Patrick Cullen - 12169892

Tirth Jaiswal - 12196525

COIT12200 Software Design and Development (T1 2024) – Assessment 3

# 

Contents

[Requirements Specification 3](#_Toc168957038)

[Use Case Diagram 3](#_Toc168957039)

[Functional requirements 3](#_Toc168957040)

[Non-Functional requirements 6](#_Toc168957041)

[System Architecture 7](#_Toc168957042)

[User Interface Design 8](#_Toc168957043)

[Use Case Descriptions 8](#_Toc168957044)

[Login Page 8](#_Toc168957045)

[Registration Page 9](#_Toc168957046)

[Main Menu Page 10](#_Toc168957047)

[Customer Search Page 12](#_Toc168957048)

[Create Customer Page 14](#_Toc168957049)

[Update Customer Page 14](#_Toc168957050)

[Property Search Page 15](#_Toc168957051)

[Create Property Page 16](#_Toc168957052)

[Update Property Page 17](#_Toc168957053)

[Job Search Page 18](#_Toc168957054)

[Create Job Page 19](#_Toc168957055)

[Update Job Page 20](#_Toc168957056)

[Manager Report Page 21](#_Toc168957057)

[Manager Report Page 21](#_Toc168957058)

[About Page 22](#_Toc168957059)

[Some Important Notes Regarding Data Entry and Searching in QPIMS 23](#_Toc168957060)

[Database Design 23](#_Toc168957061)

[SQL Scripts to Generate Tables 25](#_Toc168957062)

[SQL Sample Data 30](#_Toc168957063)

[Data Validation 31](#_Toc168957064)

[Testing 32](#_Toc168957065)

[Classes where connection is implemented: 32](#_Toc168957066)

[Testing Login Page: 33](#_Toc168957067)

[Testing Register Page: 37](#_Toc168957068)

[Testing MainMenu: 41](#_Toc168957069)

[Testing Customer: 47](#_Toc168957070)

[Testing create customer: 47](#_Toc168957071)

[Testing search customer: 52](#_Toc168957072)

[Testing search using phone number: 52](#_Toc168957073)

[Testing search using last name: 56](#_Toc168957074)

[Testing search using customer id: 59](#_Toc168957075)

[Testing update customer: 62](#_Toc168957076)

[Testing Properties: 65](#_Toc168957077)

[Testing create property: 65](#_Toc168957078)

[Testing search property: 71](#_Toc168957079)

[Testing search property using ‘Search by Address’: 71](#_Toc168957080)

[Testing search property with property Id: 76](#_Toc168957081)

[Testing update property: 79](#_Toc168957082)

[Testing Job class: 83](#_Toc168957083)

[Testing create Property: 83](#_Toc168957084)

[Testing search Job: 88](#_Toc168957085)

[Testing the ‘show all jobs’ 88](#_Toc168957086)

[Testing search jobs by JobID: 89](#_Toc168957087)

[Testing search job with propertyid: 92](#_Toc168957088)

[Testing Update Job: 95](#_Toc168957089)

[Testing Report: 98](#_Toc168957090)

[Testing About: 99](#_Toc168957091)

[User Manual 100](#_Toc168957092)

# Requirements Specification

## Use Case Diagram

A diagram of a person's network

Description automatically generated

## Functional requirements

In this section, functional requirements of core robust back end and front-end are discussed:

1. User login page:

* When the application run, the first page should be login page which provides security to the sensitive information in the system.
* The login page should contain two text fields and two buttons.
* The text fields are provided to enter Username and Password for login.
* First button should be ‘Login’ button, when user presses the button, user should be directed to customer page if the credential matches.
* Second button should be Registration button. If the user is not having registration with system, user needs to create a new registration.

1. Registration Page:

* This page is only used when the user is not having existing username and password.
* This page will take some important information from user like name, phone number, username, and preferred password.
* The Password entered in the registration page should be encrypted when storing to database.

1. Customer CRUD-Function page:

* After login, the next page connected should be customer page.
* Customer page will have buttons on top to navigate from customer to property, repair jobs or management view.
* Customer page will perform all the CRUD (Create, read, update, and delete) functions.
* Customer page relates to Customer table in database.
* Information field in customer page are: First name, Last name, Phone number, Address, and Customer ID.
* Customer ID is auto incremental in database; hence, user is not required to provide it. So, for the same reason, the CID text field is not editable. This text field will be just used to display the CID of search results.
* Functions/buttons available around customer are: ‘Add Customer’,’ Update customer’,’ search all customer’,’ search customer by ID’ and ‘search customer by Last name’.
* In Add customer, if any previous entries are open, the system will first clear those entry and then ask user to enter details. When Add customer is clicked, the data will add to database with unique and auto incremented ID.
* For update customer, user will need to search the customer using any of the search function. When user finds the needed customer to be updated then user easily make changes to detail and save it after clicking on update customer.
* User should use search all customer to search all the data present in the database.
* To find a specific customer with Last name or ID, user should use ‘search by last name’ or ‘ID’ function buttons. In ‘search by last name’, user has to enter last name in text field whereas for ‘search by ID’, user has to enter ID in specific available space just above button.
* There will be an exit button on every page to exit on the login screen.
* There will be an about button on every page to know how to use the system.

