Angel Softworks

Bloomsburys’s Automated Process Execution Recording System

Summary

This is a Software Requirements Document by Angel Softworks for the Bloomsbury’s automated execution recording system (BAPERS) software product.

Authors

Ahmed Abukar, Karan Dixit, Cristian Ghita, Kane Dharan, David Madaleno, Aimal Hederzada

Distribution

The Project manager and the development team at the Centre for Angel Softworks at City University London.

Angel Softworks, City University,
London College Building, 10 Northampton Square,
London EC1V 0HB Tel: +44 20 7123 4567,
Fax: +44 20 7123 89012

Contents

Table of Contents

Preface

1.1 Purpose and scope of the Document

This is a software requirements document for the Bloomsbury's automated process execution recording system (BAPERS) which covers the functional, behavioural and design elements of the system. The purpose of this document is to showcase the deliverables and sources we have sited throughout the project, highlighting the individual effort produced by each team member from group 12. The document explains what the system shall do but not how the functionality can be implemented.

The document will consist of the requirements specification and system design, various models are used to convey in more detail to system developers what the expected behaviour of the system is. A completed Use Case diagram, an indexed list of the Use Cases alongside its appropriate priority and 10 of the most important Use Case specifications will be included in the document. As well as this, a description of the current system and what the new system will do to improve it will be included. The system design aspect will include a copy of our completed Design Class diagram showcasing boundary, entity and control classes as well as associations, cardinalities, attributes & methods. Finally, Detailed GUI designs and GUI mappings will also be included in the document along with our entity-relationship diagram and relational database schema.

The acceptance criteria for this group project is that it addresses every one of the issues indicated by the customer, that it has been bug-tried to a satisfactory level guaranteeing there are no bugs or security vulnerabilities, and so on. A detailed analysis of this will be included later in this document.

During this project we may face some constraints, one of which may include timing, considering the fact that we are also studying another 2 modules balancing all of the work load may become difficult with deadlines approaching at similar times. Nevertheless, we believe with good planning and organisation from the entire team we will be able to mitigate this constraint effectively. One other possible constraint that we may face is individual ability, some of us are stronger in some aspects of the work than others therefore it is down to us to assign tasks while taking the strengths and weaknesses of individuals into consideration.

1.2 Intended audience

Mr. Lancaster and all stakeholders of Bloomsbury's Image Processing Laboratory.

1.3 Introduction

Bloombury's Image Processing Laboratory (BIPL) is a photographic laboratory which handles the work of professional photographers using their BAPERS system to process customer jobs effectively. Angel Softworks recognises that their current system is becoming outdated and requires some significant changes in order to improve the system and allow BIPL to successfully continue running their business. This systems requirement document will outline the BAPERS system and the design to implementation. The following will be included in this document;

- ✓ Use Case diagram
- ✓ Use Case specifications
- ✓ Use Case prioritisation
- ✓ Design Class diagram
- ✓ ER diagram
- ✓ GUI designs

The systems requirement document will also list both the system requirements and the system design with the different functionalities, as well as including the several actors and interactions within the system.

Glossary

Actor	Device or human that is interacting with the system.
UML	Unified Modelling Language.
Use Case	The link between the system and the actors.
BIPL	Bloom Bloomsbury's Image Processing Laboratory
BAPERS	Bloomsbury's Automated Process Execution Recording System
Packages	The process of grouping correlated elements together, it will be visible during design and use case diagrams. It also creates a semantic boundary and helps structure system.
ERD	Entity Relationship Diagram.
GUI	Graphical User Interface.
SQL	Standard Query Language.
BAP-ACCT	accept job at reception; this facility is where the users can request the jobs that they want.
BAP-PROC	process job through laboratory; this facility will process all the jobs that have been requested by the user
BAP-ADMN	Administering the System; this facility will allow you to control/mange the entire system.

Version Control

The document will consist of a detailed log the team will be keeping of all the changes that will be made to the document below.

Major versions consist of significant changes to the document such as adding or removing whole sections of the document, adding any of the diagrams we have created, source code, etc. The changes made will be signified by moving up a version number i.e. version 3.2 to 4.2. All members of the team must agree to this before making the change.

Minor versions consist of simple changes to the document such as modifying grammar/ spelling errors, changing the structure of the document, etc. these changes will be identified by changing the decimal number after the version number i.e. 1.2 to 1.3.

Version	Author & role	Changes made	Date
V 1.0	Ahmed Abukar – Deputy Project Manager & Systems analyst	Initial draft, setting up the cover page & Headers.	11/02/2018
V 1.1	Ahmed Abukar – Deputy Project Manager & Systems analyst	Added table of contents.	15/02/2018
V 1.2	Ahmed Abukar – Deputy Project Manager & Systems analyst	New sections and headers added to document.	20/02/2018
V 2.2	Karan Dixit – Project Manager & Designer	Added Diagrams	02/03/2018
V 3.2	Karan Dixit - Project Manager & Designer	Added 10 key Use Case Specification	03/03/2018
V 4.2	Ahmed Abukar – Deputy Project Manager & Systems analyst	Added use case indexing	04/03/2018
V 5.2	Karan Dixit - Project Manager & Designer	Added Mapping	04/03/2018

Requirements Specification

2.1 BIPL, current system

Bloomsbury's image processing laboratory (BIPL) consists of BAPERS in their photographic laboratory which handles the work of professional photographers and works within tight deadlines to deliver customers without compromising on quality. BAPERS offers its customers a variety of different jobs to choose from as well as allowing customers to make special requests for their desired job. The clients of BIPL can request for urgent jobs to be completed usually within 6 hours at a higher price or simply a standard job which will be completed within 24 hours. To carry out a single job, the client is required to provide the image or images which they require processing. The BIPL staff will carry out the adequate tasks required for the job and once completed, the client is required to pay in full by either cash or card.

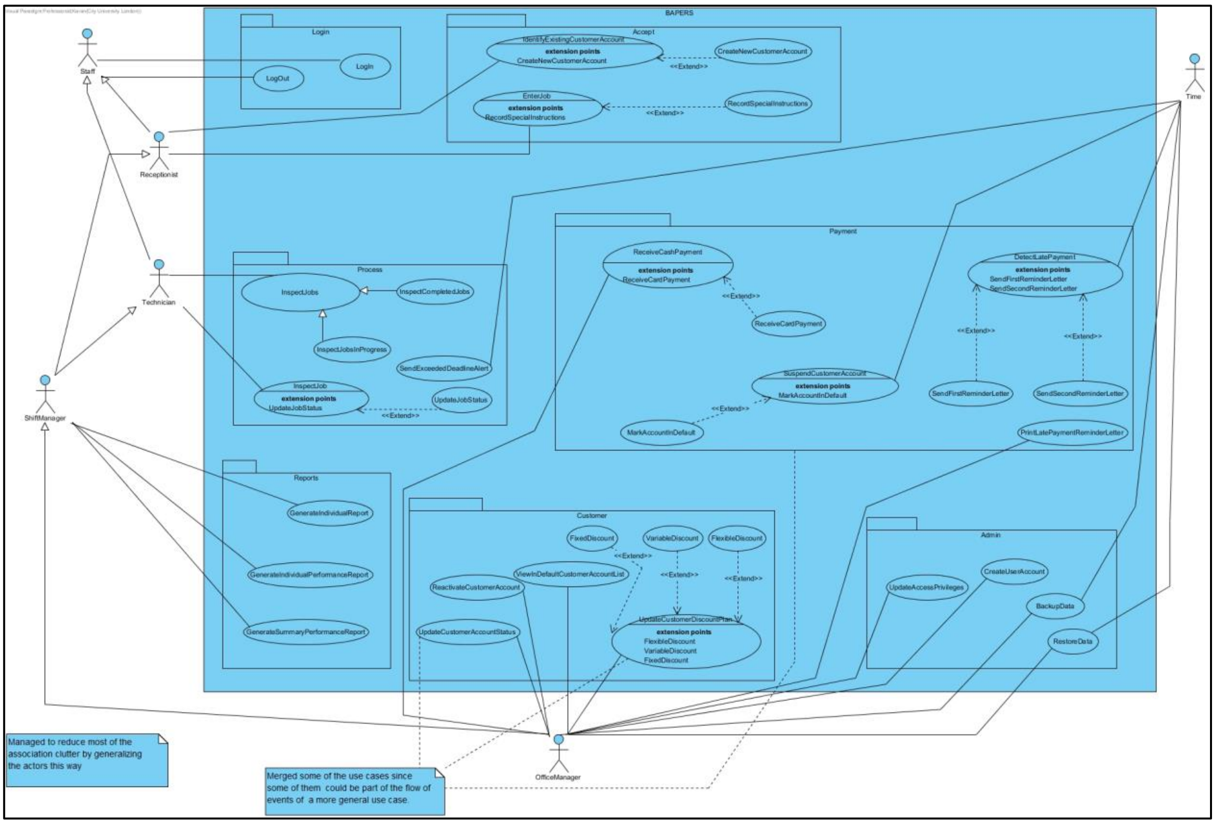
At present, BAPERS is a manual hardback system which comes with its disadvantages. For instance, the fact that the jobs need to be hand written makes it less efficient when it comes to processing data as well as making it more prone to human errors such as losing data. The current system enables its technicians to perform around 30 different tasks depending on the job at hand, these are carried out in different labs and is manually recorded on a job sheet. The customers jobs are then stored on a shelf slot with a unique identifier applied on each of the tasks, where it can then be collected for further processing or dispatched to customers.

Discounts are available and can be applied but only to high value customers. High value customers are determined by the number of tasks they request and how consistently they pay for their jobs on time. There are 3 types of discounts that BIPL offer, fixed discount; where the same percentage of discount is given to the valued customer for each job. Variable discount; which is similar to fixed discount where a fixed percentage is supplied but may vary depending on the task. Flexible discount; is where the customer is set a percentage discount depending on the value of jobs they have accumulated within that month. All discounts are allocated by the office manager.

Once a client's job has been completed, an invoice is created and is sent out to the clients address and will contain all different tasks that have been carried out by the technicians on their photos, along with the payment which is due. The system will flag up any overdue payments by the customers automatically, as the office manager is required to send out reminder letters to those customers. This process isn't the most efficient way as it may take time for a customer response. A record of each customer invoice is filed for evidence.

It is expected that newer and more efficient processes will be implemented by Angel Softworks to improve the current system. The new system aims to upgrade and support new technology and continue sustaining the current high-quality standards. Upon making the relevant changes to the system, BIPL will then be expected to carry out jobs on behalf of its customers and every job accepted will be chargeable to a customer account. BIPL will enable the employee on the reception desk to then enter the job number on a computer terminal, label the material provided by the customer and moved down to the laboratory. The laboratory staff will proceed with processing the materials while recording any progress made on a computer terminal, while avoiding any potential confusion between customer's jobs as this is vital. Queues of work may build up at work stations, however flexible scheduling will be implemented which will allow priority to be given to *urgent* jobs over the regular jobs. The new BIPL system will also provide functionality for inspecting the list of active/pending jobs as well as those already completed.

Full use case diagram



Managed to reduce most of the association clutter by generalizing the actors this way

Merged some of the use cases since some of them could be part of the flow of events of a more general use case.

Use case specification for the 10 key use cases

Use Case ID: 8	**Use Case:** ReceiveCashPayment
Brief description: As soon as a job or several jobs has been completed. The customer will make a single payment for one of more jobs. However, it's the Office Manager's responsbility to recieve the payment as the customer cannot interactive with BAPERS.	
Primary actors: OfficeManager	
Secondary actors: None	
Preconditions: Customer's job(s) has been completed	
Flow of events: 1) Actor activates the functionality for receiving payment by selecting one of the 2 options "PaymentForAllJobs" or "PaymentForParticularJob" from the job tab which displays the list of completed jobs associated with a single customer account. Extension point: PaymentForParticularJob Extension point: PaymentForSeveralJobs 2) If actor selects "PaymentForParticularJob" The system will prompt the user to select one of the jobs Actor selects single job in which customer wants to pay for at the time 3) The system places an empty payment record in the session 4) The system opens a form for entering the payment details including payment amount and type. 5) The actor types in the required details 6) The systems create a new payment record in BAPERS 7) If the system succeeds 7.1) System records the amount and payment type 8) The system places an empty payment record in the session	
Postconditions: Payment details are stored in the 'Payment' record in the session	
Alternative flow: None	

Use Case ID: 14	**Use Case:** UpdateJobStatus
Brief description: Can update status of any given job by recording completion of current task and commencement of next.	
Primary actors: Technician, Office Manager and Shift Manager	
Secondary actors: None	
Preconditions: BAPERS is operational, the primary actor has selected a task	
Flow of events: 1) The actor clicks on the “record completion” option on the task tab 2) The system records the completion of current task 3) The system prompts the actor to select the next task to commence by displaying the list of uncompleted tasks associated with the job 4) The actor selects the next task, from the list, to be commenced 5) The system will update the status of the job	
Postconditions: The system has updated the status of the job and the tasks.	
Alternative flow: None	

Use Case ID: 28	**Use Case:** UpdateCustomerAccountStatus
Brief description: The Office manager is able to upgrade or downgrade a customer’s account.	
Primary actors: OfficeManager	
Secondary actors: None	
Preconditions: BAPERS is operation, Office Manager has logged in	
Flow of events: 1) The actor selects the Customer Account in which they want to upgrade 2) The actor clicks one of the option “Upgrade” or “Downgrade” which are presented in the customer account tab. 3) If actor selects “Upgrade” System upgrades the customer account to Valued Customer Else System downgrades the account from Valued Customer to Standard Customer	
Postconditions: The system has updated the status of the Customer account	
Alternative flow: CannotUpdateCustomerAccountStatus	

Use Case ID: 30	**Use Case:** DetectLatePayment
Brief description: The system is able to detect if a payment has been beyond the deadline	
Primary actors: Time	
Secondary actors: OfficeManager	
Preconditions: Payment deadline exceed and not payment has been received. Office manager logs in.	
Flow of events: 1) For every 15 minutes and the Office manager doesn't acknowledge the message 1.1) The system generates a late payment pop-up window on the Office manager's screen 2) Office manager acknowledges the receipt by clicking on the "OK" button from the pop-up window	
Postconditions: The frequent 15-minute interval pop-ups stopped appearing	
Alternative flow: None	

Use Case ID: 24	**Use Case:** BackupData
Brief description: The Office Manager has the ability to run database backups and restores on demand.	
Primary actors: Office Manager	
Secondary actors: None	
Preconditions: 1) Actor has logged in to the system. 2) There is an existing database with existing data	
Flow of events: 1) Actor activates the functionality for storing data by clicking on the "backup" button on the Office Manager's GUI. 2) The system will prompt the actor to select the backup location 3) Actor selects the file location 4) The system will create an empty folder in the designated backup location 5) The system will store all the database data into the empty folder 6) The system prints message "database storage completed"	
Postconditions: A new folder containing all the essential database data stored within the file location chosen by the actor	
Alternative flow: None	

