Garits

Requirements specification and system design

**Servlets**

**London**

**January 2019**

**Summary**

**This is a Software Requirements Document, a proposal to the marketing team of Quick Fix Fitters by the software development team at Servlets.**

**Authors**

**Beniamin Trip**

**Radu-Cristian Dumitrana**

**Glakho Robakidze**

**Martin Galabov**

**Edward Harikantra**

**Bao Pham Distribution**

**The Franchisee of Quick Fix Fitters and the Development team of Quick Fix Fitters.**

Table of Contents

[1 Preface 1](#_Toc1815795)

[1.1 Purpose and scope of this document 1](#_Toc1815796)

[1.2 History of the document 1](#_Toc1815797)

[2 Examine old system and the problems 1](#_Toc1815798)

[3 The new System Requirements 3](#_Toc1815799)

[4 Use case analysis 4](#_Toc1815800)

[4.1 Derived use cases 4](#_Toc1815801)

[4.3 Use Case Diagram 21](#_Toc1815802)

[5 System Design 23](#_Toc1815803)

[5.1 Package Diagram 23](#_Toc1815804)

[5.2 Design class Diagram 23](#_Toc1815805)

[5.2 Packages 25](#_Toc1815806)

[5.2.1 Accounts 25](#_Toc1815807)

[5.2.2 Job 26](#_Toc1815808)

[5.2.3 Stock Control 27](#_Toc1815809)

[5.2.4 Documents 28](#_Toc1815810)

[5.2.5 Database 29](#_Toc1815811)

[5.2.6 GUI 30](#_Toc1815812)

[3 Database 31](#_Toc1815813)

# 1 Preface

## Purpose and scope of this document

This is a software requirements document for GARITS and covers the functional and non-functional requirements with implementation constraints presented by the customer Quick Fix Fitters.

This document will firstly define the existing system and the problems the company faces using the current system. Secondly, it will show the requirement specification of the new system and go in details of how the new system will be designed and implemented.

## History of the document

The current version of this document is 0.3, In this version the formatting of the document was refined, Chapter 2 was added which looks at the problems with the current system at the company and provides the cause of those problems along with how the new system can be implemented to solve those problems.

All the versions and changes made in them is shown in the table below:

|  |  |
| --- | --- |
| **Version** | **Changes made** |
| 0.1 | This was the first draft of the document, when it was created, and the rough layout of the format was created. |
| 0.2 | The first chapter was expanded, and version history was added. |
| 0.3 | Chapter 2 was added where the problems in the old system are examined and detailed, along with the solutions that might be used in the new system to |
| 0.4 | Chapter 3 was added where the new system requirements are examined and stated in a spoken word manner. The design and analysis documents that will follow will be built using these requirements. |
| 1 | Chapter 4 was added which shows the use case analysis done on the system requirements, including derived use cases, use case specification, prioritization of the use cases and the use case diagram. |
| 2 | Chapter 5.1, and 5.2 was added, which includes the fully refined class diagram, and the package diagram |
| 3 | Chapter 5.3 was added, which shows the completed Entity Relationship diagram, which will be used to create the database of the GARITS System. |
| 3.1 | Chapter 5.3.2, which includes the full set of SQL statements, corrupting to the ER diagram shown in Chapter 5.3 |
| 4 | Chapter 5.4, was added which shows the GUI diagram, including screen shots of the different forms, showing the navigation through the system. |

# 2 Examine old system and the problems

Overview

This branch of Quick Fix Fitters PLC’s system overall has a lot of problems, and mainly they are all geared towards the fact that they don’t have a proper system. Most of the operations / jobs are recorded with no proper backup, not properly documented or not having an alternative source of storing the data and operations. Jobs for the mechanic is allocated on a whiteboard, everything is filled on sheets, outstanding invoices are recorded on a book with no proper alerting system to remind about the dates, stock control is being done by person with a list making edit with a pencil, … By not having a proper system in place, the operations are being done with no attentions to details, minor / ineffective supervising and could lead to possible workflow problems and issues that can’t be fixed when get out of hand.

List of all possible problems

|  |  |  |
| --- | --- | --- |
| **Problems** | **Possible Reasons** | **Possible Consequences** |
| Part shortages are happening more frequently | - Stock control for the parts are not being recorded properly (with pencil and papers)  - Stock control is being done by person (the ledger) and the person doing it is losing track of the stock levels and when to order replacements  - Restocking is only done monthly and is being written by a person | - Lost track of stock levels, leads to misleading information to customers and to workers in the workshop to carry out operations, wasting time and resources duo to misinformation  - Customers dissatisfaction duo to task being carried out took way too long duo to misinformation |
| Jobs are not being recorded and allocated properly to mechanics | - Job sheets are being recorded / assigned on a whiteboard  - Only the mechanics that is assigned to the job can keep track of the job they are doing | - Jobs are not being properly scheduled and assigned, lead to a waste of time and late on deadlines  - This lead to unwanted issues that affects the service that the customer can receive, e.g : receiving 2 invoices, their cars are not fixed on time, etc |
| Debts and overdue invoice are not properly recorded / keeping track and getting resolved | - Reminders for the debt are being sent by the administrator  - Debts / overdue invoices are recorded on a book  - Invoices are not being reminded, just being kept on a file and wait for the customer to come and pay | - Losing money on jobs  - Can be exploit by the customers to get free services  - losing out on profits and could lead to business collapsing |
| MoT Tests are not being reminded to customers accordingly / in time | - Customers details are being filled on cards that doesn’t have enough space for further important details  - Reminder system is non-existent | - Customers are losing out on benefits that they signed up / paid for  - Losing customer’s trust on the service provided => lose customer to competitors |
| Customer’s satisfaction level is low | - The service is not properly given, e.g. jobs are being delayed, not properly supervised, parts are not available to buy  - Communication and operations between the departments are very poor and this increases job response time to customers  - There are other alternatives that can provide similar service in quicker times and better quality | - Customers leaving the business => business losing out on sale => losing out on profit => failing as a business |

# 3 The new System Requirements

The new system is designed to tackle all the problem identified earlier with the current system, improve the everyday running of the business, make them more competitive, and offer a better service to their customers.

It will have the following requirements:

* Different levels of access:
  + **Administrator.** Deals with user accounts, adding, removing, changing them, and backing up and restoring the database. This role cannot do what the other roles do,
  + **Franchisee.** This role will have a full access to the functionality related to running the garage. It can set/alter a customer account, and is the only role that can configure “pay later” option and set/alter a customer’s discount plan, which can be one of the following:
    - a *fixed* discount
    - a *variable* discount
    - a *flexible* discount

This role, however, cannot assume any privileges associated to the Administrator role.

* **Foreperson.** This role will be able to do everything that the Mechanic and Receptionist can (specified below),
* **Mechanic.** They should be allowed to pick a job from the list of pending jobs and alter the job’s status by filling in details about their work on a vehicle.
* **Receptionist**. They shall be able to do the following
  + - Take in new jobs (typically this will be done together with the Foreperson who will provide the details of the tasks to be completed).
    - They should also be able to monitor the progress made with jobs (including searching for a specific job by car number, name of the customer, etc.).
    - Produce an invoice, once all tasks listed in the job sheet have been completed.
    - Do anything related to stock control, i.e. parts ordering, including searching the stock of spare parts (at least by model of the car, or type of the spare part) and reporting about any aspect related to spare parts.
* Generate the following reports:
  + number of vehicles booked in on a monthly basis, overall and per job type (MoT/annual service/repair), and type of customer (casual or Account Holder)
  + average time, and price, per job type (i.e. MoT/annual service/repair). This should be done overall, and per job type (MoT, annual service, repair, etc.), and / or given mechanic
  + stock-level, i.e. spare parts, report.

All types of reports must be both viewable and printable.

* Be relatively easy to use by everyone, through an appropriate GUI design.
* The system should calculate and print invoices automatically.
* The late payments, by Account Holders, should be detected automatically by the system and the Franchisee should be alerted about these.
* Print standard reminder letters automatically or on demand (as a batch, specific or individual)
* A stock control system to find parts, price them up, and update the stock level.

# 4 Use case analysis

This section is built upon the use case analysis of the above stated system requirements, it comprises of different subsections, each examining each step of the use case analysis process.

## 4.1 Derived use cases

In this section we show the use cases that were derived from the requirements, along with brief descriptions and their respective priorities in our system.