1. Property Management Page:

* This page is for property management which include two basic functions of searching and adding property.
* The primary key for data will be property address as it is always unique.
* Field information required for adding property are property address, property details like built year, no. of bathrooms, bedrooms, carparks, property manager name, property type (house or flat) and related customer ID. Each property should be related to just one customer.
* The search function is facilitated by address of property. The results of search will be displayed on Text area as it will be easy to display in property case.
* There will be an exit button on every page to exit on the login screen.
* There will be an about button on every page to know how to use the system.

1. Repair Job Management Page:

* Repair job requires following data fields: Auto incremental ID, description, booking date, completion date, charge, service staff name, and job type.
* Functions/buttons available are ‘View Booking’, ‘Cancel booking’, ‘update booking’ and ‘search booking’.
* There will be an exit button on every page to exit on the login screen.
* There will be an about button on every page to know how to use the system.

1. Statistics/ Management Report Page:

* On this page, there will be statistics of repair cost, minimum cost, maximum cost, average cost, and line graph to visualize the stats.
* This page will not have buttons as stats will be provided as soon as the page opens.
* There will be an exit button on every page to exit on the login screen.
* There will be an about button on every page to know how to use the system.

1. Database requirements (including functional relations between tables)

* My SQL workbench 8.0 CE has been used with Java Netbeans 15.
* Database altogether has 4 tables, namely, Login table, Customer table, property table and repair table.
* The code has been setup to connect automatic to the My SQL 8.0 server.

1. Other requirements:

* System is design in a way where all the errors are covered with proper display messages.
* Multiple tests are performed on every single function to test the functionality.

## Non-Functional requirements

In this section, non-functional requirements are covered:

1. Performance:

* There is no to very less time in data retrieval from database with provides with minimal time consumption.
* Functions are covered with error statement and display messages which makes the system better in performance and navigation.

1. Scalability:

* As My SQL Server 8.0 is used, the system shows potential of storing large amounts of data and ready for future growth.

1. Security:

* As the passwords are encrypted while saving them in database, the security of system is enhanced.
* Login page provides more security to the system as no one without credentials would not be able to enter.

1. Maintainability:

* The data is easy to maintain and update as there are functions embedded in system to update and maintain the data where required.
* For system maintenance, the code follows MVC to easy navigation and systematic approach for maintainability.

1. Usability:

* The system is easy to use due to user-friendly interface and GUIs.
* ‘About’ button is provided on each page for information around that page and how to functions.
* The function names are given according to the functioning for easy usage.
* All data input methods are easy.

1. Compatibility:

* Application runs on several operating systems like Microsoft, MAC OS and linux.

1. Integration:

* Integration with SQL server has been performed seamlessly and successfully.

# System Architecture

We chose to use MVC because we thought it would be simpler to create than MVP. With MVC, the project would be split into three parts. The **M**odel section would handle data and interact with the database, the **V**iew section would be responsible for GUI elements and user input, and the **C**ontroller section would handle switching between views and moving data between the view and the model. MVP would require the use of a presenter class and would have increased the complexity of the code.

A diagram of a user

Description automatically generated

# User Interface Design

## Use Case Descriptions

## Login Page

Login Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Login button |
| Actors | User, QPIMS and SQL server |
| Description | When the user enters their login information and clicks ‘Login’, the username and password hash will be checked against the corresponding entry in the database. If it matches, the user will be taken to the main menu screen. If it fails, incorrect username/password error message will be displayed. |
| Stimulus | User clicking ‘Login’ |
| Response | User’s username and password hash is checked against SQL database. If details are correct, user is taken to main menu page. Otherwise, incorrect username/password error is shown to user. |

Exit Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Login button |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Exit’ the application exits gracefully. |
| Stimulus | User clicking the ‘Exit button. |
| Response | The application exits gracefully. |

## Registration Page

Register Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Register button |
| Actors | User, QPIMS and SQL server |
| Description | When the user enters their first name, last name, email, username, and password, and clicks ‘Register’, a new user login entry will be created in the SQL database. |
| Stimulus | User clicking ‘Register’ |
| Response | User’s username and password hash is checked against SQL database. If details are correct, user is taken to main menu page. Otherwise, incorrect username/password error is shown to user. |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Go Back button |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Go Back’ it will take the user back to the login page. |
| Stimulus | User clicking ‘Go Back |
| Response | The user is returned to the login page. |

## Main Menu Page

Customers Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load customer search page |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Customers’ they are taken to the Customer Search page. |
| Stimulus | User clicking the ‘Customers’ button. |
| Response | The Customer Search page is loaded. |

Properties Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load property search page |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Properties’ they are taken to the Property Search page. |
| Stimulus | User clicking the ‘Properties button. |
| Response | The Property Search page is loaded. |

Jobs Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load job search page |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Jobs they are taken to the Job Search page. |
| Stimulus | User clicking the ‘Jobs’ button. |
| Response | The Jobs Search page is loaded. |

Report Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load Manager Report page |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Report’ they are taken to the Manager Report page. |
| Stimulus | User clicking the ‘Report button. |
| Response | The Manager Report page is loaded |

About Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load About page |
| Actors | User, QPIMS |
| Description | When the user clicks ‘About they are taken to the About page. |
| Stimulus | User clicking the ‘About’ button |
| Response | The ‘About’ page is loaded |

Log Out Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return user to login screen |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Log Out’ they are returned to the login page where they will be required to log in again. |
| Stimulus | User clicking the ‘Log Out’ button. |
| Response | The login screen is loaded |

## Customer Search Page

Search by Phone Number Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search for customers by phone number. |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a customer’s phone number and click ‘Search by Phone Number’. The phone number is compared against customer records in the database. Matching customer records are returned to QPIMS and displayed in the search result as a list. |
| Stimulus | User clicks ‘Search by Phone Number’ |
| Response | Matching customer entries are displayed in the search results. |

Search by Last Name Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search for customers by last name |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a customer’s last name and click ‘Search by Last Name. The last name is compared against customer records in the database. Matching customer records are returned to QPIMS and displayed in the search result as a list. |
| Stimulus | User clicks ‘Search by Phone Number’ |
| Response | Matching customer entries are displayed in the search results. |

Search by ID Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search for customers by ID |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a customer’s ID and click ‘Search by ID’. The ID is compared against customer records in the database. Matching customer records are returned to QPIMS and displayed in the search result as a list. |
| Stimulus | User clicks ‘Search by ID’ |
| Response | Matching customer entries are displayed in the search results. |

New Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load Manager Report page |
| Actors | User, QPIMS |
| Description | Upon clicking ‘New’ the user will be taken to the Create Customer Page where they can enter the details of a new customer to create a new entry. |
| Stimulus | User clicks ‘New’ button |
| Response | Create Customer page is loaded |

Update Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load Update Customer apge |
| Actors | User, QPIMS |
| Description | Upon clicking a search result and clicking ‘Update’ the user will be taken to the Update Customer Page where they can edit the details of a customer to update an entry |
| Stimulus | User clicks ‘Update’ button |
| Response | Update Customer page is loaded |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Go Back’ it will take the user back to the Main Menu page. |
| Stimulus | User clicking ‘Go Back’ |
| Response | The user is returned to the Main Menu page. |

## Create Customer Page

Add Customer

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new customer entries. |
| Actors | User, QPIMS, SQL server |
| Description | Upon entering the first name, last name, phone number, and address of a customer, the user clicks ‘Add Customer’. The entered data is checked for errors and checked against the SQL database to make sure the entry is unique. Finally, the new customer entry is added to the SQL database. |
| Stimulus | User clicks ‘Add Customer’ |
| Response | New customer is error checked and added to SQL database. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel customer creation and return to Customer Search page |
| Actors | User, QPIMS, SQL server |
| Description | Customer creation is cancelled, and user returns to search page |
| Stimulus | User clicks ‘Cancel’ |
| Response | User returns to Customer Search page. |

## Update Customer Page

Update Customer

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Update existing customer records. |
| Actors | User, QPIMS, SQL server |
| Description | User can update the first name, last name, phone number, and address of a customer, and click ‘Update Customer’. The entered data is checked for errors and the customer entry is updated in the SQL database. |
| Stimulus | User clicks ‘Update Customer’ |
| Response | Customer details are error checked and existing SQL customer entry is updated. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel customer updating and return to Customer Search page. |
| Actors | User, QPIMS, SQL server |
| Description | User can click ‘Cancel’ and return to the Customer Search page |
| Stimulus | User clicks ‘Cancel’ |
| Response | User is returned to Customer Search page |

## Property Search Page

Search by Address Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search properties by address |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a partial or complete address and click ‘Search by Address’. The address fields are compared against properties in the SQL database. Matching properties are then returned and displayed in the search results. |
| Stimulus | User clicks ‘Search by Address’ |
| Response | Matching property entries are displayed in the search results. |

Search by ID Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search properties by ID |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a property ID and click ‘Search by Address’. The entered ID is compared against properties in the SQL database. Matching properties are then returned and displayed in the search results. |
| Stimulus | User clicks ‘Search by ID’ |
| Response | Matching property entries are displayed in the search results. |

New Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Load Create Property page |
| Actors | User, QPIMS |
| Description | Upon clicking ‘New’ the user will be taken to the Create Property Page where they can enter the details of a new property to create a new entry. |
| Stimulus | User clicks ‘New’ button |
| Response | Create Property page is loaded |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Go Back’ it will take the user back to the Main Menu page. |
| Stimulus | User clicking ‘Go Back’ |
| Response | The user is returned to the Main Menu page. |

## Create Property Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new property |
| Actors | User, QPIMS, SQL server |
| Description | Upon entering the street number, street name, suburb, state, property type, number of bedrooms, number of bathrooms, number of parking spaces, managing agent, date of construction, and optional associated customer id, the user clicks ‘Create Property’. The entered data is checked for errors and checked against the SQL database to make sure the entry is unique. Finally, the new property entry is added to the SQL database. |
| Stimulus | User fills out property fields and clicks ‘Add Property’ |
| Response | New property is error checked and added to SQL database. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel property creation and return to Property Search. |
| Actors | User, QPIMS |
| Description | The user can click ‘Cancel’ and be returned to the Property Search page. |
| Stimulus | User clicks ‘Cancel’ |
| Response | User is returned to Property Search page. |