Use Case ID: 9	**Use Case:** SendExceededDeadlineAlert
Brief description: The system should create a pop-up message, alerting the staff about missing a potential deadline.	
Primary actors: Time	
Secondary actors: ShiftManager	
Preconditions: 1) Actor (Office/Shift Manager) has logged in 2) The computed expected completion date has exceeded the set deadline	
Flow of events: 1) The system generates a visual cue that is displayed on the manager's screen which should denote the message "Alert: The expected time has exceeded the set deadline" 2) The recipient of the alert will acknowledge the alert by clicking the "OK" on the visual cue	
Postconditions: System switches the display to display that particular job which has the exceeding expected deadline.	
Alternative flow: None	

Use Case ID: 10	**Use Case:** InspectJob
Brief description: The Office Manager has the ability inspect a single job	
Primary actors: Technician	
Secondary actors: None	
Preconditions: 1) Actor has logged in to the system. 2) Actor has accessed the list of jobs	
Flow of events: 1) Actor activates the functionality for inspecting a single job by clicking on one of the jobs off the job list. 2) The system displays the page for that selected job for inspection 3) If the actor is finished with inspecting the job 3.1) The actor will click on the "Go back" button 3.2) The system displays the list of jobs page	
Postconditions: 1) The actor can now inspect the details of the selected jobs 2) The functionality of editing the details should be accessible to the actor.	
Alternative flow: None	

Use Case ID: 1	**Use Case:** Login
Brief description: The Office, Shift Manager and receptionist can log in to the system	
Primary actors: OfficeManager, Shift Manager, Receptionist	
Secondary actors: None	
Preconditions: None	
Flow of events: 1) The actor enters in their username and password 2) The system creates a login session 3) The system verifies that the actor’s input is valid or not 4) The system will determine the role of the user 5) According to the role the system identified, the system will display the menu accordingly 6) The system closes session	
Postconditions: The user has access to the main menu	
Alternative flow: InvalidInputDetails	

Use Case ID: 26	**Use Case:** UpdateAccessPrivileges
Brief description: The office manager can modify the access privileges of any existing user accounts.	
Primary actors: OfficeManager	
Secondary actors: None	
Preconditions: The actor is logged into the system	
Flow of events: 1) The actor navigates to the appropriate GUI area 2) The actor selects the relevant option 3) The actor inputs the name of the user account 4) The system searches the database for the account 5) The actor makes any necessary changes 6) The actor confirms the action 7) The changes are recorded in the database 8) A message about the outcome of the operation is displayed on screen	
Postconditions: The message can now be dismissed by selecting the appropriate option.	
Alternative flow: NoDatabaseConnection	

Use Case ID: 23	**Use Case:** CreateUserAccount
Brief description: The Office Manager creates a new user account for one of the BIPL staff members.	
Primary actors: OfficeManager	
Secondary actors: None	
Preconditions: The actor is logged into the system	
Flow of events: 1) The actor navigates to the appropriate GUI area dedicated to managing the system 2) The actor selects the option to create a new user account in the GUI 3) The actor enters the necessary details for a new account 4) The actor confirms the new credentials 5) The new user account details are recorded in the database 6) A message is displayed, showing the outcome of the operation	
Postconditions: The new user account is now available for accessing by the staff member.	
Alternative flow: NoDatabaseConnection	

Alternative flow: InvalidInputDetails
ID: 1.1
Brief description: The actor types in invalid username or password or both
Primary actors: Office, Shift manager, receptionist
Secondary actors: None
Preconditions: The actor typed in invalid credentials
Alternative flow: The alternative flow starts after step 3. Of the normal flow 1) A message is displayed indicating that the details are invalid 2) The actor can click the appropriate button to dismiss the message.
Postconditions: None

Alternative flow: NoDatabaseConnection
ID: 23.1 26.1
Brief description: The system failed to access the database.
Primary actors: OfficeManager
Secondary actors: None
Preconditions: The OfficeManager tried to create a new user account.
Alternative flow: The alternative flow starts after step 5. of the main flow. 1) A message is displayed to reflect that the operation has failed 2) The actor can click the appropriate button to dismiss the message
Postconditions: None

Alternative flow: PrintingError
ID:
Brief description: An error has been encountered while trying to print reminder letter(s).
Primary actors: OfficeManager
Secondary actors: None
Preconditions: The office manager has selected the option to print late payment reminder letter from the GUI
Flow of events: The alternative flow starts after step 4. of the main flow. A message is displayed indicating that an error has occurred while trying to print the document
Postconditions: None

Alternative flow: NoReminderLetters
ID:
Brief description: There are no late payment reminder letters currently pending.
Primary actors: OfficeManager
Secondary actors: None
Preconditions: The office manager has selected the option to print late payment reminder letter from the GUI
Flow of events: The alternative flow starts after step 2. Of the normal flow 1) A message is displayed indicating that there are no currently pending letters 2) The actor can click the appropriate button to dismiss the message.
Postconditions: None

Alternative flow: CannotUpdateCustomerAccount
ID: 28.1
Brief description: A Valued Customer Account cannot be upgraded as it already is upgraded, or a Standard Customer Account cannot be downgraded as it already is at the lowest grade.
Primary actors: Office Manager
Secondary actors: None
Preconditions: The customer account is “standard” and selects option to downgrade or the customer account is “valued” and selects the option to upgrade.
Flow of events: 1) The alternative flow starts after step 3 of the normal flow 2) BAPERS displays a message: “Account already upgraded” or “Account already downgraded”
Postconditions: None

Indexed list of prioritized use cases

Use Case Name	Use Case ID	Priority Level	Specification provided	Justification
LogIn	1	High	Yes	Staff members of BIPL will be required to log in with their unique user ID's to access the features of BAPERS. This is important as it allows the managers to monitor the progress of their staff.
LogOut	2	High	No	Staff members are required to sign out of the system once completed their job. This will prevent anyone with unauthorised access to use the system.
IdentifyExistingCustomerAccount	3	High	No	For a job to be accepted, it needs to be assigned to a valid customer account. Not being able to identify existing customers through a search would mean that the BAPERS staff would have to manually search through the database, which can be very time consuming and prone to human error.
CreateNewCustomerAccount	4	Medium	No	The receptionist can create a new customer account for first time clients. Adding new clients to the pool of already existing clients is essential as it is vital that new customers have their details captured efficiently and effortlessly for when they return in future.
EnterJob	5	High	No	BIPL staff will be able to enter a job requested by a customer. This is important BIPL staff who will be processing the job will know exactly what is required of them.
RecordSpecialInstruction	6	Medium	No	It is important that BIPL staff can clearly see any special requests made by the client. This will prevent the customer returning and being unsatisfied with the service. Customer satisfaction is vital for any business as this may determine whether they will return for your service or not.
ReceiveCardPayment	7	High	No	A soon as a job has been entered the user will be able to receive a card payment. it is vital that BIPL offer card payment services for their clients as not every customer prefers to use cash, some customers may prefer to use a credit card.
ReceiveCashPayment	8	High	Yes	A soon as a job has been entered the user will be able to receive a cash payment. Customers are able to pay in cash upon collection of their product.
SendExceededDeadlineAlert	9	Medium	Yes	BAPERS will automatically send out an exceeded deadline alert for customers who have outstanding payments due. This is important as it would be difficult and very time consuming to do this manually.
InspectJob	10	Medium	Yes	The technician has the ability to inspect a single job. Not having a function that allows the staff to search for the progress of a singular job would decrease the efficiently and increase the likelihood of human error as the staff would be required to search manually, thus increasing the time taken for each job
InspectCompletedJob	11	Medium	No	It is important that the technicians are able to view completed jobs as they'll be able to moderate it to see if the job has been completed to the customers needs. This will identify any human errors that may have occurred in the labs and able to fix the issue before it is dispatched to customers.
InspectJobsInProgress	12	Medium	No	BIPL technicians being able to check jobs that are in progress is crucial to see if the job being done is meeting the customers requirements.

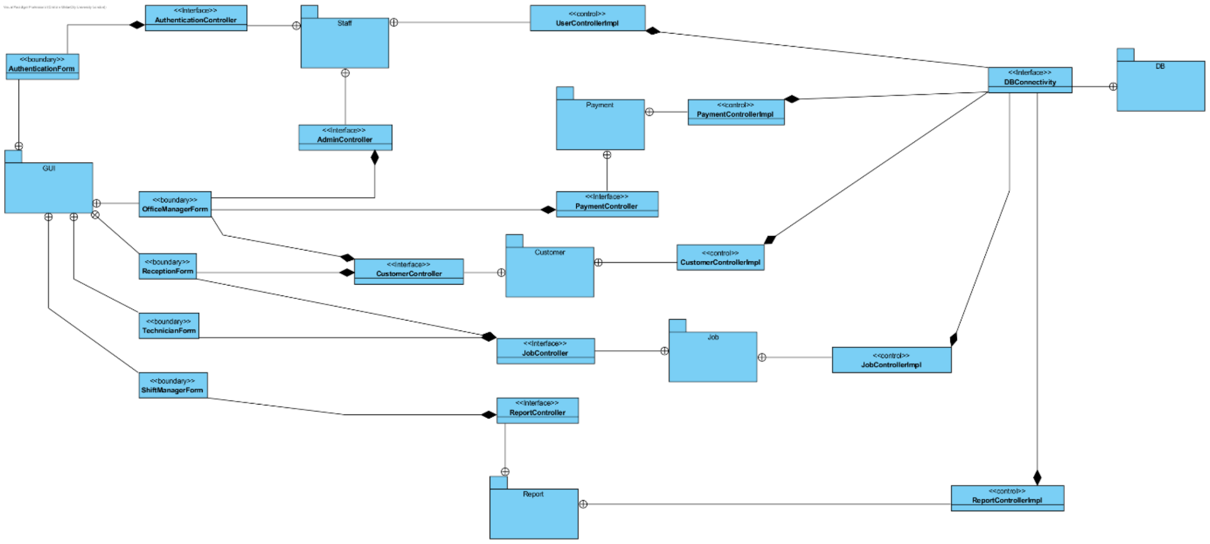
InspectJobs	13	Medium	No	BIPL staff will be capable of inspecting multiple jobs at one time making it much more efficient to inspect jobs saving a lot of time.
UpdateJobStatus	14	High	Yes	The technicians have the ability to update the job status from not completed to completed. This is crucial as BIPL staff will be able to clearly see jobs that are still outstanding and require work and ignore the ones that have already been completed.
GenerateIndividualReport	15	Low	No	The user will be able to generate an individual report on the customer. This will cover everything that the client has requested. It is important that the customer receives a copy of this, so they know exactly what is being done or if they may want to add or change anything.
GenerateIndividualPerformanceReport	16	Low	No	The user will be able to generate an individual report for the staff working on a job. This will cover how many tasks and jobs they have completed in a given day. This is important for the office manager to monitor this as staff need to work efficiently in order to meet customer deadlines.
GenerateSummaryPerformanceReport	17	Low	No	The office manager will be able to generate a summary performance report on the staff members. This will highlight the jobs completed by all BIPL staff in a given day. It is important for the office manager to stay on top of this in order to ensure production rates are in line with the customers deadlines.
SendExceededDeadlineAlert	18	Medium	No	The user will be able to send exceeded deadline alert to its clients if the job isn't completed by the estimated deadline. This is important as sometimes this occurs and its important that the customer is aware of this in order to prevent them from coming to collect their product and its not ready leaving them unsatisfied with the service.
SendFirstReminderLetter	19	Medium	No	Customers that have an outstanding balance after an extended period will receive their first reminder letter, failure to implement this would result in the customer having continued privileges without paying their balance. Also, some customers may not be aware that they have payments outstanding, so it is important to 'remind' them.
SendSecondReminderLetter	20	Medium	No	A second reminder to customers are sent after there is no response to the first one, this is important as it adds more pressure on the client to pay or some clients may have missed the first reminder letter.
PrintLatePaymentReminder	21	Medium	No	If a customer has any late payments a reminder will be sent to them so that they can make the payment. This is crucial as some customer may not be aware of the fact that they have a payment outstanding.
SuspendCustomerAccount	22	High	No	The staff of BIPL will be able to suspend a customer account if there are any late payments. This is significant as it is important that customers who have not paid up to date are unable to use the services of BAPERS until they pay their premium.
CreateUserAccount	23	Medium	Yes	The office manager will have the ability to create an account for all the new staff members. This will allow the manger to track and monitor the progress of all staff, including what job they are working on.
BackUpData	24	High	Yes	The office manager will have the ability to run backups on the data. It's vital that the manager can back up all data from the system as any loss of data

				could be detrimental for the company. The loss of client data will slow down the progress on customer jobs which would lead to customer dissatisfaction having a negative impact on the business.
RestoreData	25	High	No	The office manager will have the ability to restore the data. It is important that BIPL manager is able to restore data that has been backed up. This would be relied upon in a situation where data was loss. This could save the reputation of the business as the loss of client data could be detrimental.
Update AccessPrivileges	26	High	Yes	The office manager will be able to set up and update the access privileges of the members of staff. Failing to implement this function would mean that the office manager would not be able to set restrictions on the staff accounts, this can cause problems in the long term.
UpgradeCustomerDiscountPlan	27	Medium	No	The office manager will be able to upgrade the several discount plans. The manager will be able to look over customer accounts and jobs and able to recognise which customers are eligible for a discount, ensuring all customers are paying the correct price avoiding any confusion.
UpdateCustomerAccountStatus	28	Medium	Yes	Office manager is able to change the status of customer accounts from ‘standard’ to ‘valued’, these are the customers that are eligible for a discount, so it is important to be able to differentiate between them. Customers who are not eligible for a discount should not receive one.
ViewInDefaultCustomerAccount List	29	Low	No	The office manager will be able to view the default customer account list and will have access to it. The office manager should be aware of the customers that have failed to clear their outstanding balance, failure to do so the office manager would not know which customer accounts have which privileges.
DetectLatePayment	30	Medium	Yes	The user will be able to detect a late payment based on the time exceeded. Important that BIPL staff are identified of any late payments made by clients, this is crucial as BIPL staff need to know which customer requires a late payment reminder.
FixedDiscount	31	Medium	No	This discount allows the same percentage of discount to be deducted from the value of the job and given to a valued customer. This highlights the fact that some customers are valued by the company and want to retain them as a customer.
VariableDiscount	32	Medium	No	A percentage of discount is applied to each individual task and may vary depending on the task at hand. This indicated that the company shows loyalty to its customers ensuring a high customer satisfaction rate.
FlexibleDiscount	33	Medium	No	This discount calculates a percentage to be discounted depending on the value of the task and how many jobs they have accumulated within the calendar month. This indicates to customers that that more they use BIPL’s services the cheaper it will cost them.
MarkAccountInDefault	34	High	No	The user will be able to mark an account as default by request of the customer or if there are outstanding payments. Customers that have an outstanding balance after an extended period should be marked in default, failure to implement this would result in the customer having continued privileges without paying their balance.