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case** | **ID** | **Priority Level** | **Justification** |
| BackUpDataBase | 1 | High | The database is essential to the system, as it will store many business data, essential to the running of the system and the business and backing it up regularly (e.g. hard disk) is crucial. This is also important to the project, as it poses time and implementation risks. |
| RestoreDataBase | 2 | High | This is because it’s very important to be able to, restore the database, if something goes wrong. The system heavily relies on the database, so being able to restore it is crucial. |
| LogIn | 3 | High | The requirement state, different levels of access must be implemented, as different users will have different roles in the system. |
| LogOut | 4 | High | The requirement state, different levels of access must be implemented, as different users will have different roles in the system. |
| ManageAccount | 5 | Medium | User account management is important, because, new user will come along and go, so it’s important to keep to data with this information |
| AddUserAccount | 6 | Medium | Without user interaction, the system will not be very useful, so adding user (and new users) is important, especially when new users come along later. |
| RemoveUserAccount | 7 | Medium | Employees might leave the job and keeping them as users if wasteful and not secure, therefore they must be deleted. |
| ChangeUserAccount | 8 | Medium | User account info might change, and so must be kept up-to-date. This is especially important when changing a user’s priority access, and roles. |
| SearchUserAccount | 9 | Low | In order to manage user account, one must search for it. This should be a trivial function. |
| AddJob | 10 | High | The system should allow for new jobs to be added to the system. This is an important for the garage, who takes in a lot of jobs, and for the users, who will complete them etc. |
| ProcessJob | 11 | Medium | The requirement specify that the Franchisee will process the jobs. |
| SearchJob | 12 | Low | Trivial function, that will search for a specific job, but will help when to find a job quicker and more efficiently. |
| RemoveJob | 13 | Medium | Once a job is completed, it must not be in the system anymore. Also, a job may be abandoned, and this must be deal with. |
| SelectFromJobList | 14 | Low | - |
| SpecifyWorkDescription | 15 | Medium | Specified by the mechanic, and important when generating invoice etc., so that he customer can see what has been done |
| AlterJobStatus | 16 | Medium | Specified by the mechanic, and important when generating invoice etc., so that he customer can see what has been done |
| AlterJobDuration | 17 | Medium | Specified by the mechanic, and important when generating invoice etc., so that he customer can see what has been done |
| SpecifySparePartsUsed | 18 | High | Specified by the mechanic, crucial for the spare parts stock, which must know and update itself accordingly. |
| SelectJob | 19 | High | A list of current jobs (pending list), important for mechanics, who will need to see the current jobs, and select them to complete. Also important for customer, as the jobs must be completed for them. |
| MonitorJobProgress | 20 | Medium | The receptionist, and the whole business, need to always be aware of the progress of a job, and act accordingly. |
| SearchForSpecificJob | 21 | Low | Trivial function, that will allow for the receptionist to monitor a job, by searching for it. Thus, its low priority. |
| ProductInvoice | 22 | High | Invoices must be created for records and for customers. This is important to the business, who needs them stored. |
| AllocateMechanic | 23 | Medium | The foreperson will need to allocate a specific mechanic for each job efficiently, so it’s important that this is done correctly and efficiently. |
| ManageCustomerAccount | 24 | Medium | Important to manage a customer account, including altering/setting it. |
| AlterCustomerAccount | 25 | Medium | Information might need to update such as address, and this must be known to the business (e.g. to send letters to correct address). |
| FindCustomerAccount | 26 | Low | Trivial function to find a customer. |
| SetCustomerAccount | 27 | High | For the purpose of the system, a customer account will need to be set (with all relevant info), to aid when sending letters, creating invoices etc. This will also help distinguish between, account holders and normal customers. |
| ConfigurePayLaterOption | 28 | Medium | Requirements state, that pay later options are available for account holders. |
| ManageDiscountPlan | 29 | Medium | Requirements state, several discounts that are available to the customer, and its important to manage it, to ensure correct discount given to the customer. |
| SetDiscountPlan | 30 | Medium | Requirements state, several discounts that are available to the customer, and setting one up, is important for the customer and business, who might want to attract more customer interaction through it. |
| AlterDiscountPlan | 31 | Medium | Requirements state, several discounts that are available to the customer, and switching between them, is important for the customer and business, who might want to attract more customer interaction through it. |
| CalculateFlexibleDiscount | 32 | Medium | This is part of the flexible discount and will calculate the discount at the end of the month, or on demand, to ensure customer get that amount back. Doing it correctly is important to the customer, and business image. |
| AlertOnStockLevel | 33 | High | It is pointed in the requirements, that keeping track of the stock level was a problem in the old system and must be implemented more efficiently in the new one, to allow efficient run of the business. |
| AlertOnLatePayments | 34 | High | When late payments are tracked, and found, an alert needs to be triggered, so that the garage is aware. |
| TrackLatePayments | 35 | High | The old system had a black book to keep track of customers who have not payed their services, but many times this was out of data, or data lost, meaning the business lost money. Thus, in the new system its crucial that a keeping track of any late payment is recorded. This is also crucial for the success of the project. |
| PrintReminderLetter | 36 | High | The business sends different reminder letters to customers, and its important for the business to expand itself. |
| PrintLatePaymentReminderLetter | 37 | High | Like the old system, reminder letter on late payments must be created and send, so that the customer is aware, and the business when acting if no customer response. |
| ManageSpareParts | 38 | Medium | Function to deal with managing spare parts. Similar to what the old system used to do. |
| UpdateSpareParts | 39 | Medium | Important to keep the spare parts up to data, so that appropriate action be taken when low. Not a high priority as it’s a trivial function. |
| UpdatePrice | 40 |  | Important to be able to update the price of the spare parts, so that the business sells them at the appropriate price. |
| SearchForSpareParts | 41 | Low | Trivial function, to make searching for spare parts easier. Like most of the search function, they are more of an add on, and not essential to the success of the system. |
| UpdateStock | 42 | Medium | Important to be able to update the stock, when parts are used etc. |
| OrderSpareParts | 43 | Medium | When stock below threshold, the system should order the needed parts, so that its always up to date. Important for customer, as they can always have stock available, and business who will also have this up to date stock always available. |
| CheckStockThreshold | 44 | Low | Important to be able to check what the current threshold level is. However, a trivial function, that does not bear so much weight on the system. |
| EditSparePartThreshold | 45 | Medium | Trivial, but important as, the requirements state that the appropriate user should be able to alter the threshold level, to meet the needs of the business. |
| GenerateReport | 46 | High | The requirements state that the business needs different reports, in order to run efficiently. Ensuring that they are generate correctly with the correct information can pose a project risk(time). |
| GenerateVehicleReport | 47 | High | The requirements state that the business needs different reports, in order to run efficiently. Ensuring that they are generate correctly with the correct information can pose a project risk(time). |
| GenerateAvrgReport | 48 | High | The requirements state that the business needs different reports, in order to run efficiently. Ensuring that they are generate correctly with the correct information can pose a project risk(time). |
| GenerateStockLevelReport | 49 | High | The requirements state that the business needs different reports, in order to run efficiently. Ensuring that they are generate correctly with the correct information can pose a project risk(time). |
| ViewReport | 50 | Medium | The reports should be viewable, to be checked, etc. |
| PrintReport | 51 | Medium | The reports should be available for printing, to meet different needs of the business, so its important to have this function, when dealing with reports. |

4.2 Use case specification

This section is dedicated to the use case specification of the 10 key use cases that we have picked out of the list of use cases.

|  |
| --- |
| **Use Case:** AddJob |
| **ID:** 10 |
| **Brief description:**  Taking in new job and add them to the pending list of jobs. |
| **Primary actors:** Foreperson, Receptionist, Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. System is operational 2. The user has successfully logged in the system |
| **Main flow:**   1. The use case starts when the user selects the addJob option in the system menu 2. The system prompts a form to the user. 3. User fills in the form and submits it to the system 4. The system establishes a connection with the database 5. The system copies the data to the database 6. The system disconnects from the database 7. The system informs the user that the job has been added to the list. |
| **Postconditions:**   1. The pending job list has been updated |
| **Alternative flows:**  connectionNotMade |

|  |
| --- |
| **Use Case:** AddJob:connectionNotMade |
| **ID:** 10.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Foreperson, Receptionist, Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has submitted the data for a new job |
| **Main flow:**   * 1. The use case starts at step 4 of the main flow when the system is unable to connect to the database to add the job to the pending job list.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user with main menu |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** SetCustomerAccount |
| **ID:** 27 |
| **Brief description:**  Adding the customer data to the database. |
| **Primary actors:** Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in successfully |
| **Main flow:**   1. The use case starts when the user selects the SetCustomerAccount option in the system menu 2. The system shows a form on the GUI. 3. User fills in the form and submits it to the system 4. The system establishes a connection with the database 5. The system copies the data to the database 6. The system disconnects from the database. 7. The system informs the user that the customer information has been added to the database |
| **Postconditions:**   1. The new customer data has been added to the database 2. The system successfully disconnected from the database after finishing adding the user |
| **Alternative flows:**  ConnectionNotMade |

|  |
| --- |
| **Use Case:** SetCustomerAccount:ConnectionNotMade |
| **ID:** 27.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has submitted the data for a new customer |
| **Main flow:**   * 1. The use case starts at step 4 of the main flow when the system is unable to connect to the database.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user with main menu |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** AlterCustomerAccount |
| **ID:** 25 |
| **Brief description:**  The franchisee can alter the customer accounts by searching for a specific customer and alter their data in the database. |
| **Primary actors:** Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in |
| **Main flow:**   1. The use case starts when the user selects AlterCustomerAccount from the main menu 2. The system connects to the database and gets all customer accounts 3. The customer account data is put in a java ArrayList 4. The system shows the user the list of customers on the GUI 5. The user highlights the account they want to alter and clicks the choose button 6. The system shows another form on the GUI with the customer account data 7. The user inputs the data they want to change 8. The user clicks the submit button 9. The System first deletes the customer they were changing in the database 10. The system saves the new data to the database. |
| **Postconditions:**   1. The new data has been successfully added to the database 2. The old data has been successfully deleted from the database |
| **Alternative flows:**  ConnectionNotMade |

|  |
| --- |
| **Use Case:** AlterCustomerAccount:connectionNotMade |
| **ID:** 25.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Foreperson, Receptionist, Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has submitted the data for the customer account they want to alter |
| **Main flow:**   * 1. The use case starts at step 2 of the main flow when the system is unable to connect to the database to retrieve the customer account.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user with main menu |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:**  AllocateMechanic |
| **ID:** 23 |
| **Brief description:**  The foreperson can allocate a mechanic to a job from the pending job list, which is done by selecting a particular job from the pending list and then selecting the mechanic which will do the job. |
| **Primary actors:** Foreperson |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The Foreperson has successfully logged in the system 3. The pending job list is not empty |
| **Main flow:**   1. The use case starts when the user chooses AllocateMechanic option in the main menu. 2. The system connects to the database 3. The system retrieves the list of pending jobs. 4. The system puts the list of pending jobs in an ArrayList 5. The system shows the user the list of jobs. 6. The user highlights a specific job by clicking it in the list. 7. The user clicks select, hence chooses the job 8. The system retrieves the list of mechanics from the database 9. The system shows the user a list of mechanics. 10. The user highlights the specific mechanic they want to assign to the job. 11. The user clicks ok. 12. The system notifies the user that the mechanic has been successfully allocated to the job 13. The system disconnects from the database. |
| **Postconditions:**   1. The system has successfully changed the details of the job 2. The system successfully disconnected from the database |
| **Alternative flows:**  ConnectionNotMade |

|  |
| --- |
| **Use Case:** AllocateMechanic:connectionNotMade |
| **ID:** 23.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Foreperson, Receptionist, Franchisee |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has submitted the data for the job |
| **Main flow:**   * 1. The use case starts at step 2 of the main flow when the system is unable to connect to the database to retrieve the job searched by the user.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user with main menu |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** ProduceInvoice |
| **ID:** 22 |
| **Brief description:**  The Receptionist should be able to produce an invoice for a recently finished job, which afterwards will be forwarded to the customer. This can be done by choosing the job the invoice should be done on from the GUI and getting a query to the database, which will fetch the information about the chosen job. |
| **Primary actors:** Receptionist, Foreperson |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user is logged in the system 3. There are jobs finished |
| **Main flow:**   1. The use case starts when the user chooses the PendingJobList from the GUI 2. The system fetches the pending job list and copies the data to an ArrayList 3. The system displays the list on the GUI 4. The user highlights the job they want to produce an invoice for 5. The user clicks produce invoice button on the side 6. The system queries the database for each task done for this job 7. For each task:    1. The system copies the Task into a task object in the loop    2. The system appends the data of the task to the end of the invoice    3. The system adds the cost to session.total 8. The system calculates labour cost 9. The system appends the price to the end of the document 10. The system calculates the VAT 11. The system adds the VAT to the total and appends it to the end of the invoice 12. The system disconnects from database. |
| **Postconditions:**   1. The data in the invoice is correct 2. The system successfully disconnected from the database |
| **Alternative flows:**  ConnectionNotMade |

|  |
| --- |
| **Use Case:** GenerateReport |
| **ID:** 46 |
| **Brief description:**  The system generates a report chosen by the user and shows it to the user in viewable and printable form. |
| **Primary actors:** Franchisee. |
| **Secondary actors:**  Database |
| **Preconditions:**   1. System is operational 2. The user has successfully logged in the system |
| **Main flow:**   1. The use case starts when the user selects Generate report from the main menu 2. The system shows a form on the GUI where the user can choose a specific type of report to generate 3. Connect to the Database 4. Execute database query with specified attributes 5. Format the query result 6. Present it to the user   Extension point: View Report, Print   1. Disconnect from Database |
| **Postconditions:**   1. Results successfully fetched 2. Successfully disconnected from the Database |
| **Alternative flows:**  **ConnectionNotMade**  **NoResult** |

|  |
| --- |
| **Use Case:** GenerateReport**:** ConnectionNotMade |
| **ID:** 46.1 |
| **Brief description:**  The system failed to establish a connection to the system and database. |
| **Primary actors:** Franchisee, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational 2. The user attempts to generate report, by filling in the required attributes |
| **Main flow:**   * 1. The use case starts at step 3 of the main flow when the system is unable to connect to the database to generate the report.      1. The system informs the user that it could not connect to the database.      2. The user acknowledges the notification.      3. The system prompts the user with main menu. |
| **Postconditions:**  None**.** |