## Update Property Page

Update Customer

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Update existing property records. |
| Actors | User, QPIMS, SQL server |
| Description | Upon updating the property fields the user clicks ‘Update Property’. The entered data is checked for errors and the customer entry is updated in the SQL database. |
| Stimulus | User clicks ‘Update Property’ |
| Response | Property details are error checked and existing SQL customer entry is updated. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel property updating and return to Property Search page. |
| Actors | User, QPIMS |
| Description | User can click ‘Cancel’ and return to the Property Search page |
| Stimulus | User clicks ‘Cancel’ |
| Response | User is returned to Property Search page |

## Job Search Page

Search by Job ID Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search job by property ID |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a job ID and click ‘Search by Job ID. The entered ID is compared against existing jobs entries in the SQL database. Matching jobs are then returned and displayed in the search results. |
| Stimulus | User enters job ID and clicks ‘Search by Job ID’ |
| Response | The entered job ID is compared against job records in the SQL database and matching job entries are displayed in the search results. |

Search by Property ID Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Search jobs by property ID |
| Actors | User, QPIMS, SQL server |
| Description | The user can enter a property ID and click ‘Search by Property ID’. The entered ID is compared against job records in the SQL database. Matching jobs are then returned and displayed in the search results. |
| Stimulus | User enters a property ID and clicks ‘Search by Property ID’ |
| Response | The entered property ID is compared against job records in the SQL database and matching job entries are displayed in the search results. |

New Job Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Open Create Job page |
| Actors | User, QPIMS |
| Description | Upon clicking ‘New’ the user will be taken to the Create Job page where they can enter the details of a new job to create a new entry. |
| Stimulus | User clicks ‘New’ button |
| Response | Create Job page is loaded |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | When the user clicks ‘Go Back’ it will take the user back to the Main Menu page. |
| Stimulus | User clicking ‘Go Back’ |
| Response | The user is returned to the Main Menu page. |

## Create Job Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new job entry |
| Actors | User, QPIMS, SQL server |
| Description | Upon entering the associated property id, booking date, optional completion date, charge, service staff name, description, job type, and job status, the user clicks ‘Create Job. The entered data is checked for errors and checked against the SQL database to make sure the entry is unique. Finally, the new Job entry is added to the SQL database. |
| Stimulus | User fills out job creation fields and clicks ‘Create Job’ button. |
| Response | New job is error checked and added to SQL database. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel job creation and returns to Job Search page. |
| Actors | User, QPIMS |
| Description | User can click ‘Cancel’ and return to the Job Search page |
| Stimulus | User clicks ‘Cancel’ |
| Response | User is returned to Job Search page |

## Update Job Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new job entry |
| Actors | User, QPIMS, SQL server |
| Description | Upon editing the existing job’s details, the user clicks ‘Create Job’. The entered data is checked for errors and added to the SQL database. |
| Stimulus | User clicks ‘Update Job’ |
| Response | Job is error checked and existing SQL job entry is updated. |

Cancel Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Cancel job updating and return to Job Search page. |
| Actors | User, QPIMS |
| Description | User can click ‘Cancel’ and return to the Job Search page |
| Stimulus | User clicks ‘Cancel’ |
| Response | User is returned to Job Search page |

## Manager Report Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new job entry |
| Actors | User, QPIMS, SQL server |
| Description | The Manager Report page should have graph that correctly displays the distribution of completed jobs by job type. It should also show display the average, minimum, and maximum of job charges. |
| Stimulus | Manager Report page is loaded |
| Response | Graph correctly displays the distribution of completed jobs by job type and the average, minimum, and maximum of job charges. |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | User can click ‘Go Back and return to the main menu |
| Stimulus | User clicks ‘Go Back’ |
| Response | User is returned to the main menu |

## Manager Report Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new job entry |
| Actors | User, QPIMS, SQL server |
| Description | The Manager Report page should have graph that correctly displays the distribution of completed jobs by job type. It should also show display the average, minimum, and maximum of job charges. |
| Stimulus | Manager Report page is loaded |
| Response | Graph correctly displays the distribution of completed jobs by job type and the average, minimum, and maximum of job charges. |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | User can click ‘Go Back’ and return to the main menu |
| Stimulus | User clicks ‘Go Back’ |
| Response | User is returned to the main menu |

## About Page

Create Property

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Create new job entry |
| Actors | User, QPIMS |
| Description | The about page should display useful information explaining the purpose of QPIMS and a general overview of the main views. |
| Stimulus | The About page is loaded |
| Response | useful information explaining the purpose of QPIMS, and a general overview of the main views is displayed. |

Go Back Button

|  |  |
| --- | --- |
| System | QPIMS |
| Use Case | Return to main menu |
| Actors | User, QPIMS |
| Description | User can click ‘Go Back’ and return to the main menu |
| Stimulus | User clicks ‘Go Back’ |
| Response | User is returned to the main menu |

## Some Important Notes Regarding Data Entry and Searching in QPIMS

We have made the **customer Id** field in properties an optional field. We did this because there will be moments when the property will not have an associated customer, such as when the property is first purchased my QProperty, or during customer transitions.