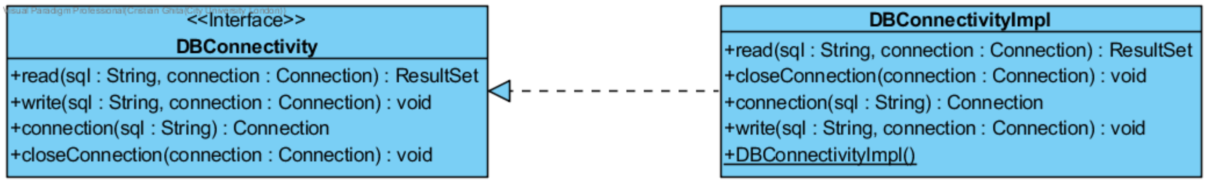
System Design

Scope of the system

Fully refined/correct design class diagram

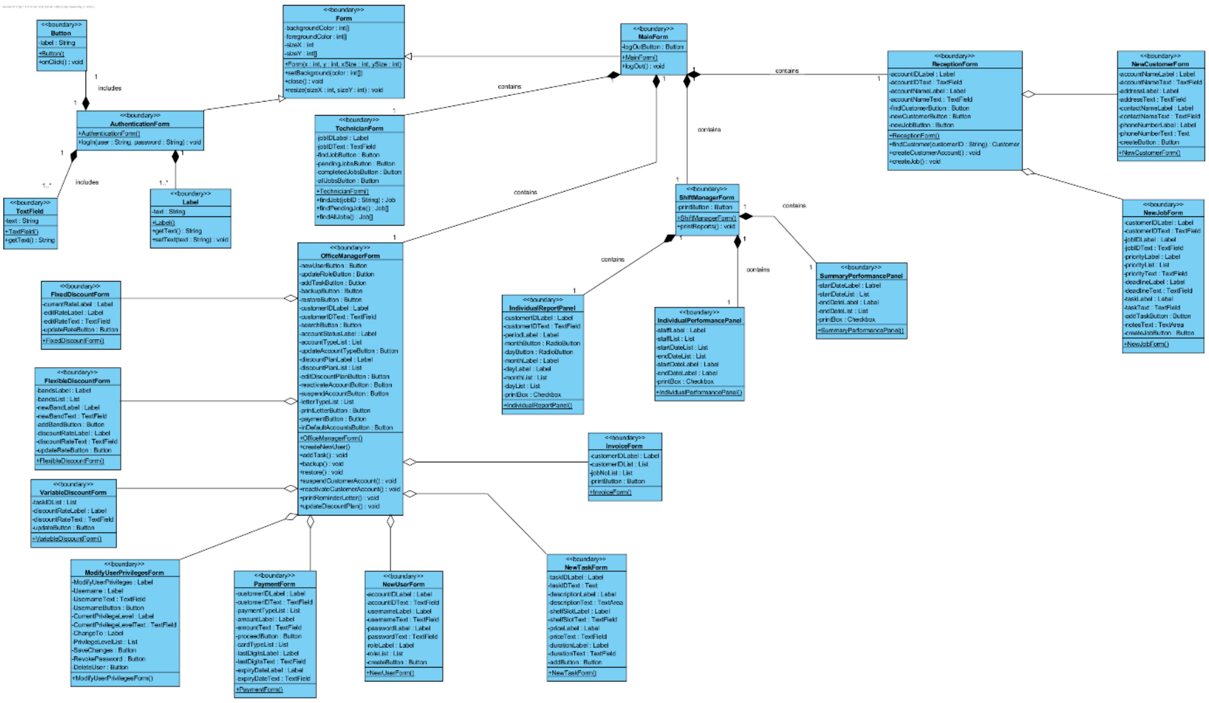


Database Package



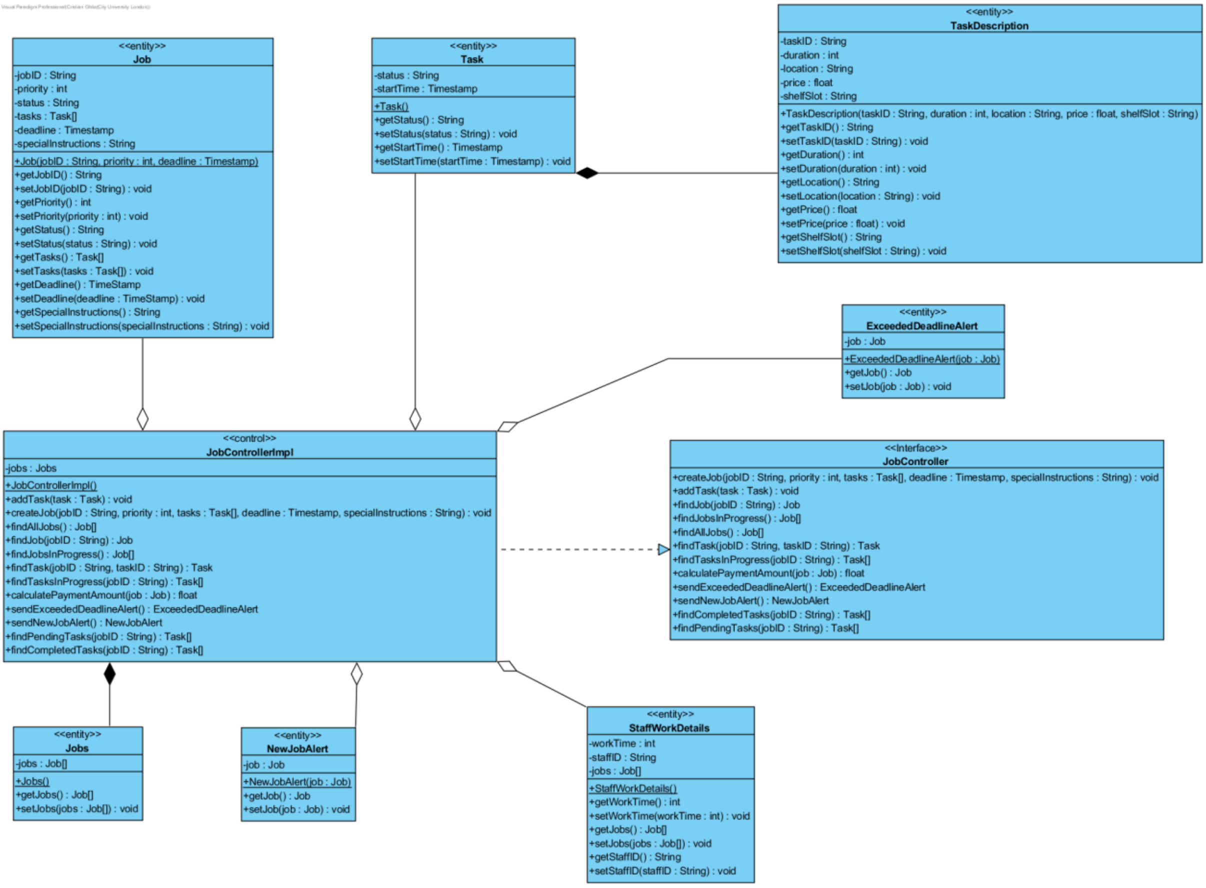
<<Interface>>
DBConnectivity
+read(sql : String, connection : Connection) : ResultSet
+write(sql : String, connection : Connection) : void
+connection(sql : String) : Connection
+closeConnection(connection : Connection) : void
DBConnectivityImpl
+read(sql : String, connection : Connection) : ResultSet
+closeConnection(connection : Connection) : void
+connection(sql : String) : Connection
+write(sql : String, connection : Connection) : void
+DBConnectivityImpl()

Graphical User Interface (GUI) Package



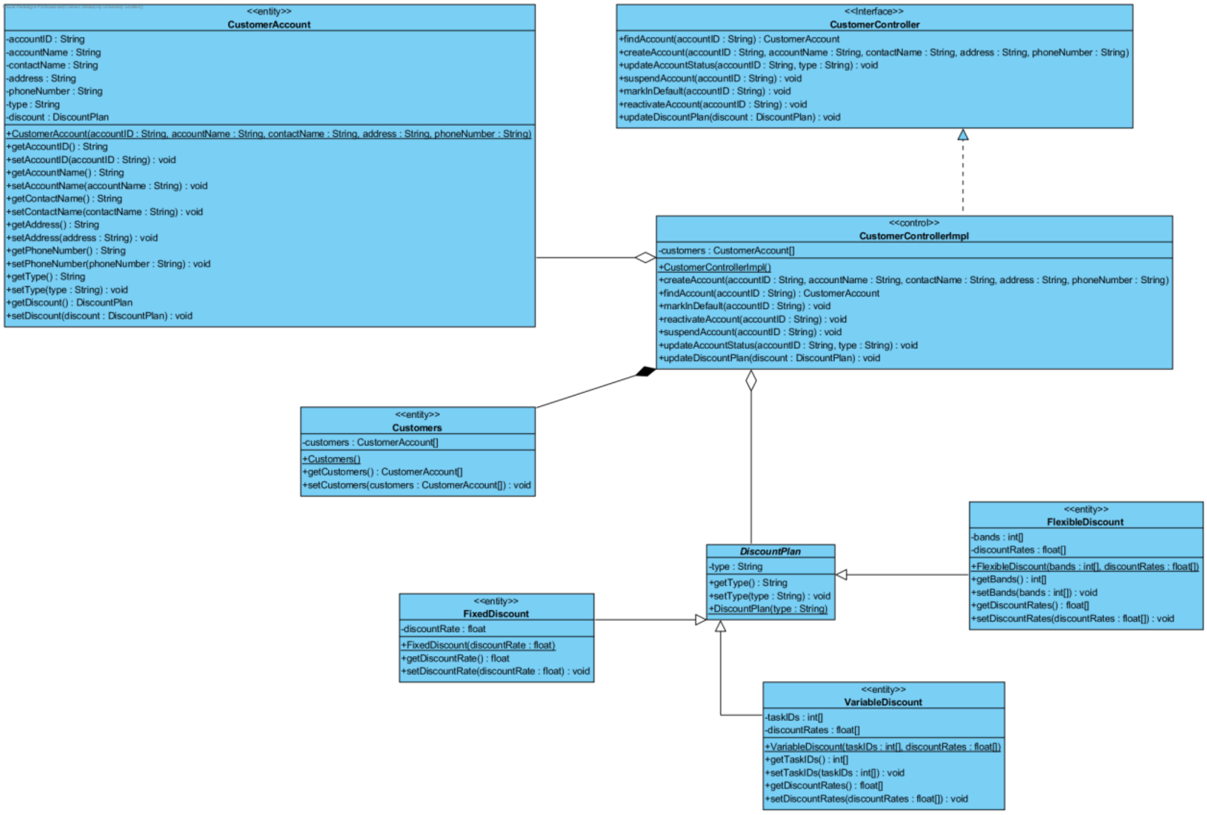
<<boundary>>
Button
-label : String
+Button()
+onClick() : void
1
includes
1
<<boundary>>
AuthenticationForm
+AuthenticationForm()
+logIn(user : String, password : String) : void
1
1..*
includes
1
1..*
<<boundary>>
TextField
-text : String
+TextField()
+getText() : String
<<boundary>>
Label
-text : String
+Label()
+getText() : String
+setText(text : String) : void
<<boundary>>
Form
-backgroundColor : int[]
-foregroundColor : int[]
-sizeX : int
-sizeY : int[]
+Form(x : int, y : int, xSize : int, ySize : int)
+setBackground(color : int[])
+close() : void
+resize(sizeX : int, sizeY : int) : void
<<boundary>>
TechnicianForm
-jobIDLabel : Label
-jobIDText : TextField
-findJobButton : Button
-pendingJobsButton : Button
-completedJobsButton : Button
-allJobsButton : Button
+TechnicianForm()
+findJob(jobID : String) : Job
+findPendingJobs() : Job[]
+findAllJobs() : Job[]
<<boundary>>
MainForm
-logOutButton : Button
+MainForm()
+logOut() : void
contains
contains
contains
contains
1
<<boundary>>
ReceptionForm
-accountIDLabel : Label
-accountIDText : TextField
-accountNameLabel : Label
-accountNameText : TextField
-findCustomerButton : Button
-newCustomerButton : Button
-newJobButton : Button
+ReceptionForm()
+findCustomer(customerID : String) : Customer
+createCustomerAccount() : void
+createJob() : void
<<boundary>>
NewCustomerForm
-accountNameLabel : Label
-accountNameText : TextField
-addressLabel : Label
-addressText : TextField
-contactNameLabel : Label
-contactNameText : TextField
-phoneNumberLabel : Label
-phoneNumberText : Text
-createButton : Button
+NewCustomerForm()
<<boundary>>
NewJobForm
-customerIDLabel : Label
-customerIDText : TextField
-jobIDLabel : Label
-jobIDText : TextField
-priorityLabel : Label
-priorityList : List
-priorityText : TextField
-deadlineLabel : Label
-deadlineText : TextField
-taskLabel : Label
-taskText : TextField
-addTaskButton : Button
-notesText : TextArea
-createJobButton : Button
+NewJobForm()
<<boundary>>
ShiftManagerForm
-printButton : Button
+ShiftManagerForm()
+printReports() : void
contains
contains
contains
<<boundary>>
IndividualReportPanel
-customerIDLabel : Label
-customerIDText : TextField
-periodLabel : Label
-monthButton : RadioButton
-dayButton : RadioButton
-monthLabel : Label
-dayLabel : Label
-monthList : List
-dayList : List
-printBox : Checkbox
+IndividualReportPanel()
<<boundary>>
IndividualPerformancePanel
-staffLabel : Label
-staffList : List
-startDateList : List
-endDateList : List
-startDateLabel : Label
-endDateLabel : Label
-printBox : Checkbox
+IndividualPerformancePanel()
<<boundary>>
SummaryPerformancePanel
-startDateLabel : Label
-startDateList : List
-endDateLabel : Label
-endDateList : List
-printBox : Checkbox
+SummaryPerformancePanel()
<<boundary>>
OfficeManagerForm
-newUserButton : Button
-updateRoleButton : Button
-addTaskButton : Button
-backupButton : Button
-restoreButton : Button
-customerIDLabel : Label
-customerIDText : TextField
-searchButton : Button
-accountStatusLabel : Label
-accountTypeList : List
-updateAccountTypeButton : Button
-discountPlanLabel : Label
-discountPlanList : List
-editDiscountPlanButton : Button
-reactivateAccountButton : Button
-suspendAccountButton : Button
-letterTypeList : List
-printLetterButton : Button
-paymentButton : Button
-inDefaultAccountsButton : Button
+OfficeManagerForm()
+createNewUser()
+addTask() : void
+backup() : void
+restore() : void
+suspendCustomerAccount() : void
+reactivateCustomerAccount() : void
+printReminderLetter() : void
+updateDiscountPlan() : void
<<boundary>>
FixedDiscountForm
-currentRateLabel : Label
-editRateLabel : Label
-editRateText : TextField
-updateRateButton : Button
+FixedDiscountForm()
<<boundary>>
FlexibleDiscountForm
-bandsLabel : Label
-bandsList : List
-newBandLabel : Label
-newBandText : TextField
-addBandButton : Button
-discountRateLabel : Label
-discountRateText : TextField
-updateRateButton : Button
+FlexibleDiscountForm()
<<boundary>>
VariableDiscountForm
-taskIDList : List
-discountRateLabel : Label
-discountRateText : TextField
-updateButton : Button
+VariableDiscountForm()
<<boundary>>
InvoiceForm
-customerIDLabel : Label
-customerIDList : List
-jobNoList : List
-printButton : Button
+InvoiceForm()
<<boundary>>
ModifyUserPrivilegesForm
-ModifyUserPrivileges : Label
-Username : Label
-UsernameText : TextField
-UsernameButton : Button
-CurrentPrivilegeLevel : Label
-CurrentPrivilegeLevelText : TextField
-ChangeTo : Label
-PrivilegeLevelList : List
-SaveChanges : Button
-RevokePassword : Button
-DeleteUser : Button
+ModifyUserPrivilegesForm()
<<boundary>>
PaymentForm
-customerIDLabel : Label
-customerIDText : TextField
-paymentTypeList : List
-amountLabel : Label
-amountText : TextField
-proceedButton : Button
-cardTypeList : List
-lastDigitsLabel : Label
-lastDigitsText : TextField
-expiryDateLabel : Label
-expiryDateText : TextField
+PaymentForm()
<<boundary>>
NewUserForm
-accountIDLabel : Label
-accountIDText : TextField
-usernameLabel : Label
-usernameText : TextField
-passwordLabel : Label
-passwordText : TextField
-roleLabel : Label
-roleList : List
-createButton : Button
+NewUserForm()
<<boundary>>
NewTaskForm
-taskIDLabel : Label
-taskIDText : Text
-descriptionLabel : Label
-descriptionText : TextArea
-shelfSlotLabel : Label
-shelfSlotText : TextField
-priceLabel : Label
-priceText : TextField
-durationLabel : Label
-durationText : TextField
-addButton : Button
+NewTaskForm()

