numberOfFloors": "1",
        "squareFeet": "450",
        "price": "410.0"
    }
}
```

### 4.1 Data Description

The information domain of the system is transformed into JSON data structures using toString() function on the major entities. JSON parser library 'com.google.code.gson:gson:2.13.2' from google.com is used to read and write the structured data-records to the file-system. Parsing is done in the Java class DataRepository isolating the implementation parsing/loading to this class.

### 4.2 Data Dictionary

The main system entities are:

Class: DataRepository ( Singleton class ) - Repository handler

manages the storage location on the file-system, and ability to parse records.

Class: Account

Account stores unique customer information and reservation record references.

Class: Reservation {type: Cabin,Hotel,House}

Is an abstract class concretely implemented by classes CabinReservation, HotelReservation and HouseReservation. These derived classes of reservation allow the Java language polymorphism to allow the reservation class to behave differently based on their specific type. And this allow the reservationsystem to apply different

policies for reservations.

Class: Address

Address is used by Account mailing address and reservation physical address and mailing address

## 5 Component Design

In this section, we take a closer look at what each component does in a more systematic way. For OO description, summarize each object member function for all the objects listed in 3.2 in code or pseudocode. Describe any local data when necessary. Use the class template (Class Name, Class Description, Class Modifiers, Class Inheritance, Class Attributes, Exceptions Thrown, and Class Methods) for each class.

**Class Name:** lodge.reservationsystem.AccomodationManager
**Class Description/Purpose:** Manager fascade to library operations. Initialize the system.
**Exceptions Thrown:** DuplicateObjectException, IllegalOperationException, IOException
**Class Constructors:** AccomodationManager(String)
**Class Methods:**

loadAll:void
retrieveAccount(String):void
AddAccount(Account):void
addReservation(Account, Reservation):boolean
findReservation(String):Reservation

**Class Name:** lodge.reservationsystem.CabinReservation
**Class Description/Purpose:** Concrete reservation data class for reservation json storage record.
**Exceptions Thrown:** InvalidOperationException
**Class Constructors:** CabinReservation(Address)
**Class Methods:**

checkValid():boolean
calculatePrice():float
ReservationType():String

**Class Name:** lodge.reservationsystem.HotelReservation
**Class Description/Purpose:** Concrete reservation data class for reservation json storage record.
**Exceptions Thrown:** InvalidOperationException
**Class Constructors:** HotelReservation(Address)
**Class Methods:**

checkValid():boolean
calculatePrice():float
ReservationType():String

**Class Name:** lodge.reservationsystem.HouseReservation
**Class Description/Purpose:** Concrete reservation data class for reservation json storage record.
**Exceptions Thrown:** InvalidOperationException
**Class Constructors:** HouseReservation(Address)
**Class Methods:**
checkValid():boolean
calculatePrice():float
ReservationType():String

**Class Name:** lodge.data.AccountReservationList
**Class Description/Purpose:** List of reservations held owned by each account.
**Exceptions Thrown:** InvalidOperationException
**Class Constructors:** AccountReservationList()
**Class Methods:**
add(Reservation):Boolean
find(String):Reservation
update(AccountReservationList):void

**Class Name:** lodge.data.AccountList
**Class Description/Purpose:** Concrete account data class for account json storage record.
**Exceptions Thrown:** InvalidOperationException, DuplicateObjectException
**Class Constructors:** Account(String,String,Address,EmailAddress)
Class Methods:
static accountSerial(String,Address,EmailAddress):String
add(Account):Boolean
find(String):Account
save(Account):void