We also made the **completion date** field in Jobs optional. This was done because the user may not know when a job will be completed, or it may never be completed in the case of a job cancellation.

When searching for properties, a complete address can be entered, or a partial date can be entered. This is possible because the properties are narrowed down further with each address field, rather than each field being a separate search query. As a result, it is also possible to list every property by leaving the address fields empty and clicking ‘Search by Address’.

# Database Design

|  |
| --- |
| PK: LoginID (INT) |
| LoginPassword (Varchar(45))  Email (Varchar(45))  Fname (Varchar(45))  Lname (Varchar(45)) |

Customer table:

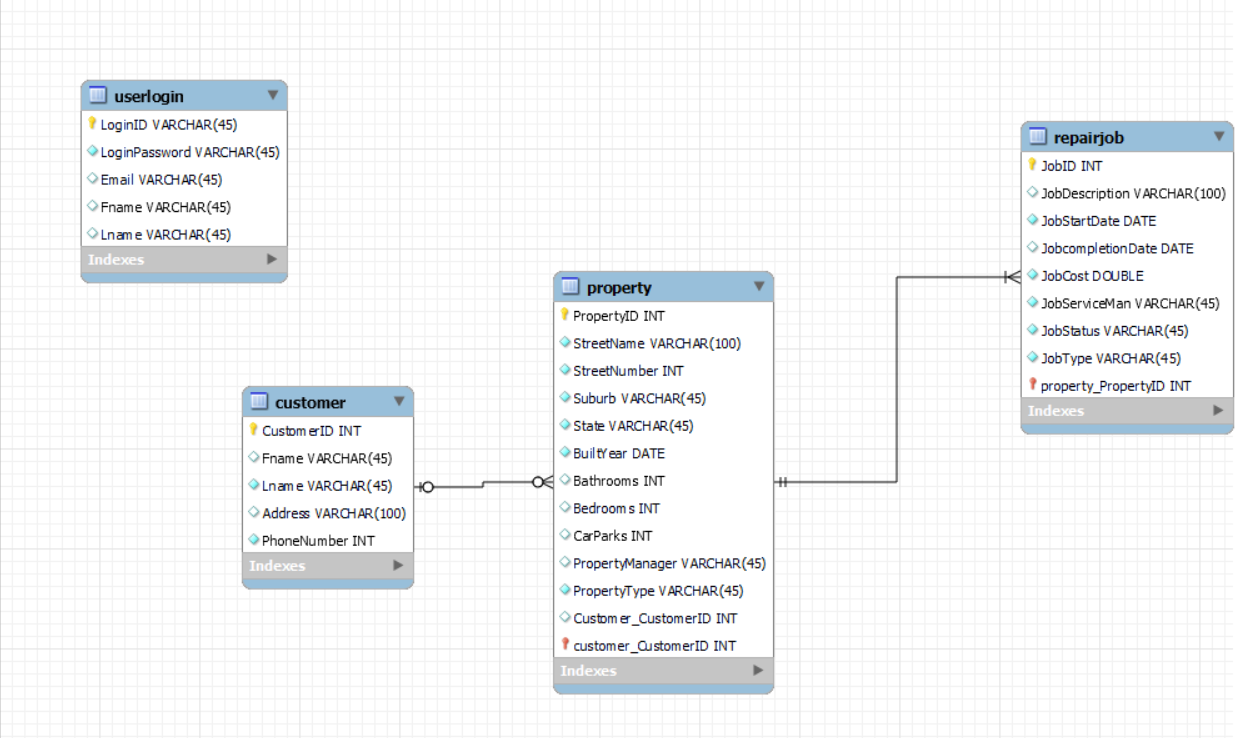
|  |
| --- |
| PK: CustomerID (INT) |
| Fname (Varchar(45))  Lname(Varchar(45))  Address (Varchar(100))  PhoneNumber (INT) |

Property table:

|  |
| --- |
| PK: PropertyID (INT) |
| StreetName (Varchar(100))  StreetNumber (INT)  Suburb (Varchar(45))  State (Varchar(45))  BuiltYear (Date)  Bathrooms (INT)  Bedrooms (INT)  CarParks (INT)  PropertyManager (Varchar(45))  PropertyType (Varchar(45)) |
| FK: Customer\_CustomerID (INT) |

Repair Table:

|  |
| --- |
| PK: JobID (INT) |
| JobDescription (Varchar(100))  JobStartDate (Date)  JobcompletionDate (Date)  JobCost (Double)  JobServiceMan (Varchar(45))  JobType (Varchar(45))  JobStatus (Varchar(45)) |
| FK: Property\_PropertyID (INT) |