|  |
| --- |
| **Use Case:** GenerateReport: NoResult |
| **ID:** 46.2 |
| **Brief description:**  The system failed to generate a report, with the specified attributes |
| **Primary actor:**  Franchisee, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational 2. Connection to the database is successful 3. The user attempts to generate report, giving some attributes (job type, etc.) |
| **Main flow:**   * 1. The use case starts at step 5 of the main flow when the system is unable to generate a report given the specified attributes      1. The system presents the user with a blank page, notifying the user that a report could not be specified given the context.      2. The user acknowledges the notification.      3. The system prompts the user to try again |
| **Postconditions:**  None**.** |

|  |
| --- |
| **Use Case:** logIn |
| **ID:** 3 |
| **Brief description:**  The function allows the user to get access to the system & database by proving their own identity through an username and password that only that particular person knows. |
| **Primary actors:** Administrator, Franchisee, Foreperson, Mechanic, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational |
| **Main flow:**   1. The use case started when the user selects the logIn function in the main menu. 2. The system prompts a form for the user. 3. User fills in the form and submits it to the system. 4. The system establishes a connection with the database. 5. The system matches the information on the form that the user filled in to the database. 6. If (while logIn exists):   6.1) The information on the form matches the information on the system about the user  6.2) The information on the form doesn’t match the information on the system about the user  6.2.1) System prompts a notification that notices the user that the information filled in is wrong and adjustment is required.  6.2.2) Loop back to 2).   1. The system prompts the user their profile page. |
| **Postconditions:**   1. User get access to their own profile page and get access to all the function that their role allowed them to perform on the system. |
| **Alternative flows:**  ConnectionNotMade  logInDetailsAreWrong |

|  |
| --- |
| **Use Case:** logIn**:**ConnectionNotMade |
| **ID:** 3.1 |
| **Brief description:**  The system failed to establish a connection to the system and database. |
| **Primary actors:** Administrator, Franchisee, Foreperson, Mechanic, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational 2. The user attempts to perform logIn function by filling the form and submits |
| **Main flow:**   * 1. The use case starts at step 6 of the main flow when the system is unable to connect to the database to allow the user to log in.      1. The system informs the user that it could not connect to the database.      2. The user acknowledges the notification.      3. The system prompts the user with main menu. |
| **Postconditions:**  None**.** |

|  |
| --- |
| **Use Case:** logIn:logIndetailsAreWrong |
| **ID:** 3.3 |
| **Brief description:**  The system can’t match the information in the form filled by the user with the system to verify the user. |
| **Primary actors:** Administrator, Franchisee, Foreperson, Mechanic, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational 2. The user attempts to perform logIn function by filling the form and submits. |
| **Main flow:**  6.2) The use case starts at step 6 of the main flow when the system is unable to match the information on the form to the information on the system about the user  6.2.1) System prompts a notification that notices the user that the information filled in is wrong and adjustment is required.  6.2.2) User acknowledges the notification  6.2.3) Loop back to step 2) in the main flow. |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** addUserAccount |
| **ID:** 6 |
| **Brief description:**  Adding new user account for employees in the business branch. |
| **Primary actors:** Administrator |
| **Secondary actors:**  Database |
| **Preconditions:**   1. System is operational 2. The user has successfully logged in the system |
| **Main flow:**   1. The use case starts when the user selects the addUserAccount option in the system menu 2. The system prompts a form to the user. 3. User fills in the form and submits it to the system 4. The system establishes a connection with the database 5. The system copies the data to the database 6. The system disconnects from the database 7. The system informs the administrator that the new user has been added to the database. |
| **Postconditions:**   1. The user list in the database is updated with one new user. |
| **Alternative flows:**  connectionNotMade |

|  |
| --- |
| **Use Case:** addUserAccount:connectionNotMade |
| **ID:** 6.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Administrator |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has submitted the data for a user account |
| **Main flow:**   * 1. The use case starts at step 4 of the main flow when the system is unable to connect to the database.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user with main menu |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** logOut |
| **ID:** 4 |
| **Brief description:**  The function allows user to remove their access to their account and their ability to perform functions on the system. |
| **Primary actors:** Administrator, Franchisee, Foreperson, Mechanics, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. System is operational 2. The user has successfully logged in the system |
| **Main flow:**   1. The use case started when the user performs the logOut function on the main menu. 2. The system establishes a connection to the system 3. The system removes the user account’s access to the database and the system 4. The system switches off the user account 5. The system disconnects from the database 6. The system prompts the main menu |
| **Postconditions:**   1. The user is switched off and get their access to the database removed. |
| **Alternative flows:**  connectionNotMade |

|  |
| --- |
| **Use Case:** logOut:connectionNotMade |
| **ID:** 4.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Administrator, Franchisee, Foreperson, Mechanics, Receptionist |
| **Secondary actors:**  System, Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has attempted to perform logOut function |
| **Main flow:**   * 1. The use case starts at step 2 of the main flow when the system is unable to connect to the database.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user back to their original destination |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** updateStock |
| **ID:** 42 |
| **Brief description:**  This function allows the user to make connection to the database and readjust and update the stock level by manually inputting the stock data. |
| **Primary actors:** Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user is logged in |
| **Main flow:**   1. The use case started when the user performs the updateStock functions when stock notifications are on 2. The system prompts out a form for user to fill out the stock that they are updating and the amount 3. User fills in the forms and submit to the system 4. The system makes a connection to the database 5. The system copies the data to the database 6. The system disconnects from the database 7. The system then informs the user that the stock level in the database has been updated. |
| **Postconditions:**   1. The new order has been set 2. An estimated time to receive the order are being shown to the user |
| **Alternative flows:**  connectionNotMade |

|  |
| --- |
| **Use Case:** updateStock:connectionNotMade |
| **ID:** 42.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has attempted to perform updateStock function |
| **Main flow:**   * 1. The use case starts at step 4 of the main flow when the system is unable to connect to the database.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user back to their original destination |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case: Print** |
| **ID:** 51 |
| **Brief description:**  This function allows user to print out any type of documents such as reports, invoice, job sheets, etc… |
| **Primary actors:** Franchisee, Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user is logged in |
| **Main flow:**   1. The use case when user performs the function print function in the main menu 2. The system then prompts a list of possible documents that the user can print 3. The user then picks the document they want to perform the function 4. The system then confirms and show the documents that the user wants to print 5. The system then prompts a list of conditions that narrows down the specific conditions that user wants to print the document in 6. The user then selects the conditions they want from the list and confirm it. 7. The system makes a connection to the database 8. While (print exists):   7.1) System then arrange the final documents that has all the conditions gathered  7.2) System then prompts the final document for the user to ask for confirmation   1. The user then confirms the document that is shown to them 2. The user then performs the print function 3. The system then translates the document and do the printing function. |
| **Postconditions:**   1. The system successfully prints out the document in the correct format. |
| **Alternative flows:**  connectionNotMade  noDocumentsAccumulated  adjustmentNotMade |

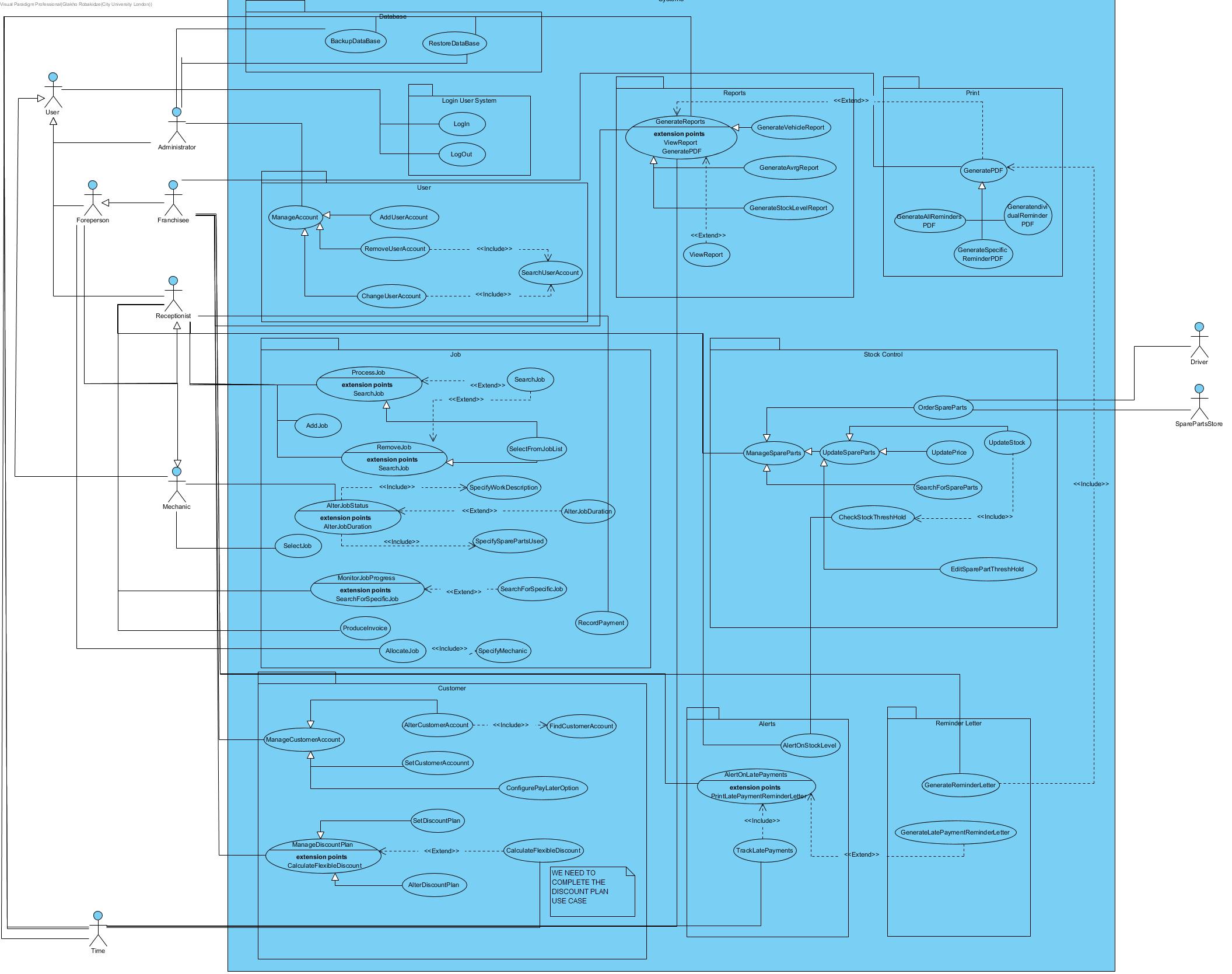
|  |
| --- |
| **Use Case:** print:connectionNotMade |
| **ID:** 51.1 |
| **Brief description:**  The system could not establish a connection to the database |
| **Primary actors:** Franchisee, Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has attempted to perform print function |
| **Main flow:**   * 1. The use case starts at step 6 of the main flow when the system is unable to connect to the database.   2. The system informs the user that it could not connect to the database.   3. The user acknowledges the notification.   4. The system prompts the user back to the condition list. |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** print:noDocumentsAccumulated |
| **ID:** 51.2 |
| **Brief description:**  The system could not find the documents available in the database of the computer. |
| **Primary actors:** Franchisee, Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has attempted to perform print function |
| **Main flow:**   * 1. The use case starts at step 2) of the main flow when the system is unable to find the document available in the database   2. The system informs the user that it could not find the documents in the system.   3. The user acknowledges the notification.   4. The system prompts the user back to the main menu. |
| **Postconditions:**  None. |

|  |
| --- |
| **Use Case:** print:adjustmentNotMade |
| **ID:** 51.3 |
| **Brief description:**  The system could not perform the printing function based on the printing settings that the user set up for the document to send to the printer. |
| **Primary actors:** Franchisee, Receptionist |
| **Secondary actors:**  Database |
| **Preconditions:**   1. The system is operational 2. The user has logged in 3. The user has attempted to perform print function |
| **Main flow:**   * + 1. The use case starts at step 8.1) of the main flow when the system is unable to perform the print function based on the options to print that is being sent by the user     2. The system informs the user that it could not print based on the print option list that the user sent.     3. The user acknowledges the notification.     4. The system prompts the user back to the print option list. |
| **Postconditions:**  None. |