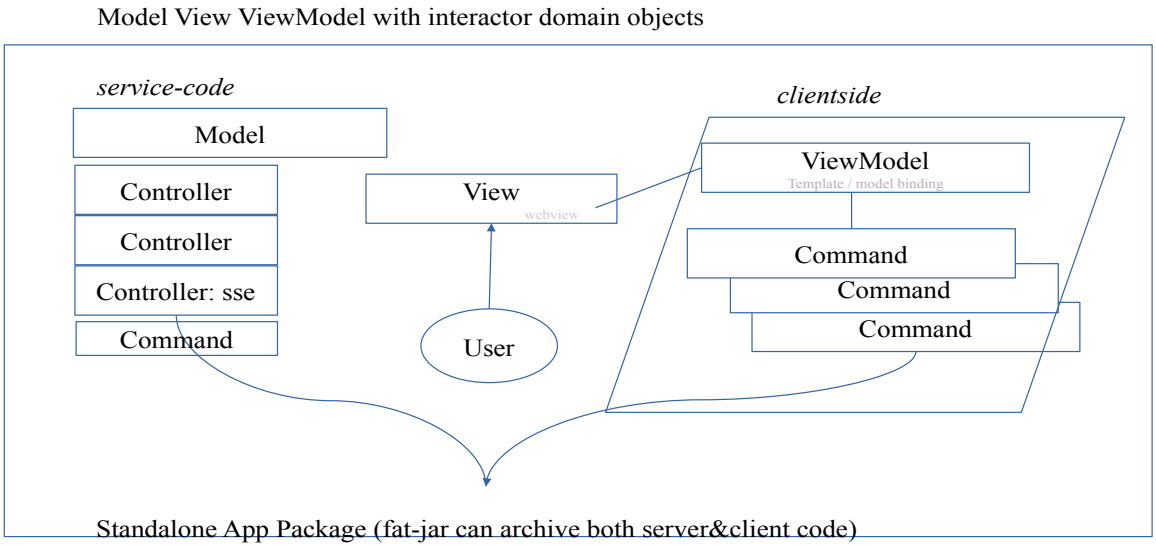
**Class Name:** lodge.data.Account
**Class Description/Purpose:** Concrete account data class for account json storage record.
Collects account attributes, and hash instances to enforce uniqueness.
**Exceptions Thrown:** InvalidOperationException
**Class Constructors:** Account(Address, String)
**Class Methods:**

**Class Name:** lodge.data.Address
**Class Description/Purpose:** Concrete Address data class for physical and mailing address.
**Exceptions Thrown:** na.
**Class Constructors:** Account(Address, String)
**Class Methods:**

## 6 Human Interface Design

### 6.1 Overview of User Interface

There are a number of hybrid cross platform application graphical user inferface frameworks JavaFx, and JCEF have community open source software libraries that target Android, IOS, Windows, and Mac OS. They can provide a presentation tier for our application. The framework in JavaFX favors separation of concern between business logic, data model, views. The framework implements messaging and binding component mapping commands to controllers through the ViewModel. All of this will be deployed standalone in a single app runtime hosted in Java SE(v24). Secondly deployment packaging can use JDK-24 jpackage to create install packages for major desktops platforms.



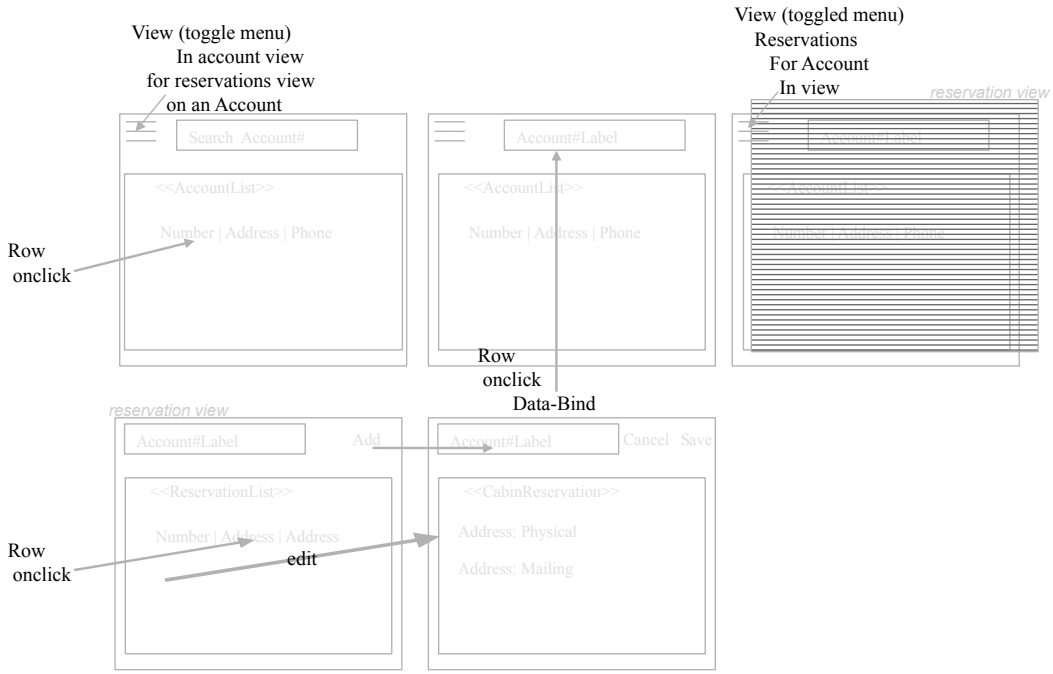

Session-layer interface support mapping Command callouts the controller, which mean on an asynchronous response, assuming the response is JSON, parse/load can be handled by the API AccommodationManager class. A fat-jar archive can package all the code with multiple jars to maintain separation of the data-layer and ui-layer. Then any view-ViewModel would initialize with a loading view as AccommodationManager completed load.