## SQL Scripts to Generate Tables

|  |
| --- |
| Customers |
| --  -- Table structure for table `customer`  --  DROP TABLE IF EXISTS `customer`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!50503 SET character\_set\_client = utf8mb4 \*/;  CREATE TABLE `customer` (  `CustomerID` int NOT NULL AUTO\_INCREMENT,  `Fname` varchar(45) DEFAULT NULL,  `Lname` varchar(45) NOT NULL,  `Address` varchar(100) DEFAULT NULL,  `PhoneNumber` int NOT NULL,  PRIMARY KEY (`CustomerID`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb3;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  --  -- Dumping data for table `customer`  --  LOCK TABLES `customer` WRITE;  /\*!40000 ALTER TABLE `customer` DISABLE KEYS \*/;  INSERT INTO `customer` VALUES (1,'Adam','Harding','07553 Gay Station Suite 316, Bryanview, SC 62748',776422413),(2,'Brittany','Thomas','183 Yellow St, Emu Park 1902',1119089129),(3,'Thomas','Clewes','27 Hill St, Emerald, QLD, 4390',411293842),(4,'Harry','Larry','89 Elder st, Logan, QLD, 4401',993848371);  /\*!40000 ALTER TABLE `customer` ENABLE KEYS \*/;  UNLOCK TABLES; |
| Property |
| --  -- Table structure for table `property`  --  DROP TABLE IF EXISTS `property`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!50503 SET character\_set\_client = utf8mb4 \*/;  CREATE TABLE `property` (  `PropertyID` int NOT NULL AUTO\_INCREMENT,  `StreetName` varchar(100) NOT NULL,  `StreetNumber` int NOT NULL,  `Suburb` varchar(45) NOT NULL,  `State` varchar(45) NOT NULL,  `BuiltYear` date NOT NULL,  `Bathrooms` int DEFAULT NULL,  `Bedrooms` int DEFAULT NULL,  `CarParks` int DEFAULT NULL,  `PropertyManager` varchar(45) DEFAULT NULL,  `PropertyType` varchar(45) NOT NULL,  `Customer\_CustomerID` int DEFAULT NULL,  PRIMARY KEY (`PropertyID`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb3;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  --  -- Dumping data for table `property`  --  LOCK TABLES `property` WRITE;  /\*!40000 ALTER TABLE `property` DISABLE KEYS \*/;  INSERT INTO `property` VALUES (1,'Hill',22,'Bundaberg','QLD','1978-09-01',1,3,4,'Peter Griffin','House',NULL),(2,'Berlin',303,'Logan','QLD','1995-12-23',1,1,1,'Ronald McDonald','Apartment',2),(3,'Grenada',41,'Park Avenue','QLD','1963-08-08',2,4,3,'Peter Griffin','House',1),(4,'Helsburg',66,'Rockhampton City','QLD','1899-09-09',1,2,1,'Peter Griffin','Unit',4);  /\*!40000 ALTER TABLE `property` ENABLE KEYS \*/;  UNLOCK TABLES; |
| Repair job |
| --  -- Table structure for table `repairjob`  --  DROP TABLE IF EXISTS `repairjob`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!50503 SET character\_set\_client = utf8mb4 \*/;  CREATE TABLE `repairjob` (  `JobID` int NOT NULL AUTO\_INCREMENT,  `JobDescription` varchar(100) DEFAULT NULL,  `JobStartDate` date NOT NULL,  `JobcompletionDate` date DEFAULT NULL,  `JobCost` double NOT NULL,  `JobServiceMan` varchar(45) NOT NULL,  `JobStatus` varchar(45) NOT NULL,  `JobType` varchar(45) NOT NULL,  `property\_PropertyID` int NOT NULL,  PRIMARY KEY (`JobID`,`property\_PropertyID`),  KEY `fk\_repairjob\_property1\_idx` (`property\_PropertyID`),  CONSTRAINT `fk\_repairjob\_property1` FOREIGN KEY (`property\_PropertyID`) REFERENCES `property` (`PropertyID`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb3;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  --  -- Dumping data for table `repairjob`  --  LOCK TABLES `repairjob` WRITE;  /\*!40000 ALTER TABLE `repairjob` DISABLE KEYS \*/;  INSERT INTO `repairjob` VALUES (1,'Water leaking through roof in kitchen','2024-06-01',NULL,385.95,'Chad Gilmore','Ongoing','Structural',1),(2,'Entire house needed to be rewired to meet new standards','2023-09-09','2023-10-01',3500,'Chad Gilmore','Completed','Electrical',3),(3,'Cockroach infestation','2023-05-02',NULL,455,'Leslie Martin','Cancelled','Extermination',1),(4,'Mowed lawn','2024-05-05','2024-05-05',50,'Chad Gilmore','Completed','Gardening',3);  /\*!40000 ALTER TABLE `repairjob` ENABLE KEYS \*/;  UNLOCK TABLES; |
| --  -- Table structure for table `userlogin`  --  DROP TABLE IF EXISTS `userlogin`;  /\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;  /\*!50503 SET character\_set\_client = utf8mb4 \*/;  CREATE TABLE `userlogin` (  `LoginID` varchar(45) NOT NULL,  `LoginPassword` varchar(45) NOT NULL,  `Email` varchar(45) DEFAULT NULL,  `Fname` varchar(45) DEFAULT NULL,  `Lname` varchar(45) DEFAULT NULL,  PRIMARY KEY (`LoginID`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;  /\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;  --  -- Dumping data for table `userlogin`  --  LOCK TABLES `userlogin` WRITE;  /\*!40000 ALTER TABLE `userlogin` DISABLE KEYS \*/;  /\*!40000 ALTER TABLE `userlogin` ENABLE KEYS \*/;  UNLOCK TABLES;  /\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;  /\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;  /\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;  /\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;  /\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  /\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  /\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;  /\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/; |