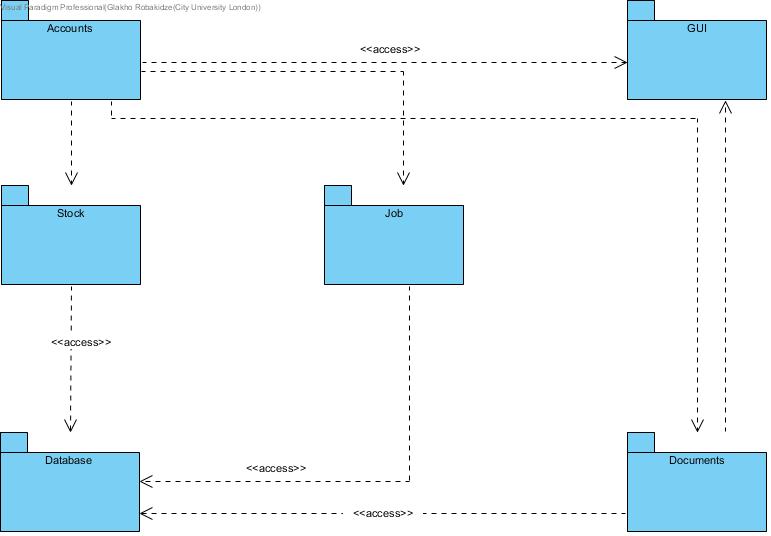
## 4.3 Use Case Diagram

This section shows the finished use case diagram of the system.



# 5 System Design

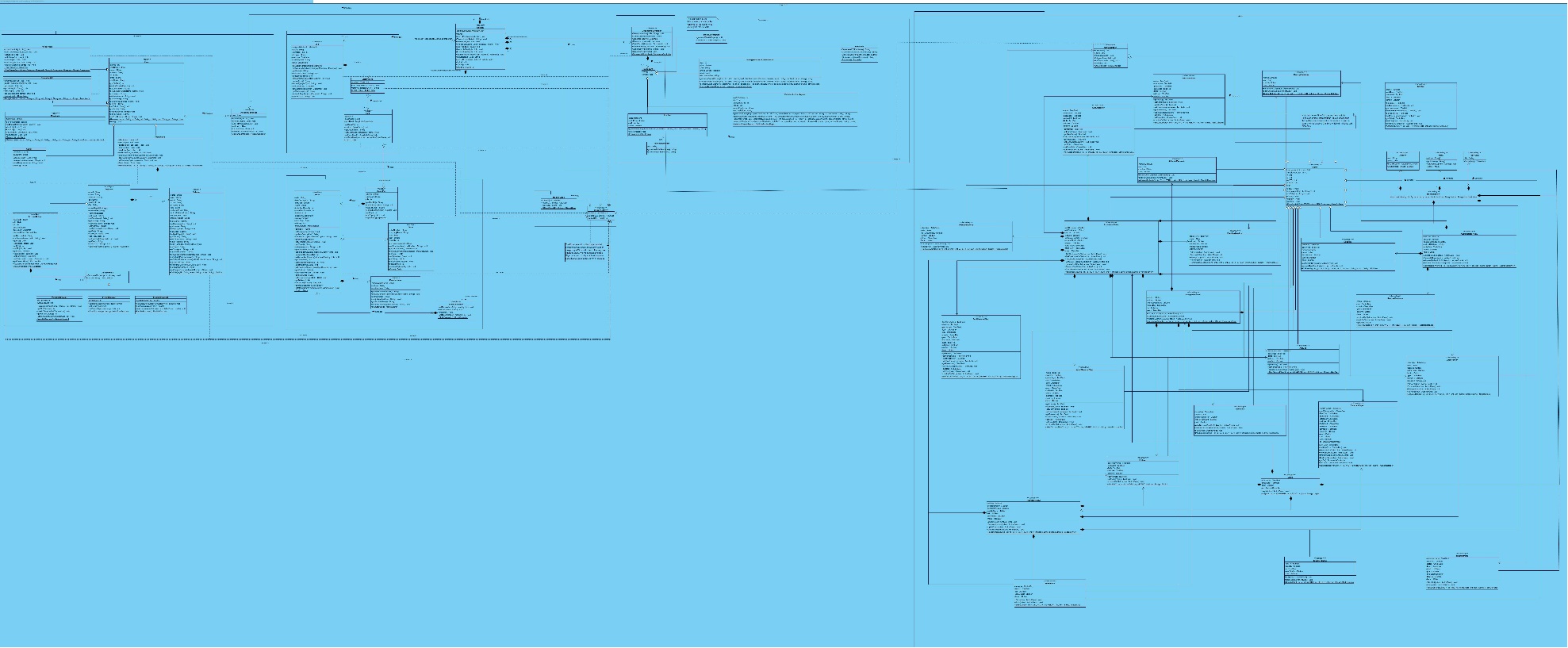
This section shows the design of the system, it includes a package diagram, a design class diagram of the whole system and a closer view of each package in the system.

5.1 Package Diagram

While designing the system we tried to include as least number of packages as possible, in order to make the diagram less complicated, so the packages categorize the classes as generally as possible.

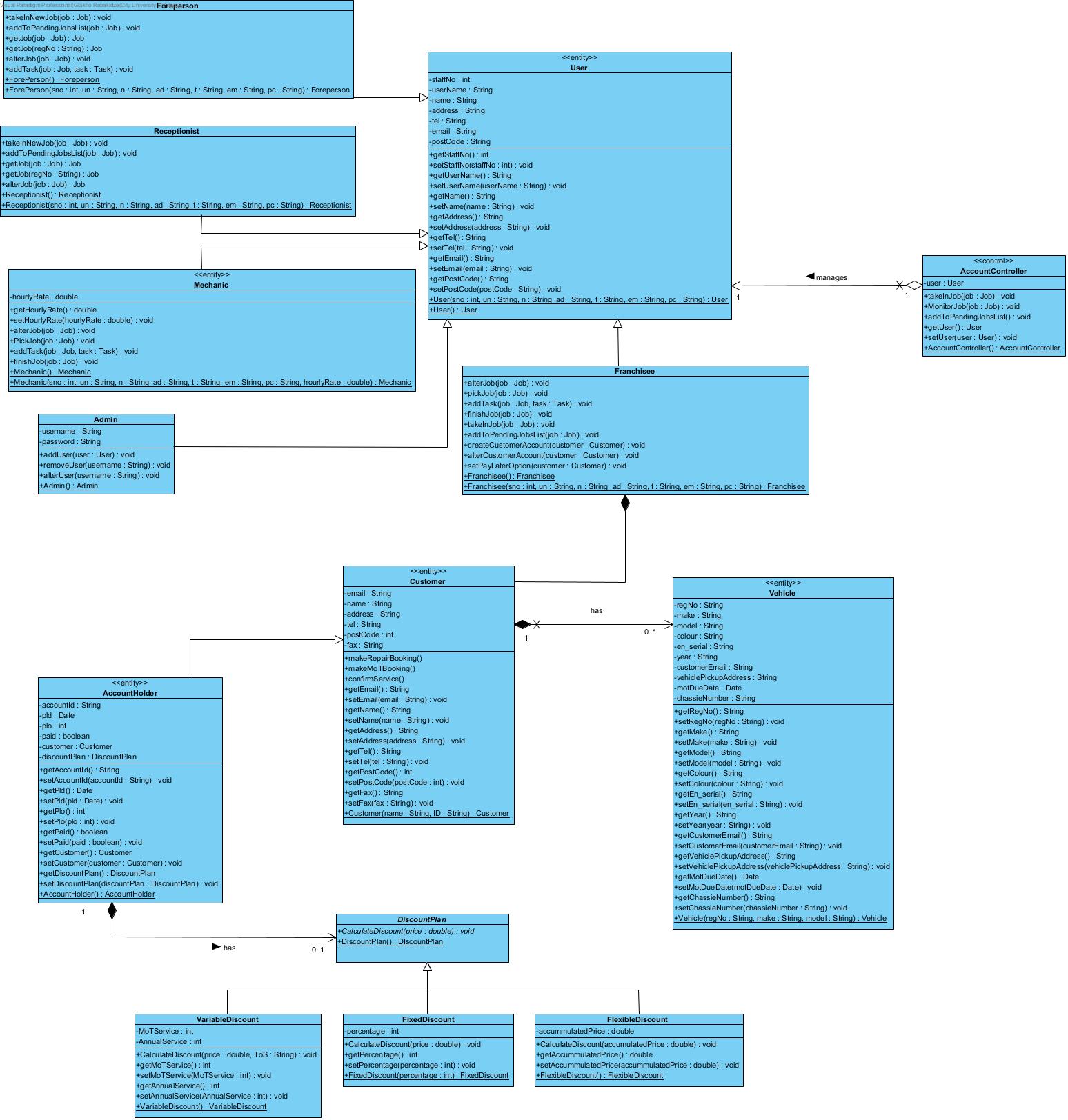
## 5.2 Design class Diagram

This is the fully refined class diagram including all of the packages in the system. Below it will also find the specific, zoomed in, views of each package.