Job Package

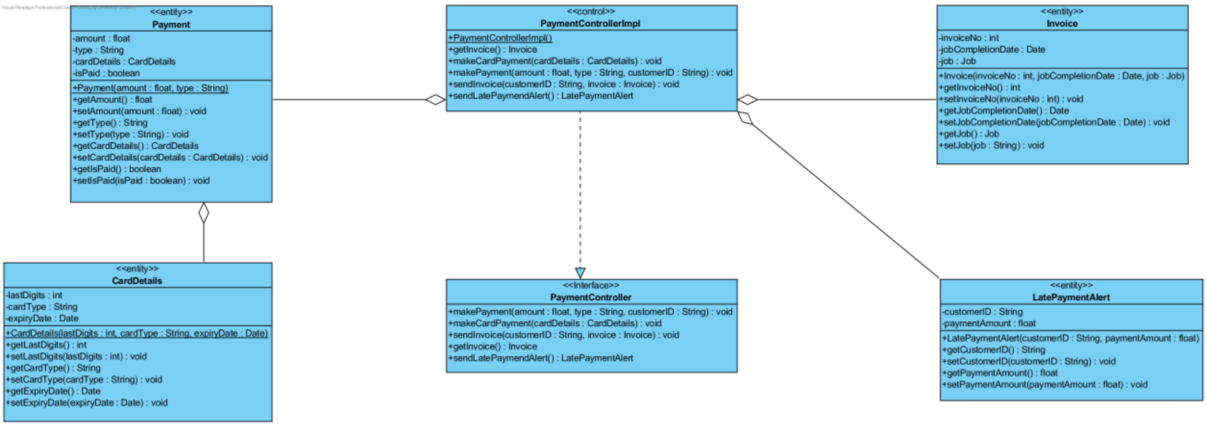


<<entity>>
Job
-jobID : String
-priority : int
-status : String
-tasks : Task[]
-deadline : Timestamp
-specialInstructions : String
+Job(jobID : String, priority : int, deadline : Timestamp)
+getJobID() : String
+setJobID(jobID : String) : void
+getPriority() : int
+setPriority(priority : int) : void
+getStatus() : String
+setStatus(status : String) : void
+getTasks() : Task[]
+setTasks(tasks : Task[]) : void
+getDeadline() : TimeStamp
+setDeadline(deadline : TimeStamp) : void
+getSpecialInstructions() : String
+setSpecialInstructions(specialInstructions : String) : void
<<entity>>
Task
-status : String
-startTime : Timestamp
+Task()
+getStatus() : String
+setStatus(status : String) : void
+getStartTime() : Timestamp
+setStartTime(startTime : Timestamp) : void
<<entity>>
TaskDescription
-taskID : String
-duration : int
-location : String
-price : float
-shelfSlot : String
+TaskDescription(taskID : String, duration : int, location : String, price : float, shelfSlot : String)
+getTaskID() : String
+setTaskID(taskID : String) : void
+getDuration() : int
+setDuration(duration : int) : void
+getLocation() : String
+setLocation(location : String) : void
+getPrice() : float
+setPrice(price : float) : void
+getShelfSlot() : String
+setShelfSlot(shelfSlot : String) : void
<<entity>>
ExceededDeadlineAlert
-job : Job
+ExceededDeadlineAlert(job : Job)
+getJob() : Job
+setJob(job : Job) : void
<<control>>
JobControllerImpl
-jobs : Jobs
+JobControllerImpl()
+addTask(task : Task) : void
+createJob(jobID : String, priority : int, tasks : Task[], deadline : Timestamp, specialInstructions : String) : void
+findAllJobs() : Job[]
+findJob(jobID : String) : Job
+findJobsInProgress() : Job[]
+findTask(jobID : String, taskID : String) : Task
+findTasksInProgress(jobID : String) : Task[]
+calculatePaymentAmount(job : Job) : float
+sendExceededDeadlineAlert() : ExceededDeadlineAlert
+sendNewJobAlert() : NewJobAlert
+findPendingTasks(jobID : String) : Task[]
+findCompletedTasks(jobID : String) : Task[]
<<Interface>>
JobController
+createJob(jobID : String, priority : int, tasks : Task[], deadline : Timestamp, specialInstructions : String) : void
+addTask(task : Task) : void
+findJob(jobID : String) : Job
+findJobsInProgress() : Job[]
+findAllJobs() : Job[]
+findTask(jobID : String, taskID : String) : Task
+findTasksInProgress(jobID : String) : Task[]
+calculatePaymentAmount(job : Job) : float
+sendExceededDeadlineAlert() : ExceededDeadlineAlert
+sendNewJobAlert() : NewJobAlert
+findCompletedTasks(jobID : String) : Task[]
+findPendingTasks(jobID : String) : Task[]
<<entity>>
Jobs
-jobs : Job[]
+Jobs()
+getJobs() : Job[]
+setJobs(jobs : Job[]) : void
<<entity>>
NewJobAlert
-job : Job
+NewJobAlert(job : Job)
+getJob() : Job
+setJob(job : Job) : void
<<entity>>
StaffWorkDetails
-workTime : int
-staffID : String
-jobs : Job[]
+StaffWorkDetails()
+getWorkTime() : int
+setWorkTime(workTime : int) : void
+getJobs() : Job[]
+setJobs(jobs : Job[]) : void
+getStaffID() : String
+setStaffID(staffID : String) : void

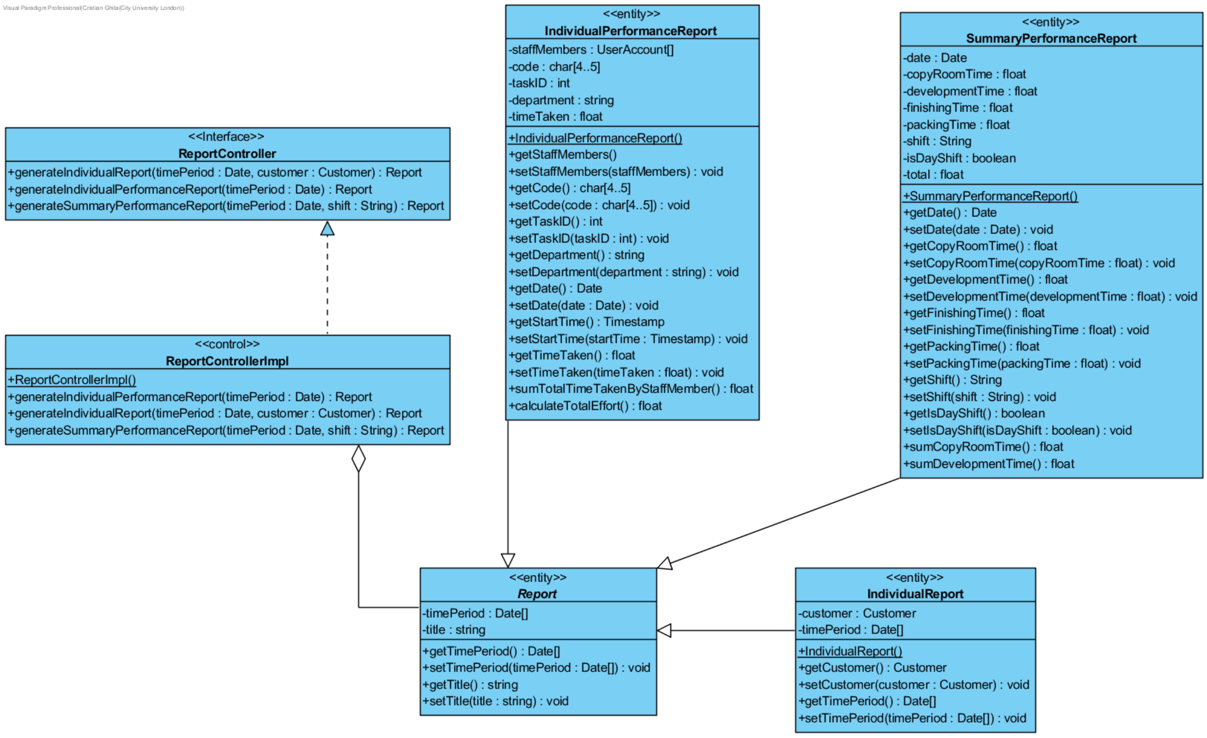
Customer Package


<<entity>>
CustomerAccount
-accountID : String
-accountName : String
-contactName : String
-address : String
-phoneNumber : String
-type : String
-discount : DiscountPlan
+CustomerAccount(accountID : String, accountName : String, contactName : String, address : String, phoneNumber : String)
+getAccountID() : String
+setAccountID(accountID : String) : void
+getAccountName() : String
+setAccountName(accountName : String) : void
+getContactName() : String
+setContactName(contactName : String) : void
+getAddress() : String
+setAddress(address : String) : void
+getPhoneNumber() : String
+setPhoneNumber(phoneNumber : String) : void
+getType() : String
+setType(type : String) : void
+getDiscount() : DiscountPlan
+setDiscount(discount : DiscountPlan) : void
<<Interface>>
CustomerController
+findAccount(accountID : String) : CustomerAccount
+createAccount(accountID : String, accountName : String, contactName : String, address : String, phoneNumber : String)
+updateAccountStatus(accountID : String, type : String) : void
+suspendAccount(accountID : String) : void
+markInDefault(accountID : String) : void
+reactivateAccount(accountID : String) : void
+updateDiscountPlan(discount : DiscountPlan) : void
<<control>>
CustomerControllerImpl
-customers : CustomerAccount[]
+CustomerControllerImpl()
+createAccount(accountID : String, accountName : String, contactName : String, address : String, phoneNumber : String)
+findAccount(accountID : String) : CustomerAccount
+markInDefault(accountID : String) : void
+reactivateAccount(accountID : String) : void
+suspendAccount(accountID : String) : void
+updateAccountStatus(accountID : String, type : String) : void
+updateDiscountPlan(discount : DiscountPlan) : void
<<entity>>
Customers
-customers : CustomerAccount[]
+Customers()
+getCustomers() : CustomerAccount[]
+setCustomers(customers : CustomerAccount[]) : void
<<entity>>
FlexibleDiscount
-bands : int[]
-discountRates : float[]
+FlexibleDiscount(bands : int[], discountRates : float[])
+getBands() : int[]
+setBands(bands : int[]) : void
+getDiscountRates() : float[]
+setDiscountRates(discountRates : float[]) : void
DiscountPlan
-type : String
+getType() : String
+setType(type : String) : void
+DiscountPlan(type : String)
<<entity>>
FixedDiscount
-discountRate : float
+FixedDiscount(discountRate : float)
+getDiscountRate() : float
+setDiscountRate(discountRate : float) : void
<<entity>>
VariableDiscount
-taskIDs : int[]
-discountRates : float[]
+VariableDiscount(taskIDs : int[], discountRates : float[])
+getTaskIDs() : int[]
+setTaskIDs(taskIDs : int[]) : void
+getDiscountRates() : float[]
+setDiscountRates(discountRates : float[]) : void