## SQL Sample Data

|  |
| --- |
| Customer |
|  |
| Property |
|  |
| Jobs |
|  |
| Users |
|  |

# Data Validation

It is important that appropriate error checking and data validation measures are taken before data is moved from the application to the SQL database. Most of the data validation will be done at the controller/view level before it moves to the model. Any remaining data validation will be done at the model level, and no data validation will be done at the database level. For example, when a user enters the street number of a property, the application will need to ensure that an integer is entered before so that the application doesn’t crash, and so that dirty data isn’t sent to the database. It is also important to ensure that duplicates are not accidentally created when moving data from the application to the database. For example, if a customer entry is modified, it needs to override the previous entry rather than adding a new, separate entry.

# Testing

## Classes where connection is implemented:

The password and root name are implemented in ‘App Class’.

The connection is implemented in following classes:

1. CustomerModel
2. PropertyModel
3. JobModel

## Testing Login Page:

When no data is provided, and login is clicked:

A screenshot of a computer

Description automatically generated

When wrong credentials are provided:

A screenshot of a computer

Description automatically generated

When provided right credentials:

A screenshot of a login screen

Description automatically generated

A screenshot of a computer

Description automatically generated

When clicked on Register button:

The user clicked ‘Register’ button on the login page and it lend the user on the below page:

A screenshot of a computer

Description automatically generated

On pressing ‘Exit’ button, the system will successfully close.

## Testing Register Page:

If no details are provided:

A screenshot of a computer

Description automatically generated

If any detail is left to be filled:

A screenshot of a computer

Description automatically generated

If all details are provided and correct:

A screenshot of a computer

Description automatically generated

When pressed on ‘Cancel’ button:

A screenshot of a login screen

Description automatically generated

# Testing MainMenu:

A screenshot of a computer

Description automatically generated

When clicked on ‘Customers’ button, the FXML will switch to customer search fxml:

A screenshot of a computer

Description automatically generated

When clicked on ‘Properties’ button, the FXML will switch to Property search fxml:

A screenshot of a computer

Description automatically generated

When clicked on ‘Jobs’ button, the FXML will switch to Job search fxml:

A screenshot of a computer

Description automatically generated

When clicked on ‘Report’ button, the FXML will switch to report fxml:

A screenshot of a computer

Description automatically generated

When clicked on ‘About’ button, the FXML will switch to about fxml:

A screenshot of a computer screen

Description automatically generated

# Testing Customer:

## Testing create customer:

When clicked on ‘New’ button, the FXML will be changed to Create Customer where user can create customer:

A screenshot of a computer

Description automatically generated

In Creating customer, providing no details will result in following outcome:

A screenshot of a computer

Description automatically generated

If there is one or more customer details are missed, in this I have missed the First Name but in reality, it can be any details, the outcome will be:

A screenshot of a computer

Description automatically generated

If all the details are correctly provided, the outcome expected will be:

A screenshot of a computer

Description automatically generated

When clicked on the back button, the create customer FXML will be switched to search customer FXML:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Testing search customer:

### Testing search using phone number:

While searching using the phone number, if no phone number is provided, following output is expected:

A screenshot of a computer

Description automatically generated

If incorrect phone number is provided which does not exist in the system, the output is expected to be:

A screenshot of a computer

Description automatically generated

If the input format is used any other than integers, the output expected to be is:

A screenshot of a computer

Description automatically generated

Finally, if the input is correct:

A screenshot of a computer

Description automatically generated

### Testing search using last name:

While searching using the last name, if no input is provided, the output is expected to be:

A screenshot of a computer

Description automatically generated

If the incorrect last name is entered which is not present in the database, the expected outcome is:

A screenshot of a computer

Description automatically generated

Finally, if the correct information is provided as an input for last name:

A screenshot of a computer

Description automatically generated

### Testing search using customer id:

While searching with customer id, if the customer id is not provided:

A screenshot of a computer

Description automatically generated

If incorrect customer id is provided:

A screenshot of a computer

Description automatically generated

Finally, if the correct and existing customer id is provided as the input:

A screenshot of a computer

Description automatically generated

## Testing update customer:

When a customer entry from the list view is selected, the update button gets activated, customer can click on update the customer details by clicking on the update button:

**A screenshot of a computer

Description automatically generated**

When a detail is missing while updating the customer details, following outcome is expected:

A screenshot of a computer

Description automatically generated

When all the details are entered, the customer details will be update in database:

A screenshot of a computer

Description automatically generated

# Testing Properties:

## Testing create property:

While creating a property, if no details are provided, the outcome expected is:

A screenshot of a computer

Description automatically generated

If one or more details are missed, the outcome expected to be is:

A screenshot of a computer

Description automatically generated

If the format of input wrong, i.e. at the places where integer is required and string is provided, the outcome expected to be is:

A screenshot of a computer

Description automatically generated

If while creating a property, if the date format is not according to ‘yyyy-mm-dd’, the following outcome is expected:

A screenshot of a computer

Description automatically generated

If the customer ID provided is not in the database, following outcome is expected:

A screenshot of a computer

Description automatically generated

If all the details are valid and customer ID is existing, following outcome is expected:

A screenshot of a computer

Description automatically generated

## Testing search property:

### Testing search property using ‘Search by Address’:

The search by address works on inputs provided in text fields:

1. Street Number
2. Street Name
3. Suburb
4. Type

If anyone of the text input is provided, the search is going to take place with that one input but if more than one input is provided, the search is going to take place with combination of those inputs.