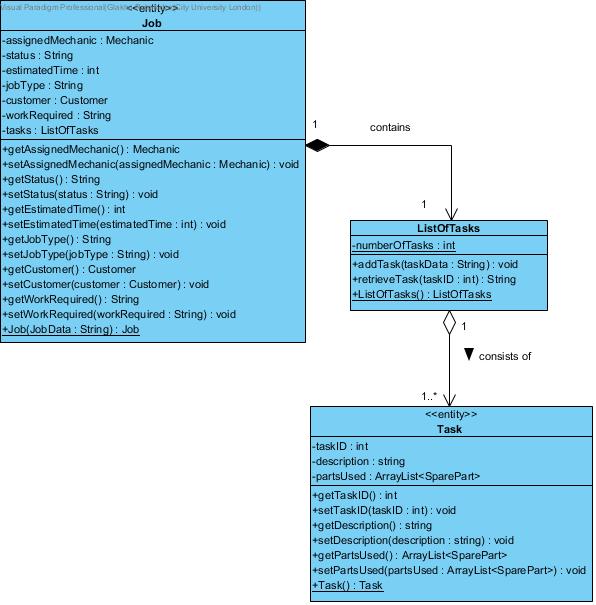


## 5.2 Packages

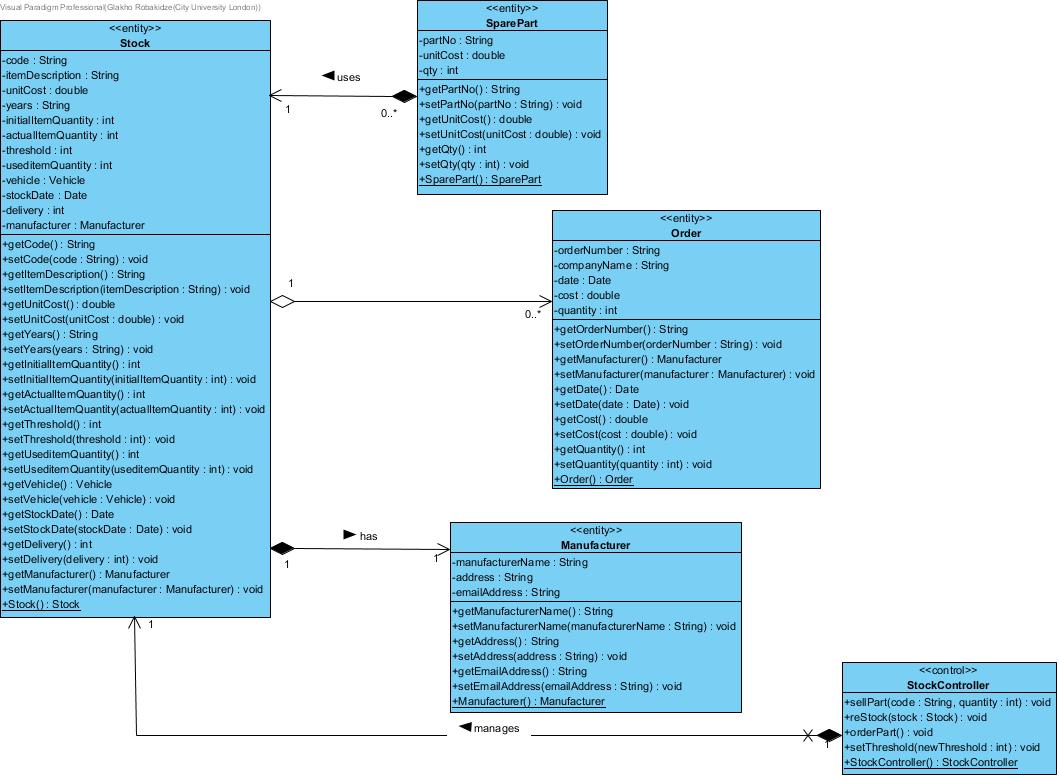
### 5.2.1 Accounts



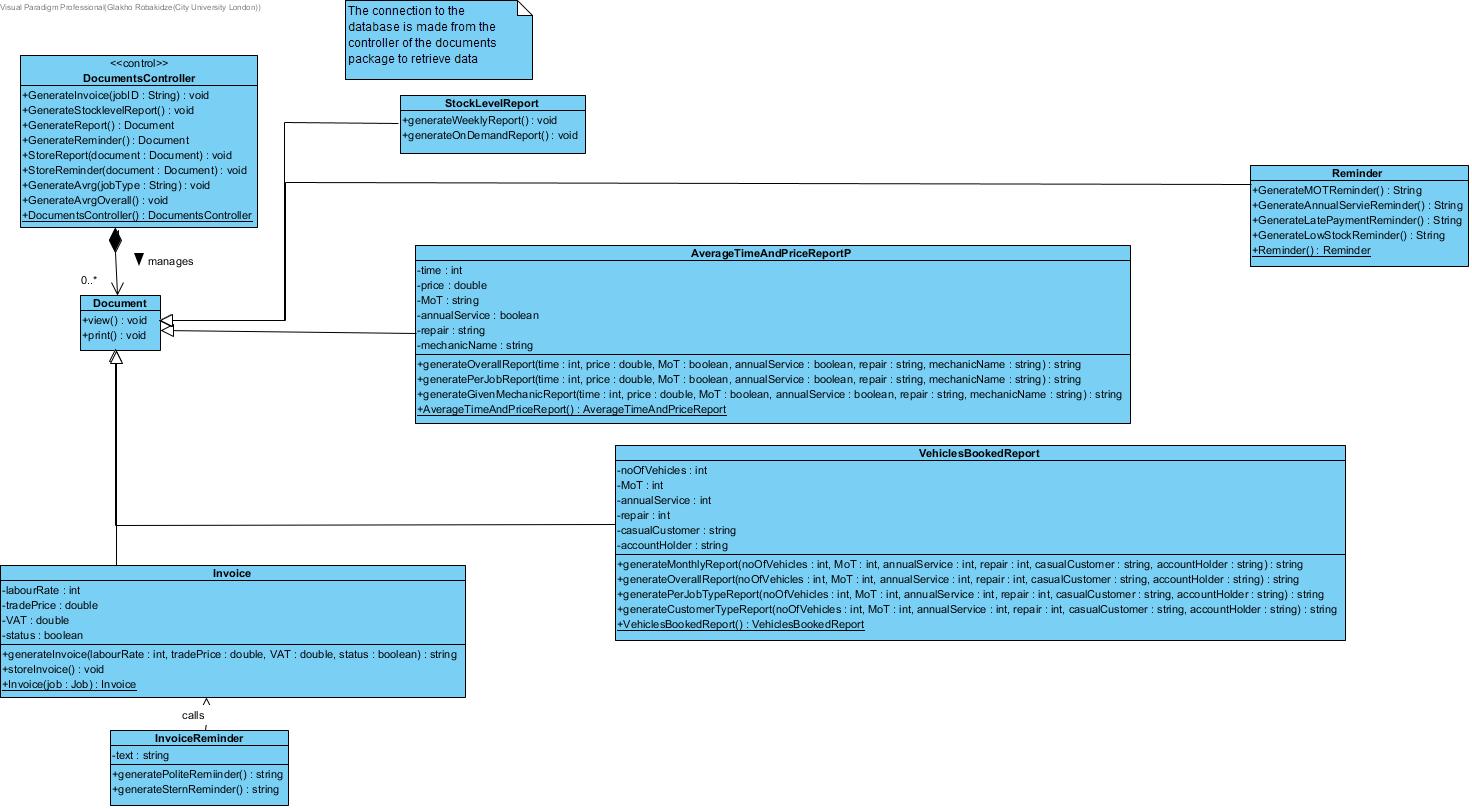
### 5.2.2 Job



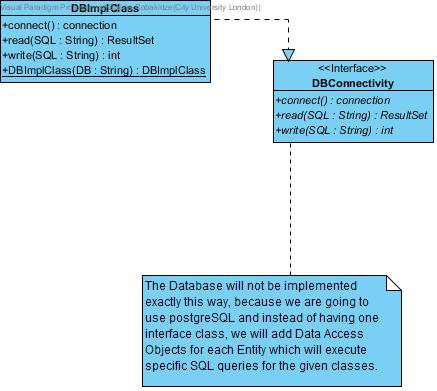
### 5.2.3 Stock Control



### 5.2.4 Documents



### 5.2.5 Database



### GUI

# 5.3 Database

### 5.3.1 ER Diagram

# 5.3.2 SQL Statements

Below are the complete set of SQL statements, including DDL and SQL DML.

### 5.3.2.1 DLL Statements

CREATE TABLE Customer (email varchar(25) NOT NULL, name varchar(24) NOT NULL, address varchar(40) NOT NULL, tel varchar(15) NOT NULL, post\_code integer(8) NOT NULL, Fax varchar(20), PRIMARY KEY (email));

CREATE TABLE Vehicle (reg\_number varchar(7) NOT NULL, make varchar(20) NOT NULL, model varchar(20) NOT NULL, eng\_serial varchar(20) NOT NULL, chassis\_number varchar(20) NOT NULL, colour varchar(20) NOT NULL, Customeremail varchar(25) NOT NULL, year varchar(4) NOT NULL, mot\_due\_date timestamp NOT NULL, "vehicle\_pick-up\_address" varchar(70), PRIMARY KEY (reg\_number));

CREATE TABLE Account\_Holder (account\_id varchar(255) NOT NULL, Customeremail varchar(25) NOT NULL, pay\_later\_date date, pay\_later\_option integer(10), paid blob NOT NULL, PRIMARY KEY (account\_id), FOREIGN KEY(Customeremail) REFERENCES Customer(email));

CREATE TABLE Job (job\_no INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, date\_booked\_in date NOT NULL, Customeremail varchar(25) NOT NULL, work\_required varchar(255) NOT NULL, job\_status varchar(30) NOT NULL, job\_type varchar(40) NOT NULL, MechanicStaffstaff\_no integer(10) NOT NULL, FOREIGN KEY(Customeremail) REFERENCES Customer(email), FOREIGN KEY(MechanicStaffstaff\_no) REFERENCES Mechanic(Staffstaff\_no));

CREATE TABLE Task (Jobjob\_no integer(10) NOT NULL, task\_duration timestamp NOT NULL, work\_carried\_out varchar(255) NOT NULL, estimated\_time timestamp NOT NULL, FOREIGN KEY(Jobjob\_no) REFERENCES Job(job\_no));

CREATE TABLE Job\_spare\_part (part\_no varchar(20) NOT NULL, quantity integer(10) NOT NULL, "Jobjob no" integer(10) NOT NULL, StockCode varchar(20) NOT NULL, PRIMARY KEY (part\_no), FOREIGN KEY("Jobjob no") REFERENCES Job(job\_no), FOREIGN KEY(StockCode) REFERENCES Stock(code));

CREATE TABLE Stock (code varchar(20) NOT NULL, item\_description varchar(50) NOT NULL, unit\_cost double(10) NOT NULL, years varchar(10) NOT NULL, initial\_item\_quantity integer(10) NOT NULL, actual\_item\_quantity integer(10) NOT NULL, low\_quantity\_threshold integer(10) NOT NULL, used\_item\_quantity integer(10) NOT NULL, vehicle\_type varchar(40) NOT NULL, stock\_date date NOT NULL, delivery integer(10) NOT NULL, Manufacturermanufacturer\_name varchar(20) NOT NULL, PRIMARY KEY (code), FOREIGN KEY(Manufacturermanufacturer\_name) REFERENCES Manufacturer(manufacturer\_name));

CREATE TABLE Staff (staff\_no INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, Loginusername varchar(255) NOT NULL, name varchar(70) NOT NULL, address varchar(100) NOT NULL, tel varchar(15) NOT NULL, email varchar(50) NOT NULL, post\_code varchar(10) NOT NULL, FOREIGN KEY(Loginusername) REFERENCES Login(username));

CREATE TABLE Admin (Staffstaff\_no integer(10) NOT NULL, FOREIGN KEY(Staffstaff\_no) REFERENCES Staff(staff\_no));