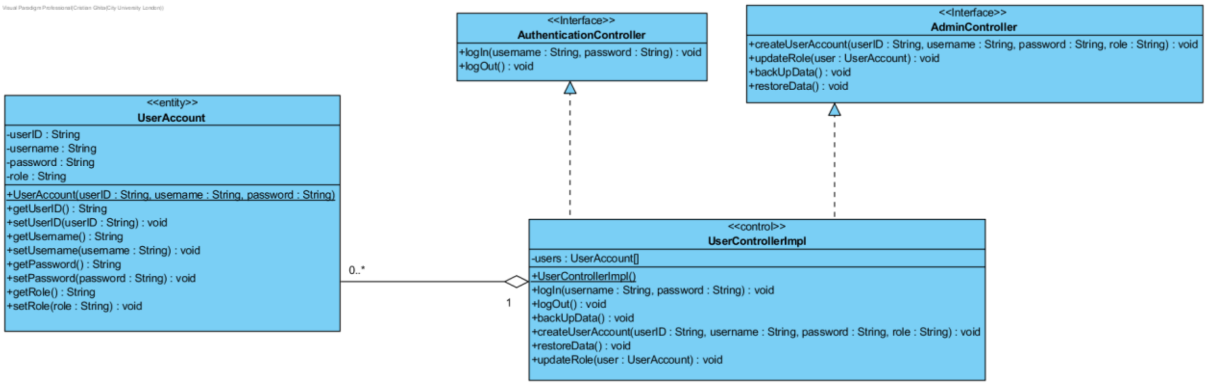
Payment Package


<<entity>>
Payment
-amount : float
-type : String
-cardDetails : CardDetails
-isPaid : boolean
+Payment(amount : float, type : String)
+getAmount() : float
+setAmount(amount : float) : void
+getType() : String
+setType(type : String) : void
+getCardDetails() : CardDetails
+setCardDetails(cardDetails : CardDetails) : void
+getIsPaid() : boolean
+setIsPaid(isPaid : boolean) : void
<<control>>
PaymentControllerImpl
+PaymentControllerImpl()
+getInvoice() : Invoice
+makeCardPayment(cardDetails : CardDetails) : void
+makePayment(amount : float, type : String, customerID : String) : void
+sendInvoice(customerID : String, invoice : Invoice) : void
+sendLatePaymendAlert() : LatePaymentAlert
<<entity>>
Invoice
-invoiceNo : int
-jobCompletionDate : Date
-job : Job
+Invoice(invoiceNo : int, jobCompletionDate : Date, job : Job)
+getInvoiceNo() : int
+setInvoiceNo(invoiceNo : int) : void
+getJobCompletionDate() : Date
+setJobCompletionDate(jobCompletionDate : Date) : void
+getJob() : Job
+setJob(job : String) : void
<<entity>>
CardDetails
-lastDigits : int
-cardType : String
-expiryDate : Date
+CardDetails(lastDigits : int, cardType : String, expiryDate : Date)
+getLastDigits() : int
+setLastDigits(lastDigits : int) : void
+getCardType() : String
+setCardType(cardType : String) : void
+getExpiryDate() : Date
+setExpiryDate(expiryDate : Date) : void
<<Interface>>
PaymentController
+makePayment(amount : float, type : String, customerID : String) : void
+makeCardPayment(cardDetails : CardDetails) : void
+sendInvoice(customerID : String, invoice : Invoice) : void
+getInvoice() : Invoice
+sendLatePaymendAlert() : LatePaymentAlert
<<entity>>
LatePaymentAlert
-customerID : String
-paymentAmount : float
+LatePaymentAlert(customerID : String, paymentAmount : float)
+getCustomerID() : String
+setCustomerID(customerID : String) : void
+getPaymentAmount() : float
+setPaymentAmount(paymentAmount : float) : void

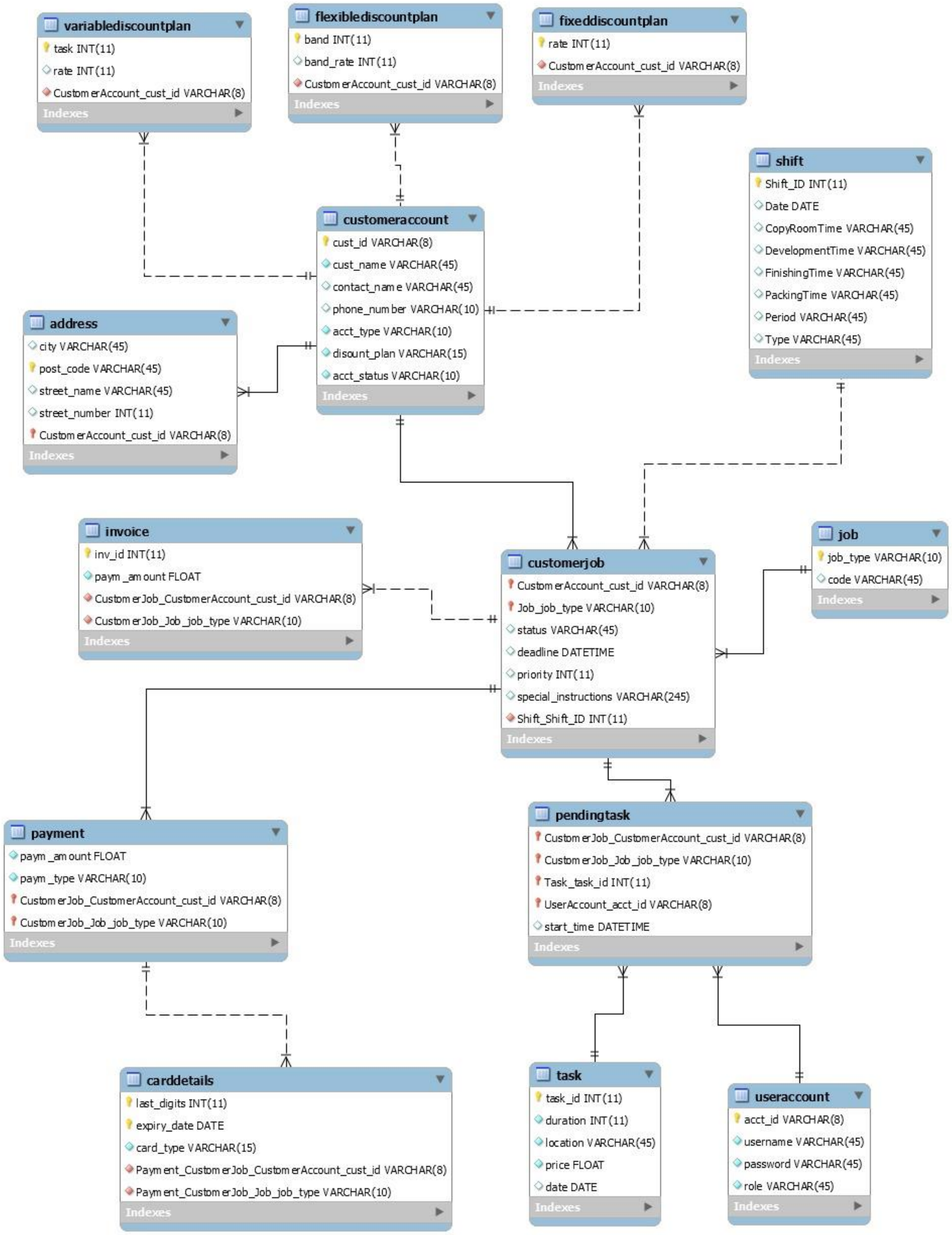
Report Package


Visual Paradigm Professional(Cristian Ghita(City University London))
<<Interface>>
ReportController
+generateIndividualReport(timePeriod : Date, customer : Customer) : Report
+generateIndividualPerformanceReport(timePeriod : Date) : Report
+generateSummaryPerformanceReport(timePeriod : Date, shift : String) : Report
<<control>>
ReportControllerImpl
+ReportControllerImpl()
+generateIndividualPerformanceReport(timePeriod : Date) : Report
+generateIndividualReport(timePeriod : Date, customer : Customer) : Report
+generateSummaryPerformanceReport(timePeriod : Date, shift : String) : Report
<<entity>>
IndividualPerformanceReport
-staffMembers : UserAccount[]
-code : char[4..5]
-taskID : int
-department : string
-timeTaken : float
+IndividualPerformanceReport()
+getStaffMembers()
+setStaffMembers(staffMembers) : void
+getCode() : char[4..5]
+setCode(code : char[4..5]) : void
+getTaskID() : int
+setTaskID(taskID : int) : void
+getDepartment() : string
+setDepartment(department : string) : void
+getDate() : Date
+setDate(date : Date) : void
+getStartTime() : Timestamp
+setStartTime(startTime : Timestamp) : void
+getTimeTaken() : float
+setTimeTaken(timeTaken : float) : void
+sumTotalTimeTakenByStaffMember() : float
+calculateTotalEffort() : float
<<entity>>
SummaryPerformanceReport
-date : Date
-copyRoomTime : float
-developmentTime : float
-finishingTime : float
-packingTime : float
-shift : String
-isDayShift : boolean
-total : float
+SummaryPerformanceReport()
+getDate() : Date
+setDate(date : Date) : void
+getCopyRoomTime() : float
+setCopyRoomTime(copyRoomTime : float) : void
+getDevelopmentTime() : float
+setDevelopmentTime(developmentTime : float) : void
+getFinishingTime() : float
+setFinishingTime(finishingTime : float) : void
+getPackingTime() : float
+setPackingTime(packingTime : float) : void
+getShift() : String
+setShift(shift : String) : void
+getIsDayShift() : boolean
+setIsDayShift(isDayShift : boolean) : void
+sumCopyRoomTime() : float
+sumDevelopmentTime() : float
<<entity>>
Report
-timePeriod : Date[]
-title : string
+getTimePeriod() : Date[]
+setTimePeriod(timePeriod : Date[]) : void
+getTitle() : string
+setTitle(title : string) : void
<<entity>>
IndividualReport
-customer : Customer
-timePeriod : Date[]
+IndividualReport()
+getCustomer() : Customer
+setCustomer(customer : Customer) : void
+getTimePeriod() : Date[]
+setTimePeriod(timePeriod : Date[]) : void

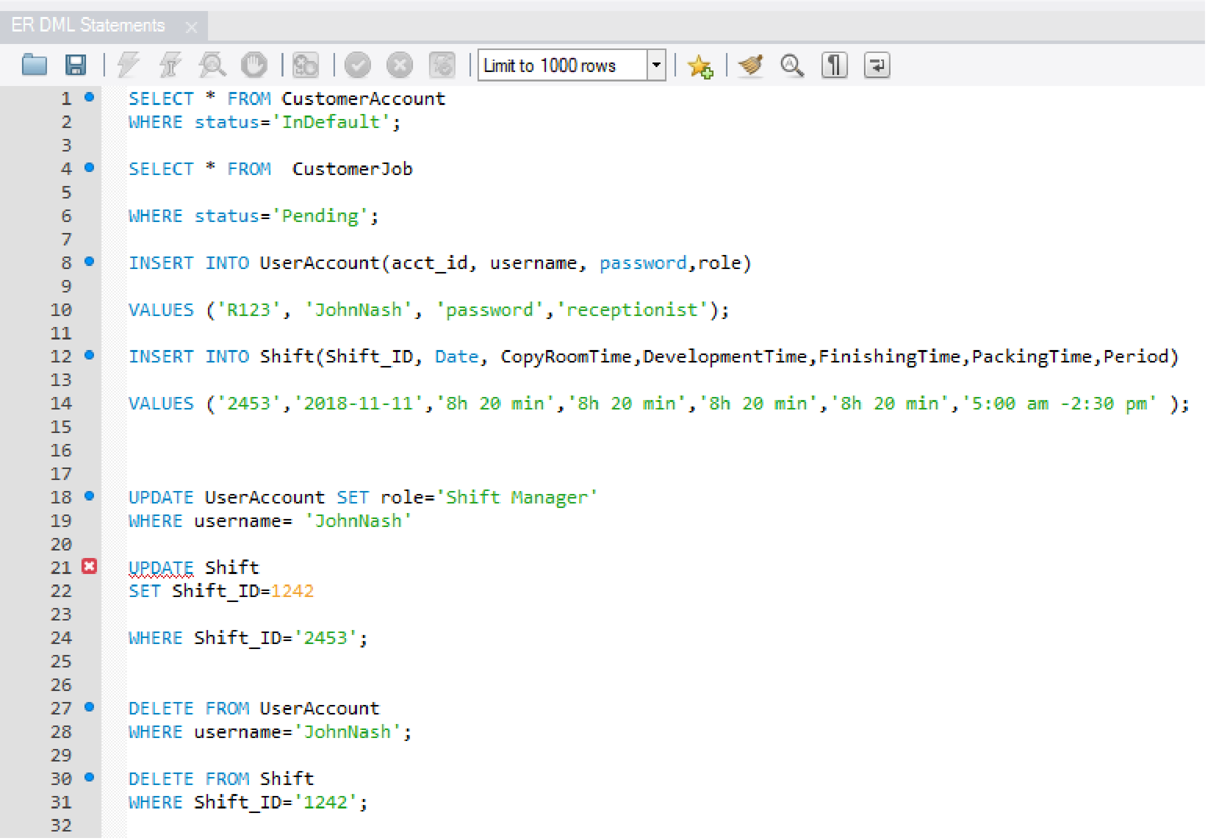
Staff Package


Visual Paradigm Professional(Cristian Ghita(City University London))
<<Interface>>
AuthenticationController
+logIn(username : String, password : String) : void
+logOut() : void
<<Interface>>
AdminController
+createUserAccount(userID : String, username : String, password : String, role : String) : void
+updateRole(user : UserAccount) : void
+backUpData() : void
+restoreData() : void
<<entity>>
UserAccount
-userID : String
-username : String
-password : String
-role : String
+UserAccount(userID : String, username : String, password : String)
+getUserID() : String
+setUserID(userID : String) : void
+getUsername() : String
+setUsername(username : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getRole() : String
+setRole(role : String) : void
0..*
1
<<control>>
UserControllerImpl
-users : UserAccount[]
+UserControllerImpl()
+logIn(username : String, password : String) : void
+logOut() : void
+backUpData() : void
+createUserAccount(userID : String, username : String, password : String, role : String) : void
+restoreData() : void
+updateRole(user : UserAccount) : void