If no input is provided, all the result stored in database are expected to be displayed on list view:

A screenshot of a search

Description automatically generated

If combination of input is provided, the search will be happening with help of that combination.

For example, lets say, street name Tirth and type House is provided, is the result matching with this input fields will be displayed:

A screenshot of a search box

Description automatically generated

If single search input is provided, the following outcome is expected:

A screenshot of a search box

Description automatically generated

If incorrect input is provided, which does not match any result stored in the database, the following outcome is expected:

A screenshot of a computer

Description automatically generated

### Testing search property with property Id:

If the ID provided is not an integer, following outcome is expected:

A screenshot of a computer

Description automatically generated

If incorrect property Id is provided:

A screenshot of a computer

Description automatically generated

If correct property Id is provided:

A screenshot of a search box

Description automatically generated

## Testing update property:

If the property entry from the listview is selected, the update button will become active with will help in updating the selected property:

A screenshot of a search box

Description automatically generated

While updating, if any field is left to be filled, the following outcome is expected:

A screenshot of a computer

Description automatically generated

If the format of the fields are incorrect, the following outcome is expected:

A screenshot of a computer

Description automatically generated

If all details are correct and provided, the property will be updated:

A screenshot of a computer

Description automatically generated

# Testing Job class:

## Testing create Property:

Create page can be accessed with by clicking on ‘New Booking’ button on the search page:

A screenshot of a computer

Description automatically generated

While making a booking, if property Id is not provided, following outcome is expected:

A screenshot of a computer

Description automatically generated

StartDate is not provided or provided in a wrong format:

A screenshot of a computer

Description automatically generated

If any other data is not provided:

A screenshot of a computer

Description automatically generated

If all the details are provided and valid, the job will be created:

A screenshot of a computer

Description automatically generated

## Testing search Job:

### Testing the ‘show all jobs’

When clicked on show all the jobs, all the jobs stored in the database will be displayed on the listview:

A screenshot of a computer

Description automatically generated

### Testing search jobs by JobID:

If the ID entered is not an integer:

A screenshot of a computer

Description automatically generated

If the ID provided does not exist in the database:

A screenshot of a computer

Description automatically generated

If the job ID provided is correct and exists:

A screenshot of a computer

Description automatically generated

### Testing search job with propertyid:

If the property Id provided is not in integer format:

A screenshot of a computer

Description automatically generated

If the property Id provided does not exists:

A screenshot of a computer

Description automatically generated

If the property Id provided is correct and exists, the job related to that property is expected to be in listview:

A screenshot of a computer

Description automatically generated

# Testing Update Job:

Jobs can be updated by selecting them from the list view and clicking on the update button:

A screenshot of a computer

Description automatically generated

If any details are changes but details do not satisfy the system requirements, some outcome like below is expected:

A screenshot of a computer

Description automatically generated

If all the details are correct, the job will be updated.

Job update function will be majorly used from changing the status of job from ongoing to completed:

A screenshot of a computer

Description automatically generated

# Testing Report:

In this section, a bar graph is displayed which depicts the number of jobs with complete status. The x-axis on the bar graph signifies the classification of jobs like cleaning, electrician, etc. where the y-axis signifies the number of ‘completed’ jobs in that classified field.

The maximum charge, minimum charge and average charge are calculated on basis of all jobs regardless of completed, ongoing or cancelled.

A screenshot of a computer

Description automatically generated

# Testing About:

In this page of application, the information around different sections of the application i.e. customer, property and job are given. The about page looks like:

A screenshot of a computer

Description automatically generated

# User Manual

Queensland Property Information Management System (QPIMS) is a software system designed to efficiently collect and manage data about customers, properties under management, and maintenance services or repair jobs. The main view or interface comprises four distinct views: Customers, Properties, Repair Jobs, and Manager Report. Each view has a corresponding button that can be clicked to access the view. In addition to these four view buttons there is an About button that displays information explaining how to use the system. There is also a button to log out and return to the login screen.

In the customers view, users can search for customers by phone number or last name. Users can update an existing customer record by selecting a customer entry and clicking 'Update'. This will take them to a different view where they can edit the user’s details. Users can also create new customer records by clicking 'New User'. This will take them to a view where they can fill out the new user’s details.

In the properties view, users can search for existing properties by entering a full or partial address. Users can update an existing property record by selecting a property entry and clicking 'Update Property'. From here, users can change the property details, including the associated customer id. Users can also create new property records by clicking 'New Property. This will take them to a form that the user can fill out for the new property.

In the Jobs view, users can find existing maintenance/repair service records by searching by property id or job id. Users can update an existing job record by selecting a job entry and clicking 'Update Job'. From here, users can change the job details, update the job status, and update/set the completion date. Users can also create a new job record by clicking 'New Job'. This will take them to a form that the user can fill out to create a new job.