CREATE TABLE Mechanic (Staffstaff\_no integer(10) NOT NULL, hourly\_rate integer(10) NOT NULL, PRIMARY KEY (Staffstaff\_no), FOREIGN KEY(Staffstaff\_no) REFERENCES Staff(staff\_no));

CREATE TABLE "Order" (order\_no varchar(255) NOT NULL, company\_name varchar(255) NOT NULL, order\_date varchar(255) NOT NULL, StockCode varchar(20) NOT NULL, order\_cost integer(10) NOT NULL, order\_quantity integer(10) NOT NULL, PRIMARY KEY (order\_no), FOREIGN KEY(StockCode) REFERENCES Stock(code));

CREATE TABLE Franchisee (Staffstaff\_no integer(10) NOT NULL, FOREIGN KEY(Staffstaff\_no) REFERENCES Staff(staff\_no));

CREATE TABLE Foreperson (Staffstaff\_no integer(10) NOT NULL, FOREIGN KEY(Staffstaff\_no) REFERENCES Staff(staff\_no));

CREATE TABLE Login (password varchar(255) NOT NULL, username varchar(255) NOT NULL, PRIMARY KEY (username));

CREATE TABLE Receptionist (Staffstaff\_no integer(10) NOT NULL, FOREIGN KEY(Staffstaff\_no) REFERENCES Staff(staff\_no));

CREATE TABLE Manufacturer (manufacturer\_name varchar(20) NOT NULL, address varchar(100) NOT NULL, email\_address integer(10) NOT NULL, PRIMARY KEY (manufacturer\_name));

CREATE TABLE Invoice (invoice\_no INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, grand\_total integer(10) NOT NULL, "Jobjob no" integer(10) NOT NULL, "date" date NOT NULL, FOREIGN KEY("Jobjob no") REFERENCES Job(job\_no));

CREATE TABLE Discount\_plan (type integer(10) NOT NULL, percentage integer(10) NOT NULL, Account\_Holderaccount\_id varchar(255) NOT NULL, PRIMARY KEY (Account\_Holderaccount\_id), FOREIGN KEY(Account\_Holderaccount\_id) REFERENCES Account\_Holder(account\_id));

CREATE TABLE Flexible\_discount (paid\_back blob NOT NULL, mot\_percentage integer(10), special\_service\_percentage integer(10), parts\_order\_percentage integer(10), Discount\_planAccount\_Holderaccount\_id varchar(255) NOT NULL, FOREIGN KEY(Discount\_planAccount\_Holderaccount\_id) REFERENCES Discount\_plan(Account\_Holderaccount\_id));

CREATE TABLE Fixed\_discount (Discount\_planAccount\_Holderaccount\_id varchar(255) NOT NULL, FOREIGN KEY(Discount\_planAccount\_Holderaccount\_id) REFERENCES Discount\_plan(Account\_Holderaccount\_id));

CREATE TABLE Variable\_discount (discount\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, Discount\_planAccount\_Holderaccount\_id varchar(255) NOT NULL, FOREIGN KEY(Discount\_planAccount\_Holderaccount\_id) REFERENCES Discount\_plan(Account\_Holderaccount\_id));

CREATE TABLE Payment (payment\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, Invoiceinvoice\_no integer(10) NOT NULL, method integer(10) NOT NULL, Amount integer(10) NOT NULL, FOREIGN KEY(Invoiceinvoice\_no) REFERENCES Invoice(invoice\_no));

### 5.3.2.2 DML

2 Sets of Delete, Insert, Select and Update:

* SELECT \* FROM Customer;
* SELECT \* FROM Stock;
* UPDATE Customer SET tel='072132193812' WHERE email='example@gmail.com';
* UPDATE Stock SET used\_item\_quantity=used\_item\_quantity+1 WHERE Code='X66745877';
* DELETE FROM Customer WHERE email='example@gmail.com';
* DELETE FROM Stock WHERE Code='X66745877';
* INSERT INTO CUSTOMER

(email,name,address,tel,post\_code,fax)

VALUES('example@gmail.com','Boris Johnson','Twot Street 4th','043295231232','SW10 2ra','fax something')

* INSERT INTO STOCK(item\_description,Code,Manufacturermanufacturer\_name,vehicle\_type,years,unit\_cost,initial\_item\_quantity,used\_item\_quantity,delivery,actual\_item\_quantity,low\_quantity\_threshold,stock\_date)

VALUES('Grommet','X66745877','Fjord','Krapa','2011-2015',0.90,34,2,0,32,10,'20180626')

### 5.3.2.3 DML for 2 Reports

### 5.3.2.3.1 Stock Level Report

DML statements to produce the Stock Level Report:

### SELECT STOCK.\*,

### (STOCK.initial\_item\_quantity\*STOCK.unit\_cost) AS initial\_cost,

### (STOCK.actual\_item\_quantity\*STOCK.unit\_cost) AS actual\_cost ,

### SUM(STOCK.initial\_item\_quantity\*STOCK.unit\_cost) AS initial\_total,

### SUM(STOCK.actual\_item\_quantity\* STOCK.unit\_cost) AS actual\_total

### ,manufacturer\_name

### FROM STOCK INNER JOIN MANUFACTURER ON STOCK.Manufacturermanufacturer\_name=Manufacturer.manufacturer\_name;

### 5.3.2.3.2 Average Time and Price Report

*Note: For any given queries, the values conditional MECHANIC.Staffstaff\_no="12345" and JOB.job\_type="mot" should be changed depending on what mechanic and/or job type you want. If you want both filters, you just put both conditionals inside the statement conditional.*

*Calculates average labor* cost, overall per job type.

SELECT AVG(MECHANIC.hourly\_rate\*avrgtime) AS avrglabour,avrgtime FROM (SELECT AVG(TASK.task\_duration) AS avrgtime FROM TASK,JOB WHERE JOB.job\_no=TASK.Jobjob\_no ORDER BY JOB.job\_type), MECHANIC,JOB,TASK WHERE MECHANIC.Staffstaff\_no=JOB.MechanicStaffstaff\_no=TASK.Jobjob\_no ORDER BY JOB.job\_type

*Calculates average labor cost, per mechanic grouped by job type.*

SELECT AVG(MECHANIC.hourly\_rate\*avrgtime) AS avrglabour,avrgtime FROM (SELECT AVG(TASK.task\_duration) AS avrgtime FROM TASK,JOB WHERE JOB.job\_no=TASK.Jobjob\_no ORDER BY JOB.job\_type), MECHANIC,JOB,TASK WHERE MECHANIC.Staffstaff\_no=JOB.MechanicStaffstaff\_no=TASK.Jobjob\_no AND MECHANIC.Staffstaff\_no="12345" ORDER BY JOB.job\_type

### *Calculates average labor cost, per specific job.*

### SELECT AVG(MECHANIC.hourly\_rate\*avrgtime) AS avrglabour,avrgtime FROM (SELECT AVG(TASK.task\_duration) AS avrgtime FROM TASK,JOB WHERE JOB.job\_no=TASK.Jobjob\_no ORDER BY JOB.job\_type), MECHANIC,JOB,TASK WHERE MECHANIC.Staffstaff\_no=JOB.MechanicStaffstaff\_no=TASK.Jobjob\_no AND JOB.job\_type="mot" ORDER BY JOB.job\_type

Spare Parts calculations(uses views)

### CREATE VIEW spare\_parts\_price AS

### SELECT JOB\_SPARE\_PART.quantity\*STOCK.unit\_cost AS sparepart\_cost,JOB\_SPARE\_PART.Tasktask\_id FROM JOB\_SPARE\_PART,STOCK,TASK WHERE STOCK.code=JOB\_SPARE\_PART.Stockcode

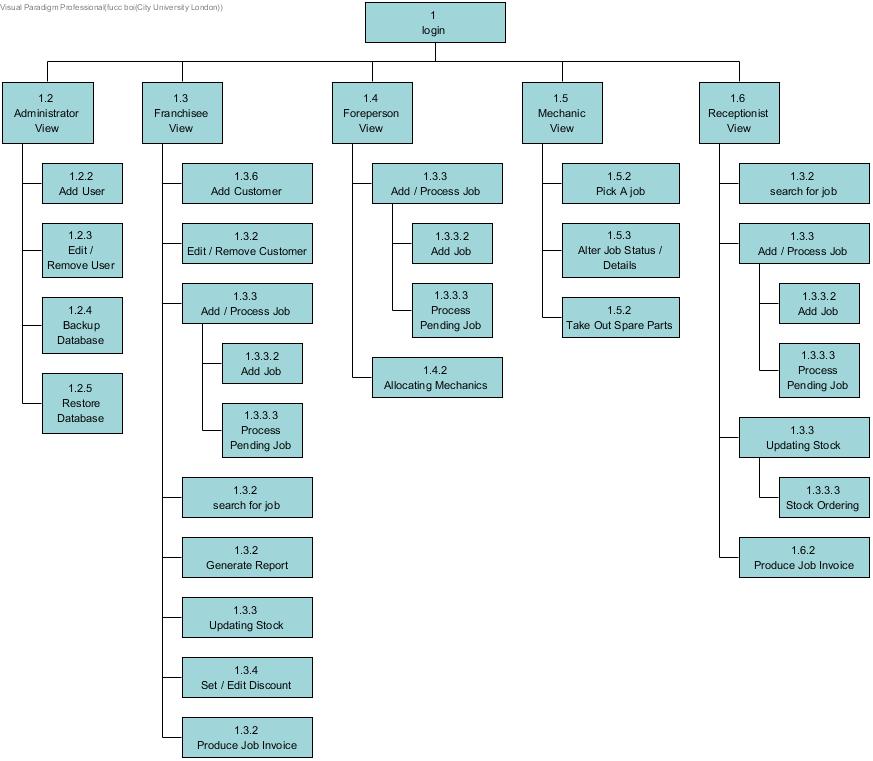
### CREATE VIEW task\_price AS

### SELECT AVG(spare\_parts\_price.sparepart\_cost) AS avg\_task\_cost,Task.Jobjob\_no as job\_id FROM spare\_parts, TASK,JOB where TASK.task\_id = JOB\_SPARE\_PART.Tasktask\_id

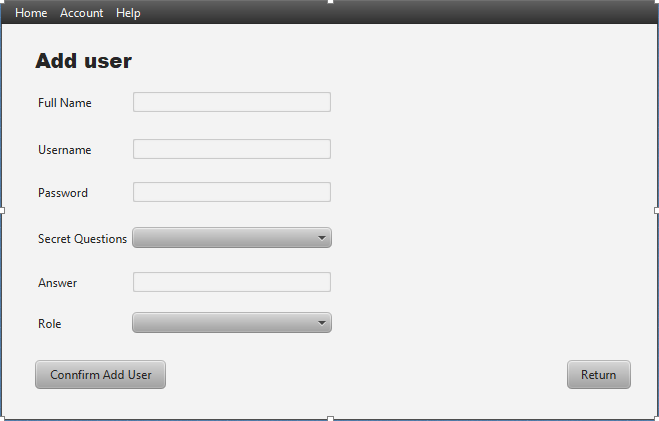
### SELECT AVG(task\_price.avg\_task\_cost),JOB.job\_type FROM task\_price,JOB WHERE JOB.job\_no=task\_price.job\_id ORDER BY JOB.job\_type