Entity-relation diagram



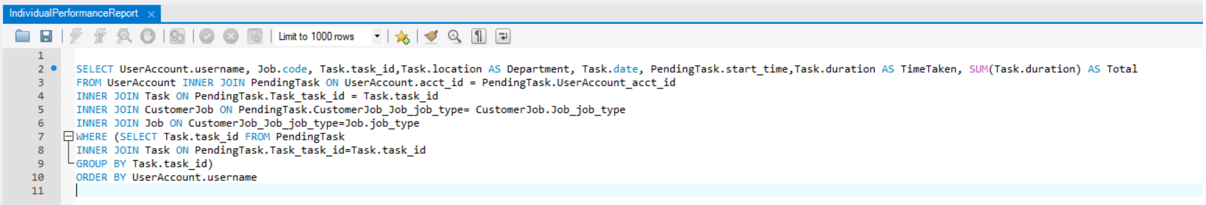
Data Manipulation Language statements (DML)



```
SELECT * FROM CustomerAccount
WHERE status='InDefault';

SELECT * FROM  CustomerJob

WHERE status='Pending';

INSERT INTO UserAccount(acct_id, username, password,role)

VALUES ('R123', 'JohnNash', 'password','receptionist');

INSERT INTO Shift(Shift_ID, Date, CopyRoomTime,DevelopmentTime,FinishingTime,PackingTime,Period)

VALUES ('2453','2018-11-11','8h 20 min','8h 20 min','8h 20 min','8h 20 min','5:00 am -2:30 pm' );



UPDATE UserAccount SET role='Shift Manager'
WHERE username= 'JohnNash'

UPDATE Shift
SET Shift_ID=1242

WHERE Shift_ID='2453';


DELETE FROM UserAccount
WHERE username='JohnNash';

DELETE FROM Shift
WHERE Shift_ID='1242';
```

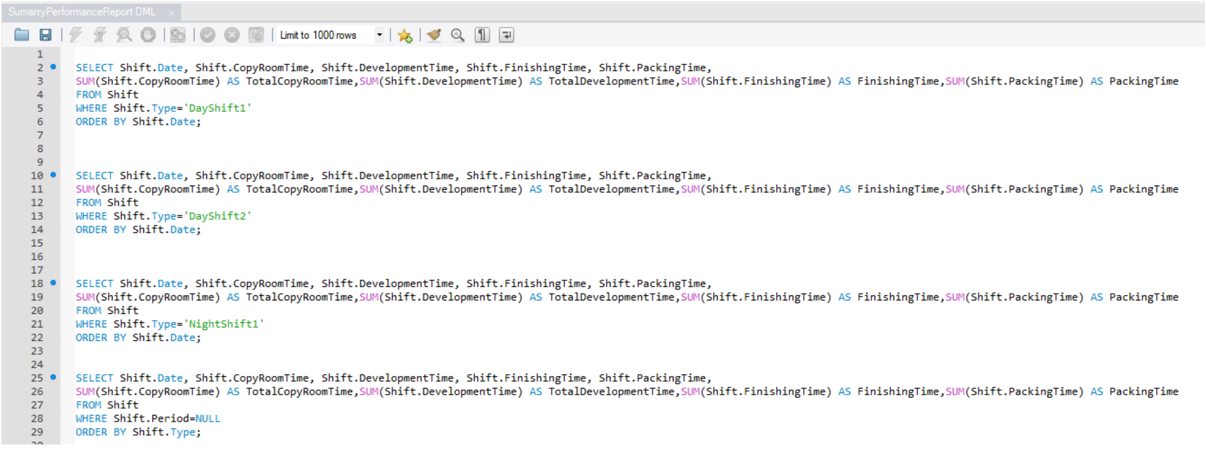
Non-trivial reports

Individual Performance Report



```

SELECT UserAccount.username, Job.code, Task.task_id,Task.location AS Department, Task.date, PendingTask.start_time,Task.duration AS TimeTaken, SUM(Task.duration) AS Total
FROM UserAccount INNER JOIN PendingTask ON UserAccount.acct_id = PendingTask.UserAccount_acct_id
INNER JOIN Task ON PendingTask.Task_task_id = Task.task_id
INNER JOIN CustomerJob ON PendingTask.CustomerJob_Job_job_type= CustomerJob.Job_job_type
INNER JOIN Job ON CustomerJob_Job_job_type=Job.job_type
WHERE (SELECT Task.task_id FROM PendingTask
INNER JOIN Task ON PendingTask.Task_task_id=Task.task_id
GROUP BY Task.task_id)
ORDER BY UserAccount.username
```

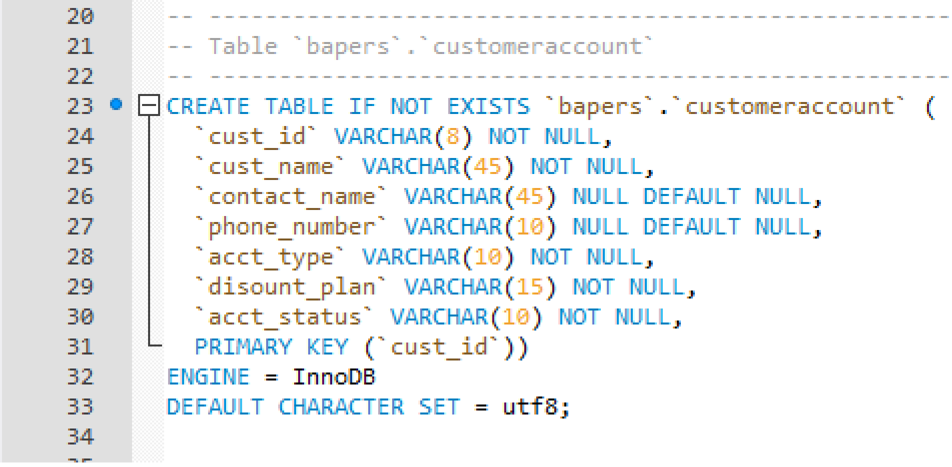
Summary Performance Report



```

SELECT Shift.Date, Shift.CopyRoomTime, Shift.DevelopmentTime, Shift.FinishingTime, Shift.PackingTime,
SUM(Shift.CopyRoomTime) AS TotalCopyRoomTime,SUM(Shift.DevelopmentTime) AS TotalDevelopmentTime,SUM(Shift.FinishingTime) AS FinishingTime,SUM(Shift.PackingTime) AS PackingTime
FROM Shift
WHERE Shift.Type='DayShift1'
ORDER BY Shift.Date;



SELECT Shift.Date, Shift.CopyRoomTime, Shift.DevelopmentTime, Shift.FinishingTime, Shift.PackingTime,
SUM(Shift.CopyRoomTime) AS TotalCopyRoomTime,SUM(Shift.DevelopmentTime) AS TotalDevelopmentTime,SUM(Shift.FinishingTime) AS FinishingTime,SUM(Shift.PackingTime) AS PackingTime
FROM Shift
WHERE Shift.Type='DayShift2'
ORDER BY Shift.Date;



SELECT Shift.Date, Shift.CopyRoomTime, Shift.DevelopmentTime, Shift.FinishingTime, Shift.PackingTime,
SUM(Shift.CopyRoomTime) AS TotalCopyRoomTime,SUM(Shift.DevelopmentTime) AS TotalDevelopmentTime,SUM(Shift.FinishingTime) AS FinishingTime,SUM(Shift.PackingTime) AS PackingTime
FROM Shift
WHERE Shift.Type='NightShift1'
ORDER BY Shift.Date;


SELECT Shift.Date, Shift.CopyRoomTime, Shift.DevelopmentTime, Shift.FinishingTime, Shift.PackingTime,
SUM(Shift.CopyRoomTime) AS TotalCopyRoomTime,SUM(Shift.DevelopmentTime) AS TotalDevelopmentTime,SUM(Shift.FinishingTime) AS FinishingTime,SUM(Shift.PackingTime) AS PackingTime
FROM Shift
WHERE Shift.Period=NULL
ORDER BY Shift.Type;
```

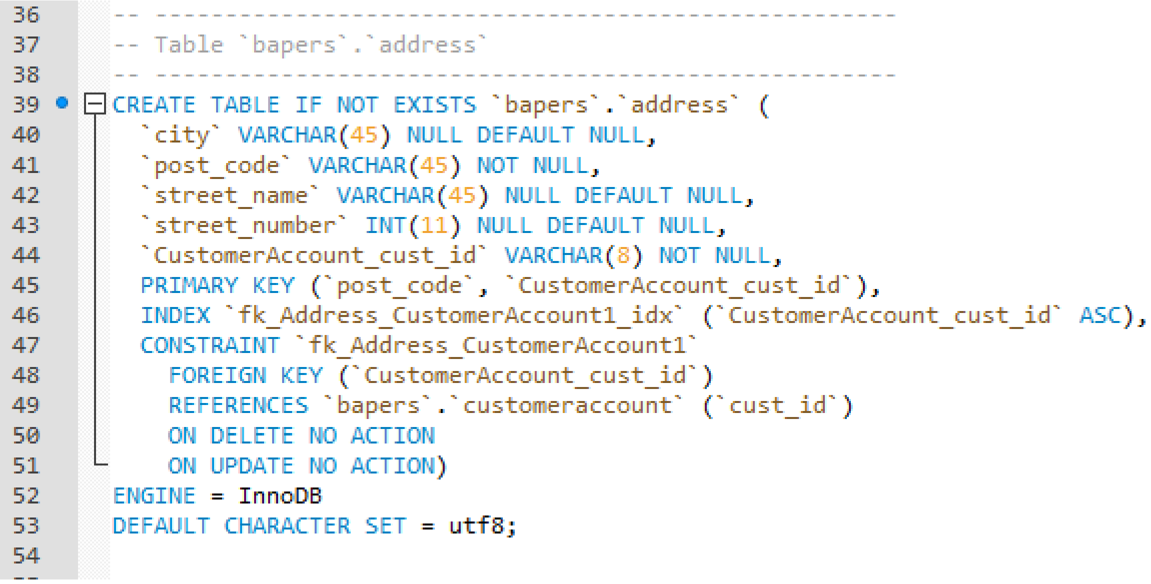
Data Definition Language statements (DDL)

Customer Account Table



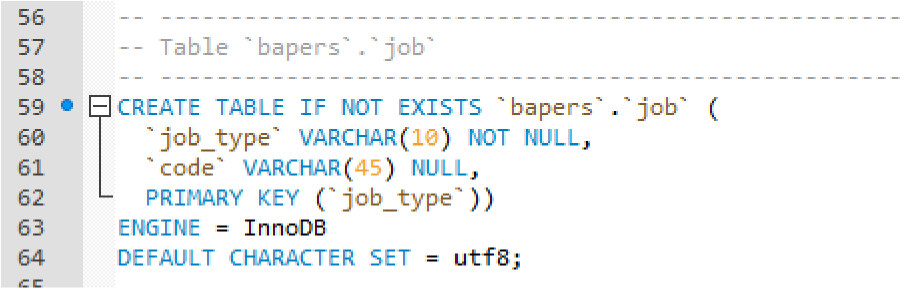
```sql
-- -----------------------------------------------------
-- Table `bapers`.`customeraccount`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`customeraccount` (
  `cust_id` VARCHAR(8) NOT NULL,
  `cust_name` VARCHAR(45) NOT NULL,
  `contact_name` VARCHAR(45) NULL DEFAULT NULL,
  `phone_number` VARCHAR(10) NULL DEFAULT NULL,
  `acct_type` VARCHAR(10) NOT NULL,
  `disount_plan` VARCHAR(15) NOT NULL,
  `acct_status` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`cust_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Address Table



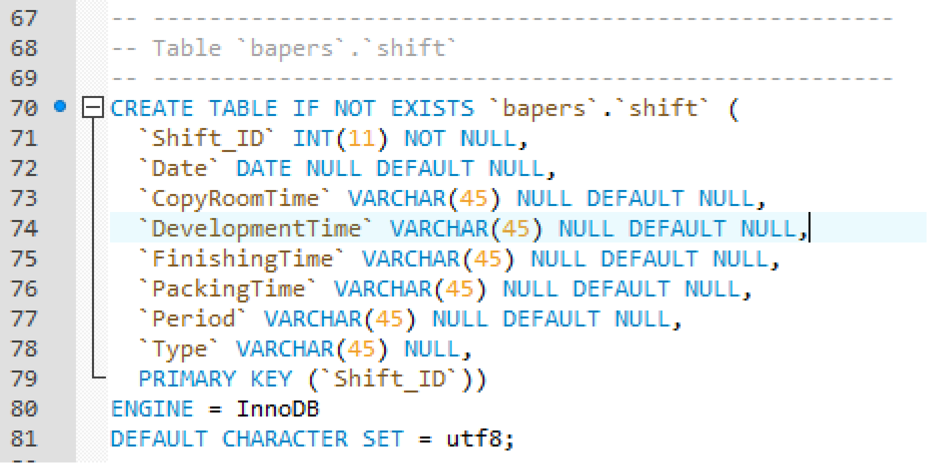
```sql
-- -----------------------------------------------------
-- Table `bapers`.`address`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`address` (
  `city` VARCHAR(45) NULL DEFAULT NULL,
  `post_code` VARCHAR(45) NOT NULL,
  `street_name` VARCHAR(45) NULL DEFAULT NULL,
  `street_number` INT(11) NULL DEFAULT NULL,
  `CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`post_code`, `CustomerAccount_cust_id`),
  INDEX `fk_Address_CustomerAccount1_idx` (`CustomerAccount_cust_id` ASC),
  CONSTRAINT `fk_Address_CustomerAccount1`
    FOREIGN KEY (`CustomerAccount_cust_id`)
    REFERENCES `bapers`.`customeraccount` (`cust_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Job Table