When the ui-layer view activates and calls a command for the accounts list, this would already have been loaded by AccommodationManager.GetListOfAccounts(). Binding in the viewModel component will render a list of account models, refreshing the user's ui-view-control.

Some software components that support the connection would be Server-Sent-Events(sse) a browser facility available in ECMAScript web browsers, and in JavaFX webview, where the web-ui platform library maintains the connection with the javascript runtime in the browser. An additional feature of Server-Sent-Events is the server can send events to the UI-Scripted library out-of-band after a secure-connection is initialized. The project will use only modern Java 24 with JEP 517, which upgrades the HTTP Client, to HTTP/3, to support our use case.

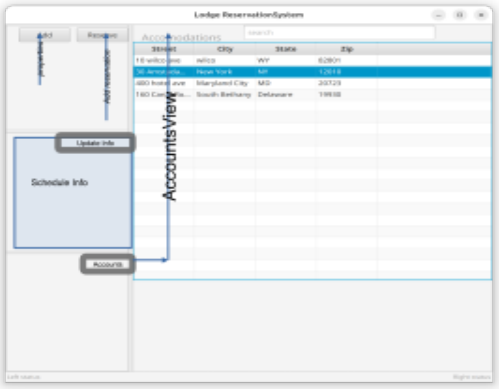
### 6.2 Screen Images

Display all the screenshots showing the interface from the user's perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible and they need to be complete - all screens that user would interact with need to be included. Make sure **all major functionality** is accounted for in the images.
(Graph paper works well.)




### 6.3 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.



## 7 Requirements Matrix

Provide a cross-reference that traces components and data structures to the requirements in your SRS document.

| | |
|---|---|
| RQX-0011 | AccomodationManager: Manager is initialized on system startup with All Accounts loaded. |
| RQX-0013 | AccountList: unique generated number (A followed by 9 characters) |
| RQX-0014 | AccomodationManager: Account is created with list of reservations[empty] |
| RQX-0015 | Account: with values in { account number mailing address phone number email address |
| RQX-0016 | Account: Account number can not be changed. |
| RQX-0017 | AccountList: Account can not be removed. |
| RQX-0018 | AccountList: No duplicated accounts allowed. |

# 8 APPENDICES

**This section is optional.**

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.

*Logged: Run*
cd reservationsystem
java -XX:+ShowCodeDetailsInExceptionMessages -cp ./bin/main:./libs/gson-2.13.2.jar:./libs/error_prone_annotations-2.41.0.jar lodge.TestReservations

*Logged: Test output on first run:*
Directory: reservationsystem/src/resources
Deserializing LoadAll Accounts 0
Error No Dups, Reservation exists: R0535276622.
lodge.datamodel.IllegalOperationException: Invalid Change, reservation has begun.
Hotel Per Night Rate: 120.000000
Account A1450981765: R0535276622, 40 cabin ave
Account A1450981765: R0499811708, 3000 Osage ave
Account A2074212339: R0123087344, 400 hotel ave
Account A2074212339: R2042828431, 30 cabin ave
Program Completed.

*Logged: Test output on second run:*
Directory: reservationsystem/src/resources
File: reservationsystem/src/resources/acc-A1450981765.json
File: reservationsystem/src/resources/acc-A2074212339.json
File: reservationsystem/src/resources/res-R0123087344.json
File: reservationsystem/src/resources/res-R2042828431.json
File: reservationsystem/src/resources/res-R0535276622.json
File: reservationsystem/src/resources/res-R0499811708.json
Deserializing LoadAll Accounts 2
Account A1450981765 exists, duplicates not allowed.
Account A2074212339 exists, duplicates not allowed.
Error No Dups, Reservation exists: R0123087344.
Error No Dups, Reservation exists: R2042828431.
Error No Dups, Reservation exists: R0535276622.
Error No Dups, Reservation exists: R0499811708.
Error No Dups, Reservation exists: R0535276622.
lodge.datamodel.IllegalOperationException: Invalid Change, reservation has begun.
lodge.datamodel.IllegalOperationException: Invalid Change, reservation has Canceled.
lodge.datamodel.IllegalOperationException: Invalid Change, reservation has begun.
Hotel Per Night Rate: 120.000000
Account A1450981765: R0535276622, 40 cabin ave
Account A1450981765: R0499811708, 3000 Osage ave
Account A2074212339: R0123087344, 400 hotel ave
Account A2074212339: R2042828431, 30 cabin ave
Program Completed.