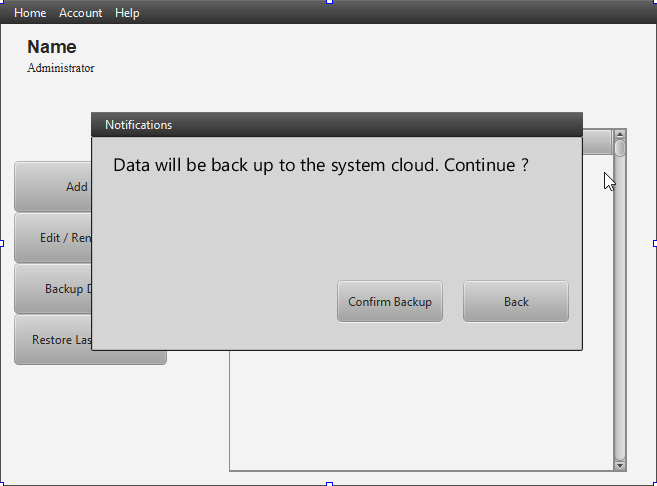
### SELECT AVRG(TASK.task\_duration) AS avrgtime FROM TASK,JOB WHERE JOB.job\_no=TASK.Jobjob\_no ORDER BY JOB.job\_type

### 5.4 GUI Diagram and Navigation

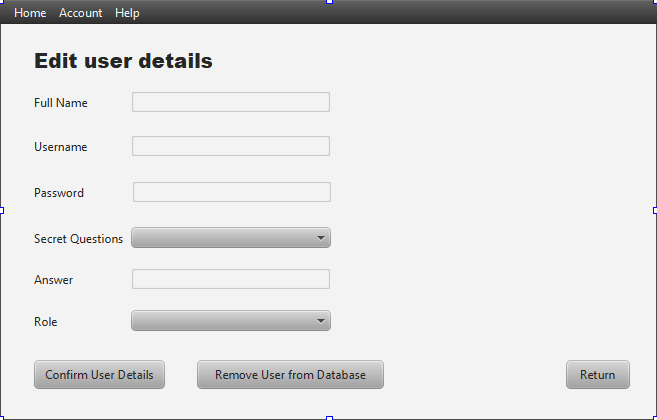
This section shows the GUI diagram, and the navigations though the different screens.

Administrator View - Add User

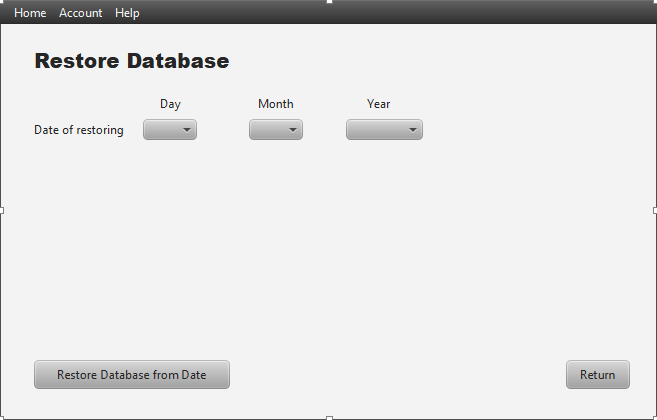


Administrator View - Backup Database

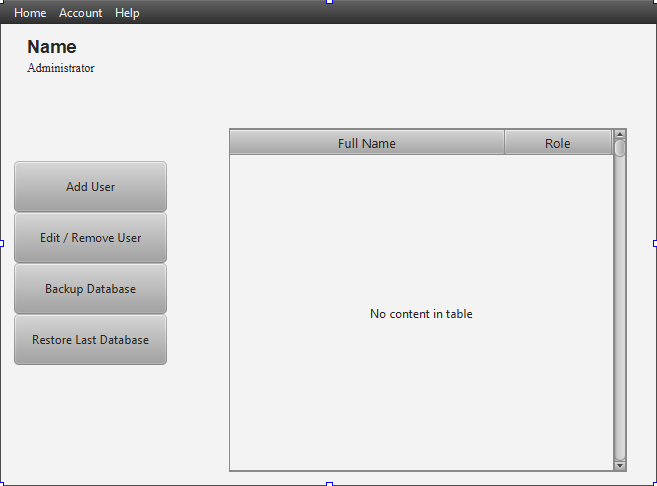
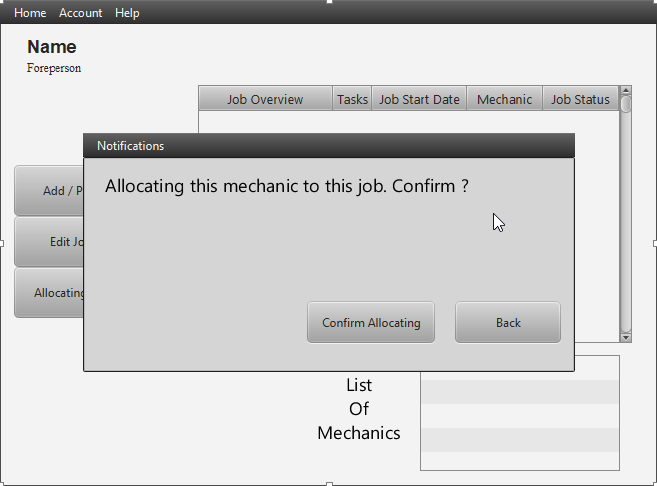
Administrator View - Edit / Remove User