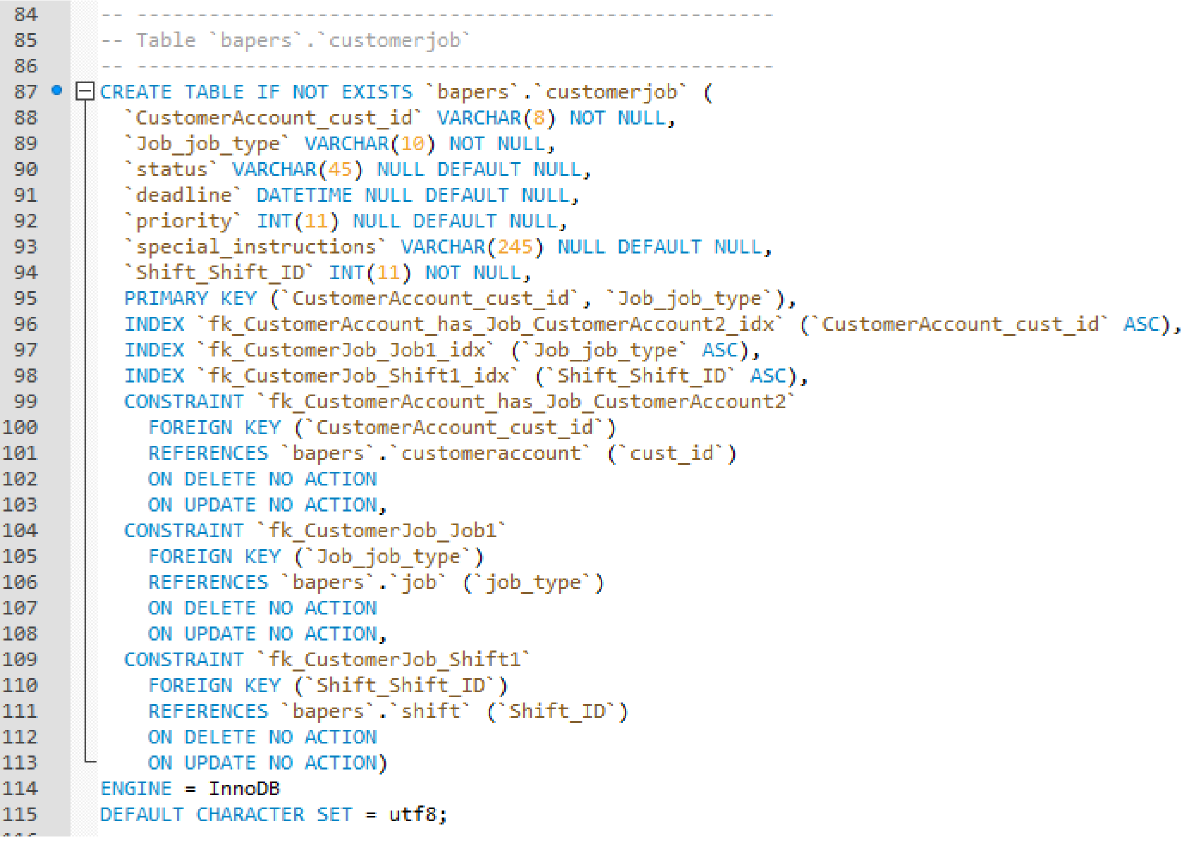
```sql
-- -----------------------------------------------------
-- Table `bapers`.`job`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`job` (
  `job_type` VARCHAR(10) NOT NULL,
  `code` VARCHAR(45) NULL,
  PRIMARY KEY (`job_type`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Shift Table



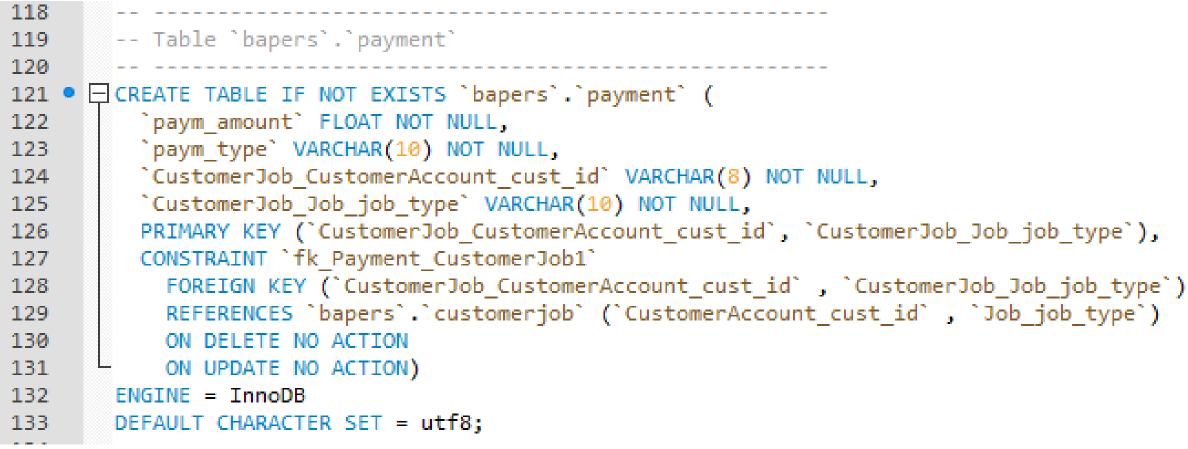
```sql
-- -----------------------------------------------------
-- Table `bapers`.`shift`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`shift` (
  `Shift_ID` INT(11) NOT NULL,
  `Date` DATE NULL DEFAULT NULL,
  `CopyRoomTime` VARCHAR(45) NULL DEFAULT NULL,
  `DevelopmentTime` VARCHAR(45) NULL DEFAULT NULL,
  `FinishingTime` VARCHAR(45) NULL DEFAULT NULL,
  `PackingTime` VARCHAR(45) NULL DEFAULT NULL,
  `Period` VARCHAR(45) NULL DEFAULT NULL,
  `Type` VARCHAR(45) NULL,
  PRIMARY KEY (`Shift_ID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Customer job Table



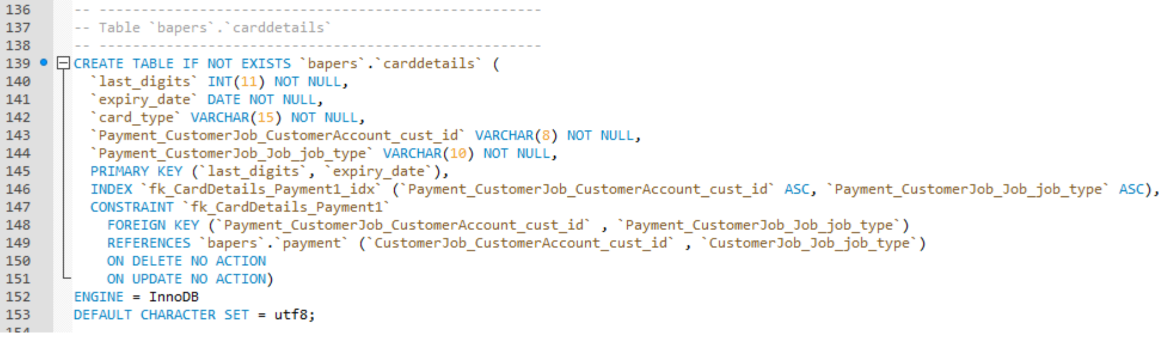
```sql
-- -----------------------------------------------------
-- Table `bapers`.`customerjob`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`customerjob` (
  `CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  `Job_job_type` VARCHAR(10) NOT NULL,
  `status` VARCHAR(45) NULL DEFAULT NULL,
  `deadline` DATETIME NULL DEFAULT NULL,
  `priority` INT(11) NULL DEFAULT NULL,
  `special_instructions` VARCHAR(245) NULL DEFAULT NULL,
  `Shift_Shift_ID` INT(11) NOT NULL,
  PRIMARY KEY (`CustomerAccount_cust_id`, `Job_job_type`),
  INDEX `fk_CustomerAccount_has_Job_CustomerAccount2_idx` (`CustomerAccount_cust_id` ASC),
  INDEX `fk_CustomerJob_Job1_idx` (`Job_job_type` ASC),
  INDEX `fk_CustomerJob_Shift1_idx` (`Shift_Shift_ID` ASC),
  CONSTRAINT `fk_CustomerAccount_has_Job_CustomerAccount2`
    FOREIGN KEY (`CustomerAccount_cust_id`)
    REFERENCES `bapers`.`customeraccount` (`cust_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CustomerJob_Job1`
    FOREIGN KEY (`Job_job_type`)
    REFERENCES `bapers`.`job` (`job_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CustomerJob_Shift1`
    FOREIGN KEY (`Shift_Shift_ID`)
    REFERENCES `bapers`.`shift` (`Shift_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Payment Table



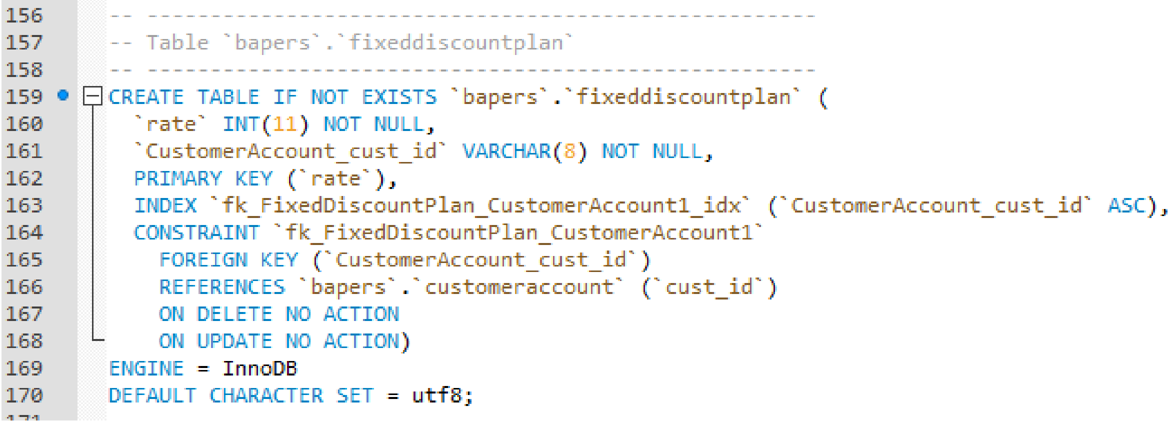
```sql
-- -----------------------------------------------------
-- Table `bapers`.`payment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`payment` (
  `paym_amount` FLOAT NOT NULL,
  `paym_type` VARCHAR(10) NOT NULL,
  `CustomerJob_CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  `CustomerJob_Job_job_type` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`CustomerJob_CustomerAccount_cust_id`, `CustomerJob_Job_job_type`),
  CONSTRAINT `fk_Payment_CustomerJob1`
    FOREIGN KEY (`CustomerJob_CustomerAccount_cust_id` , `CustomerJob_Job_job_type`)
    REFERENCES `bapers`.`customerjob` (`CustomerAccount_cust_id` , `Job_job_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Card details Table



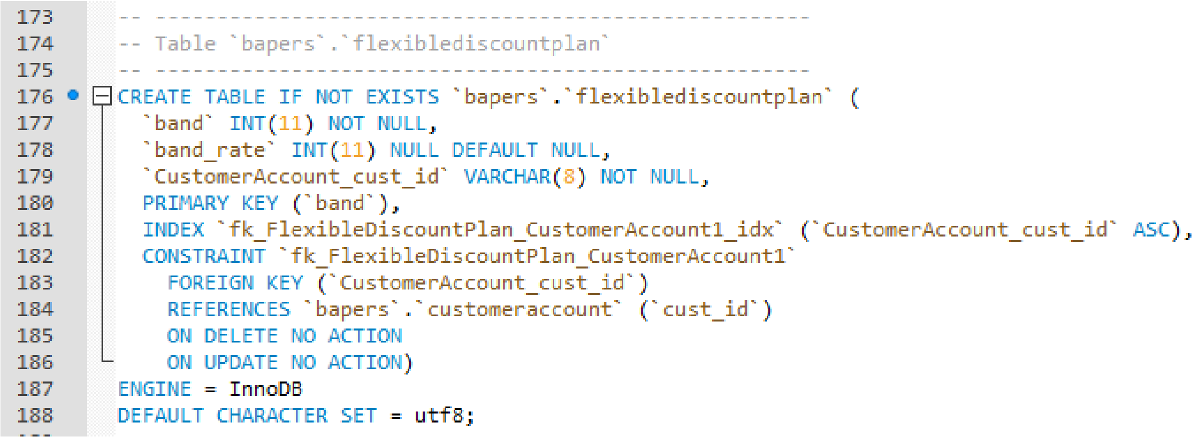
```sql
-- -----------------------------------------------------
-- Table `bapers`.`carddetails`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`carddetails` (
  `last_digits` INT(11) NOT NULL,
  `expiry_date` DATE NOT NULL,
  `card_type` VARCHAR(15) NOT NULL,
  `Payment_CustomerJob_CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  `Payment_CustomerJob_Job_job_type` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`last_digits`, `expiry_date`),
  INDEX `fk_CardDetails_Payment1_idx` (`Payment_CustomerJob_CustomerAccount_cust_id` ASC, `Payment_CustomerJob_Job_job_type` ASC),
  CONSTRAINT `fk_CardDetails_Payment1`
    FOREIGN KEY (`Payment_CustomerJob_CustomerAccount_cust_id` , `Payment_CustomerJob_Job_job_type`)
    REFERENCES `bapers`.`payment` (`CustomerJob_CustomerAccount_cust_id` , `CustomerJob_Job_job_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Fixed Discount Plan Table



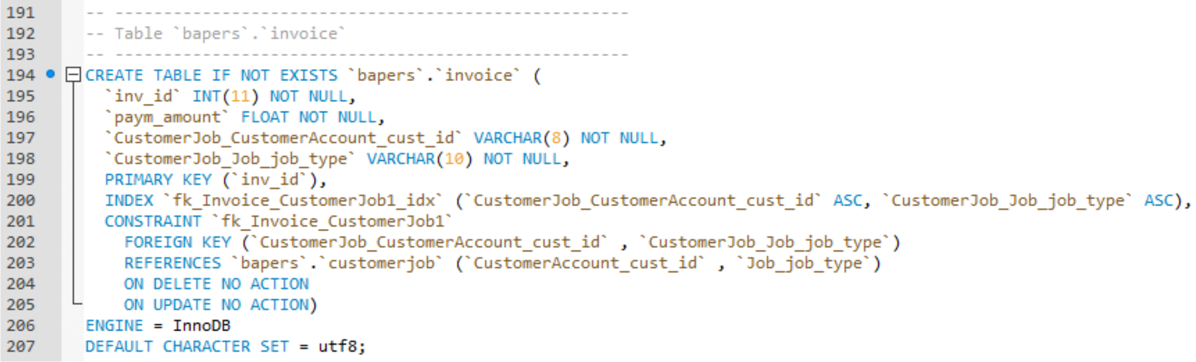
```sql
-- -----------------------------------------------------
-- Table `bapers`.`fixeddiscountplan`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`fixeddiscountplan` (
  `rate` INT(11) NOT NULL,
  `CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`rate`),
  INDEX `fk_FixedDiscountPlan_CustomerAccount1_idx` (`CustomerAccount_cust_id` ASC),
  CONSTRAINT `fk_FixedDiscountPlan_CustomerAccount1`
    FOREIGN KEY (`CustomerAccount_cust_id`)
    REFERENCES `bapers`.`customeraccount` (`cust_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Flexible Discount Plan Table