Administrator View - Restore Database

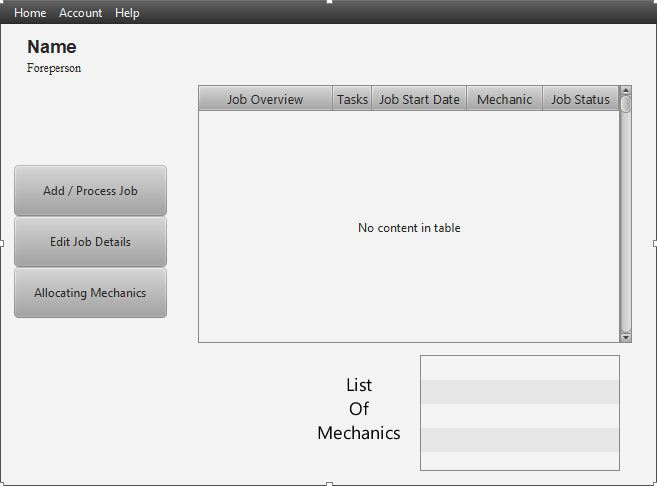


Administrator View

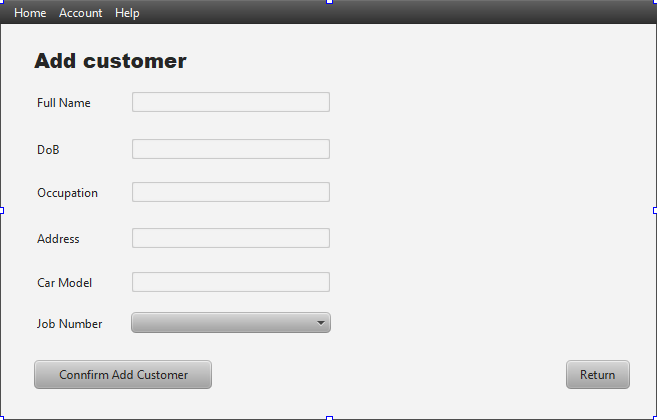


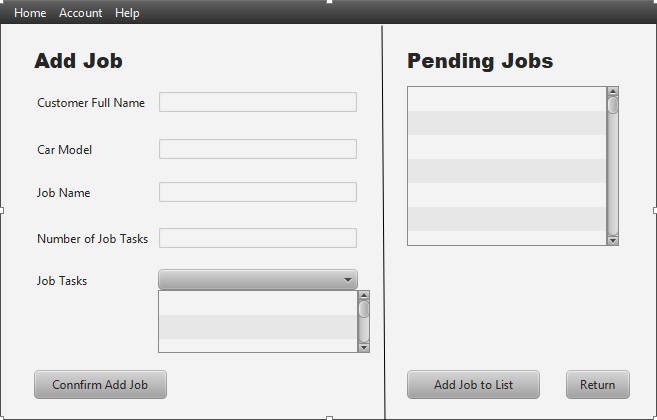
Foreperson View - Allocating Mechanics

Foreperson View

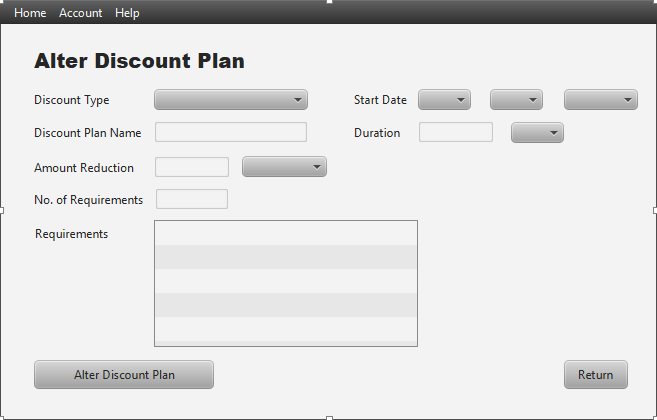


Franchisee View - Add Customer



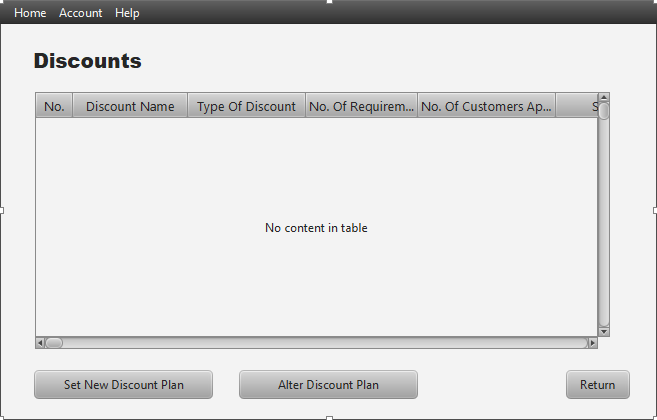


Franchisee View - Add / Process Pending Jobs

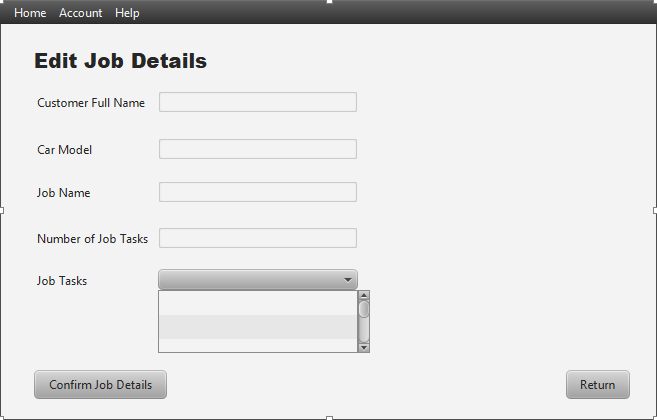


Franchisee View - Alter Discount Plan

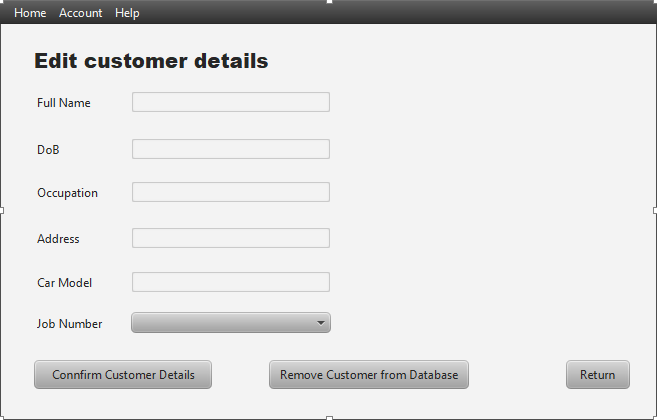
Franchisee View - Discount



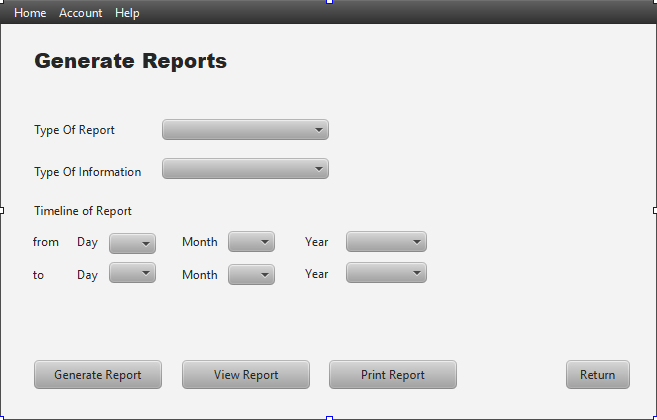
Franchisee View - Edit Job Details

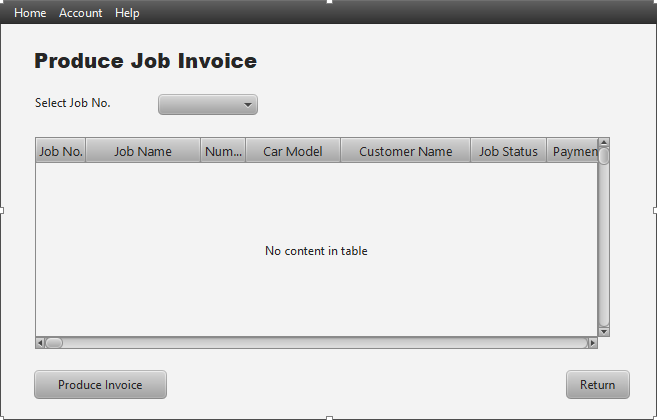


Franchisee View - Edit / Remove Customer

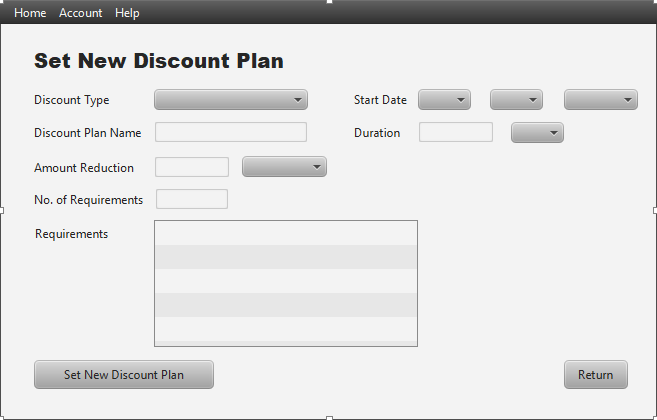


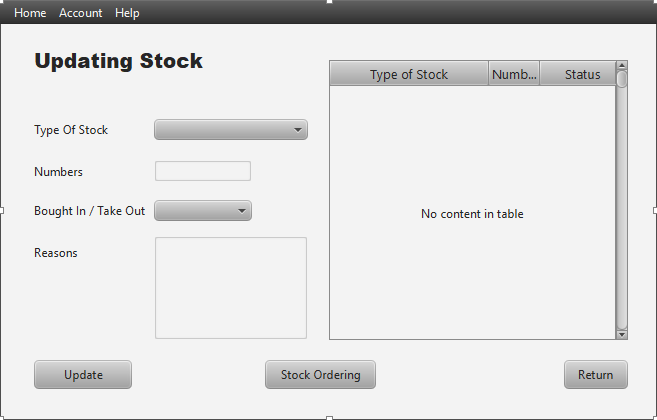
Franchisee View - Generate Reports



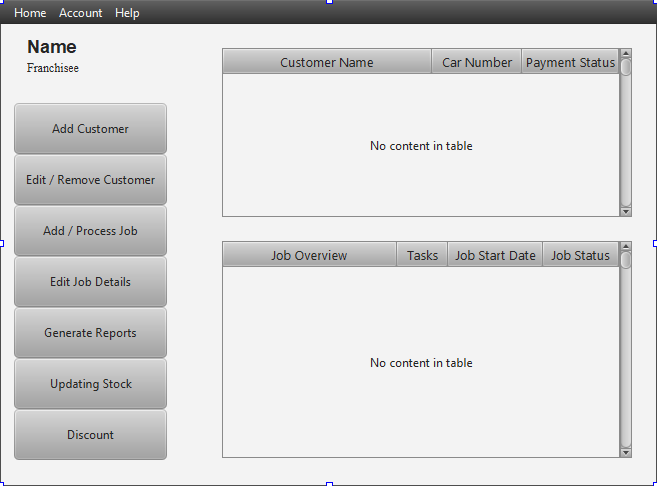
Franchisee View - Produce Job Invoice

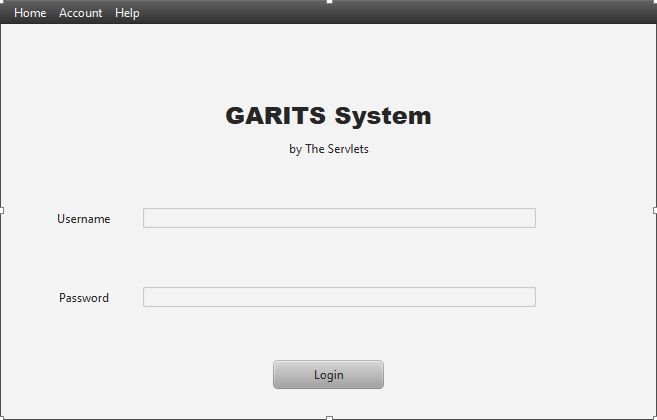
Franchisee View - Set New Discount Plan



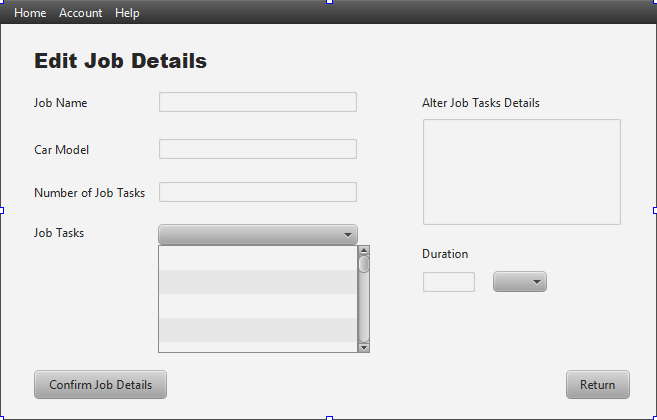
Franchisee View - Updating Stock

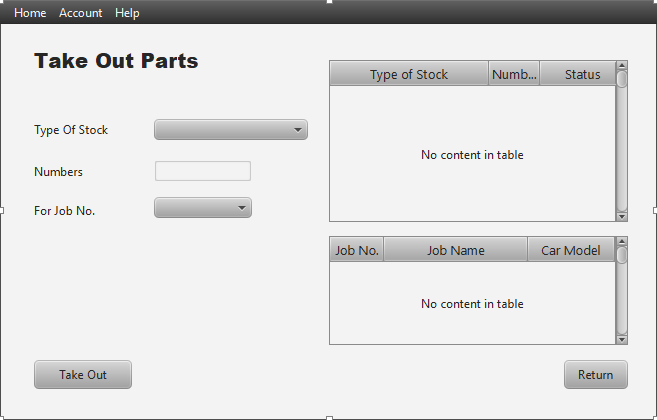
Franchisee View



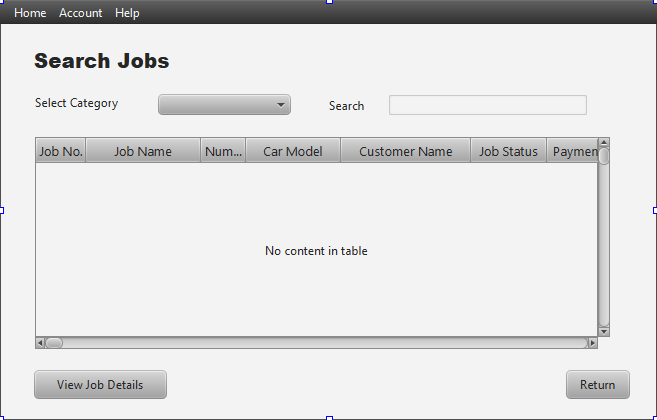
Log In

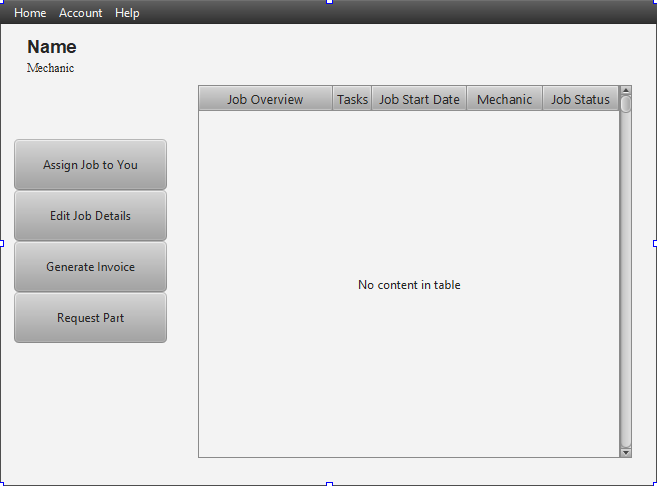
Mechanic View - Alter Job Details



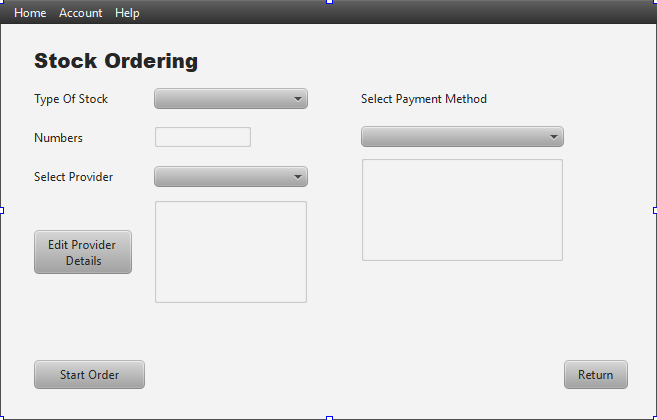
Mechanic View - Take Out Parts

Receptionist View - Search Job



Mechanic View

Receptionist View - Stock Ordering



Receptionist View