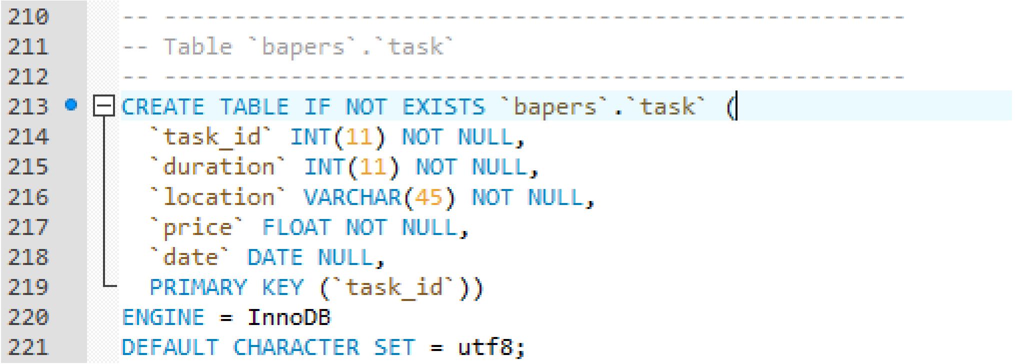
```
-- -----------------------------------------------------
-- Table `bapers`.`flexiblediscountplan`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`flexiblediscountplan` (
  `band` INT(11) NOT NULL,
  `band_rate` INT(11) NULL DEFAULT NULL,
  `CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  PRIMARY KEY (`band`),
  INDEX `fk_FlexibleDiscountPlan_CustomerAccount1_idx` (`CustomerAccount_cust_id` ASC),
  CONSTRAINT `fk_FlexibleDiscountPlan_CustomerAccount1`
    FOREIGN KEY (`CustomerAccount_cust_id`)
    REFERENCES `bapers`.`customeraccount` (`cust_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Invoice Table



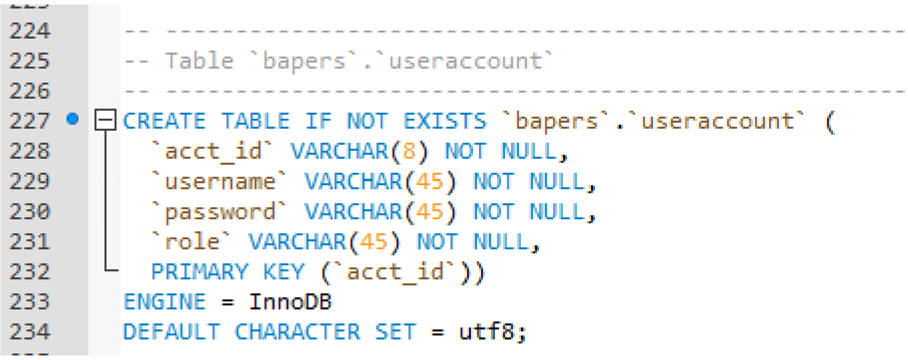
```
-- -----------------------------------------------------
-- Table `bapers`.`invoice`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`invoice` (
  `inv_id` INT(11) NOT NULL,
  `paym_amount` FLOAT NOT NULL,
  `CustomerJob_CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  `CustomerJob_Job_job_type` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`inv_id`),
  INDEX `fk_Invoice_CustomerJob1_idx` (`CustomerJob_CustomerAccount_cust_id` ASC, `CustomerJob_Job_job_type` ASC),
  CONSTRAINT `fk_Invoice_CustomerJob1`
    FOREIGN KEY (`CustomerJob_CustomerAccount_cust_id` , `CustomerJob_Job_job_type`)
    REFERENCES `bapers`.`customerjob` (`CustomerAccount_cust_id` , `Job_job_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Task Table



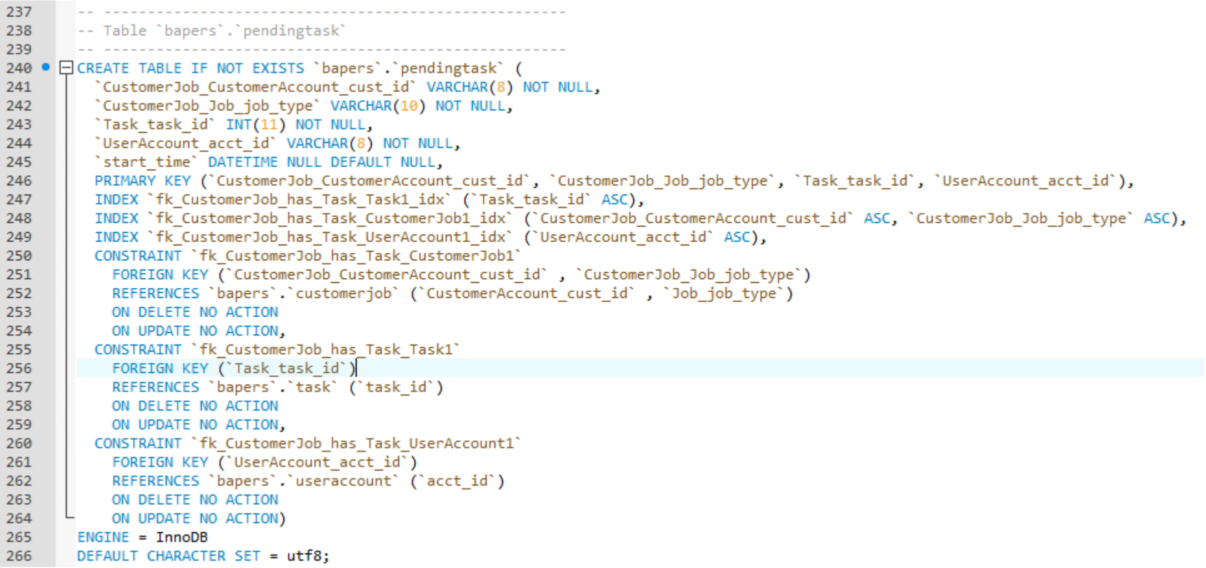
```
-- -----------------------------------------------------
-- Table `bapers`.`task`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`task` (
  `task_id` INT(11) NOT NULL,
  `duration` INT(11) NOT NULL,
  `location` VARCHAR(45) NOT NULL,
  `price` FLOAT NOT NULL,
  `date` DATE NULL,
  PRIMARY KEY (`task_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

User account Table



```sql
-- -----------------------------------------------------
-- Table `bapers`.`useraccount`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`useraccount` (
  `acct_id` VARCHAR(8) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `role` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`acct_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

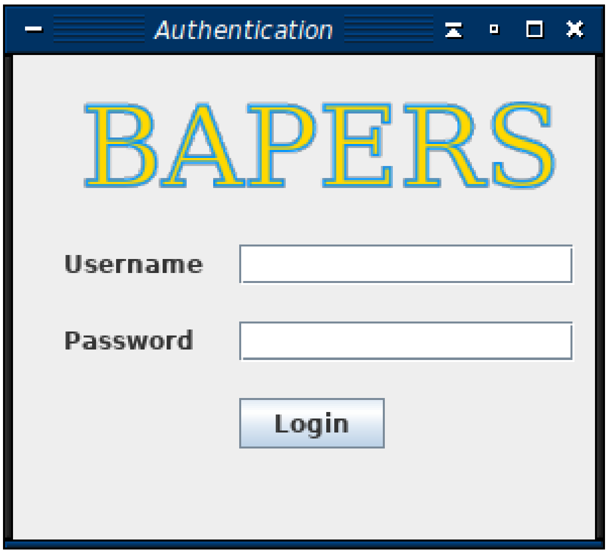
Pending Task Table



```sql
-- -----------------------------------------------------
-- Table `bapers`.`pendingtask`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `bapers`.`pendingtask` (
  `CustomerJob_CustomerAccount_cust_id` VARCHAR(8) NOT NULL,
  `CustomerJob_Job_job_type` VARCHAR(10) NOT NULL,
  `Task_task_id` INT(11) NOT NULL,
  `UserAccount_acct_id` VARCHAR(8) NOT NULL,
  `start_time` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`CustomerJob_CustomerAccount_cust_id`, `CustomerJob_Job_job_type`, `Task_task_id`, `UserAccount_acct_id`),
  INDEX `fk_CustomerJob_has_Task_Task1_idx` (`Task_task_id` ASC),
  INDEX `fk_CustomerJob_has_Task_CustomerJob1_idx` (`CustomerJob_CustomerAccount_cust_id` ASC, `CustomerJob_Job_job_type` ASC),
  INDEX `fk_CustomerJob_has_Task_UserAccount1_idx` (`UserAccount_acct_id` ASC),
  CONSTRAINT `fk_CustomerJob_has_Task_CustomerJob1`
    FOREIGN KEY (`CustomerJob_CustomerAccount_cust_id` , `CustomerJob_Job_job_type`)
    REFERENCES `bapers`.`customerjob` (`CustomerAccount_cust_id` , `Job_job_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CustomerJob_has_Task_Task1`
    FOREIGN KEY (`Task_task_id`)
    REFERENCES `bapers`.`task` (`task_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CustomerJob_has_Task_UserAccount1`
    FOREIGN KEY (`UserAccount_acct_id`)
    REFERENCES `bapers`.`useraccount` (`acct_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

GUI design

Authentication Form



Authentication

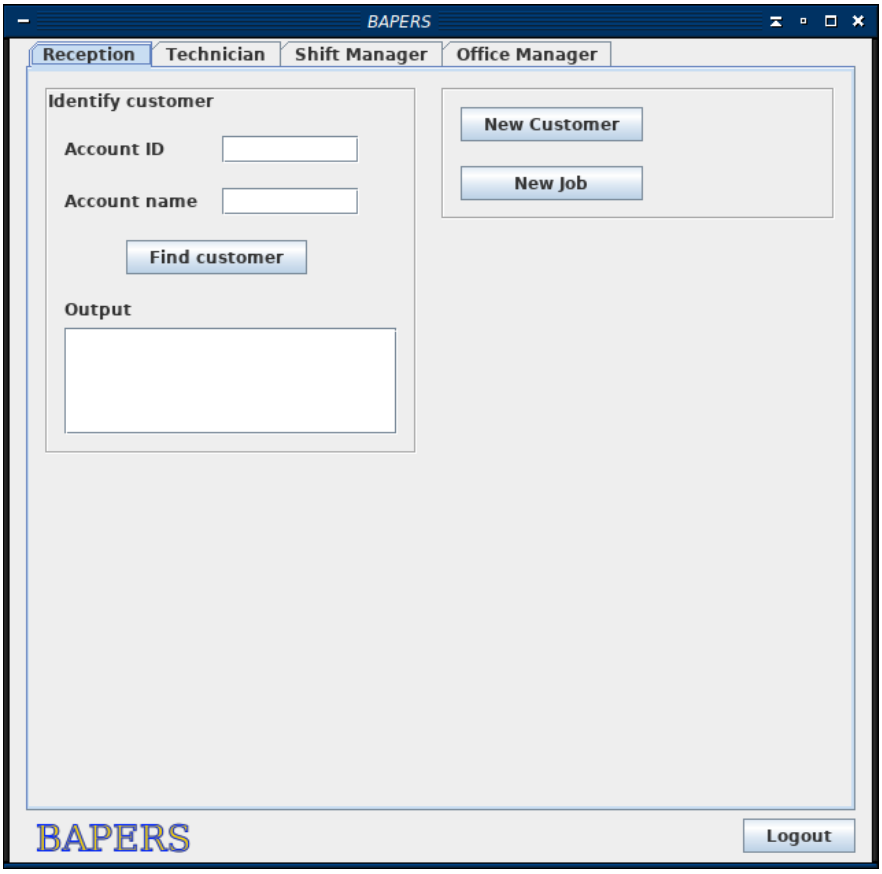
BAPERS

Username

Password

Login

Receptionist form



BAPERS

Reception | Technician | Shift Manager | Office Manager

Identify customer

Account ID

Account name

Find customer

Output

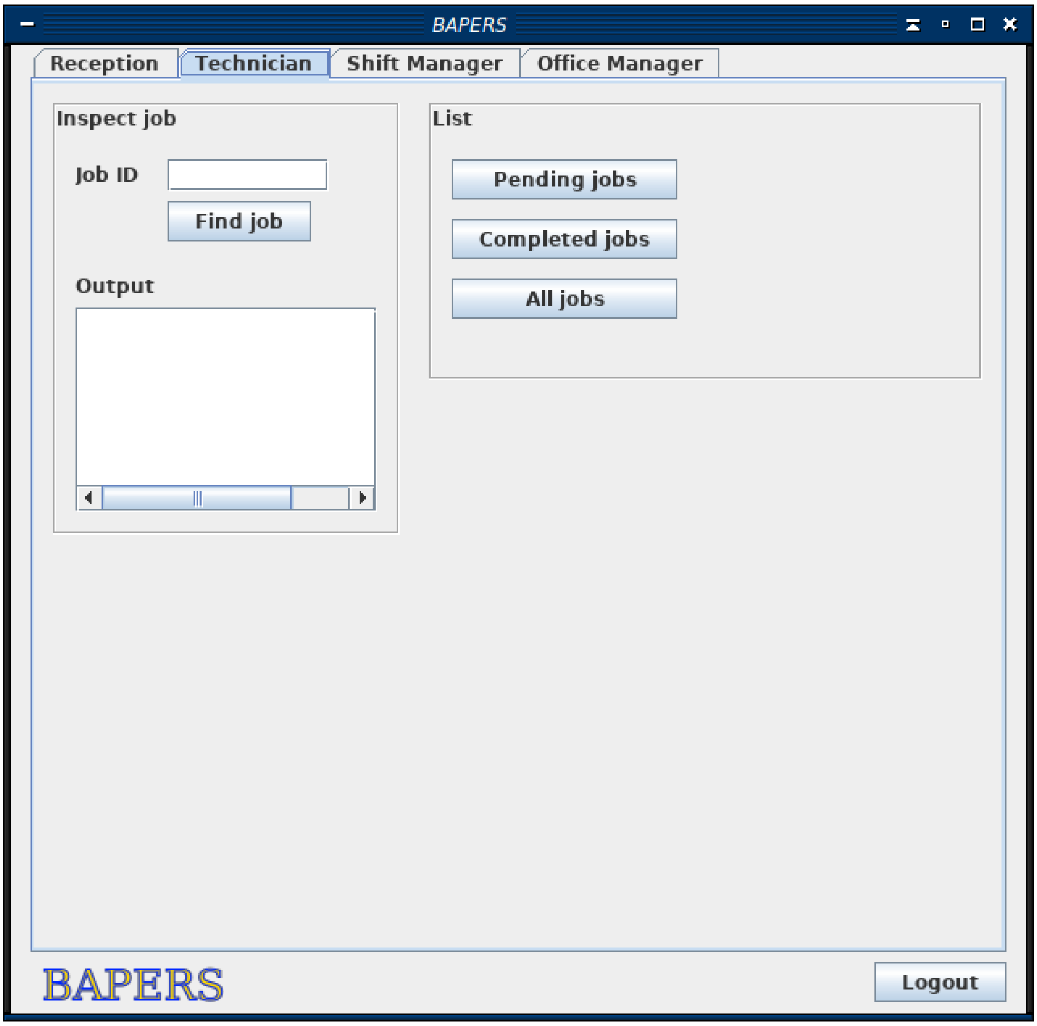
New Customer

New Job

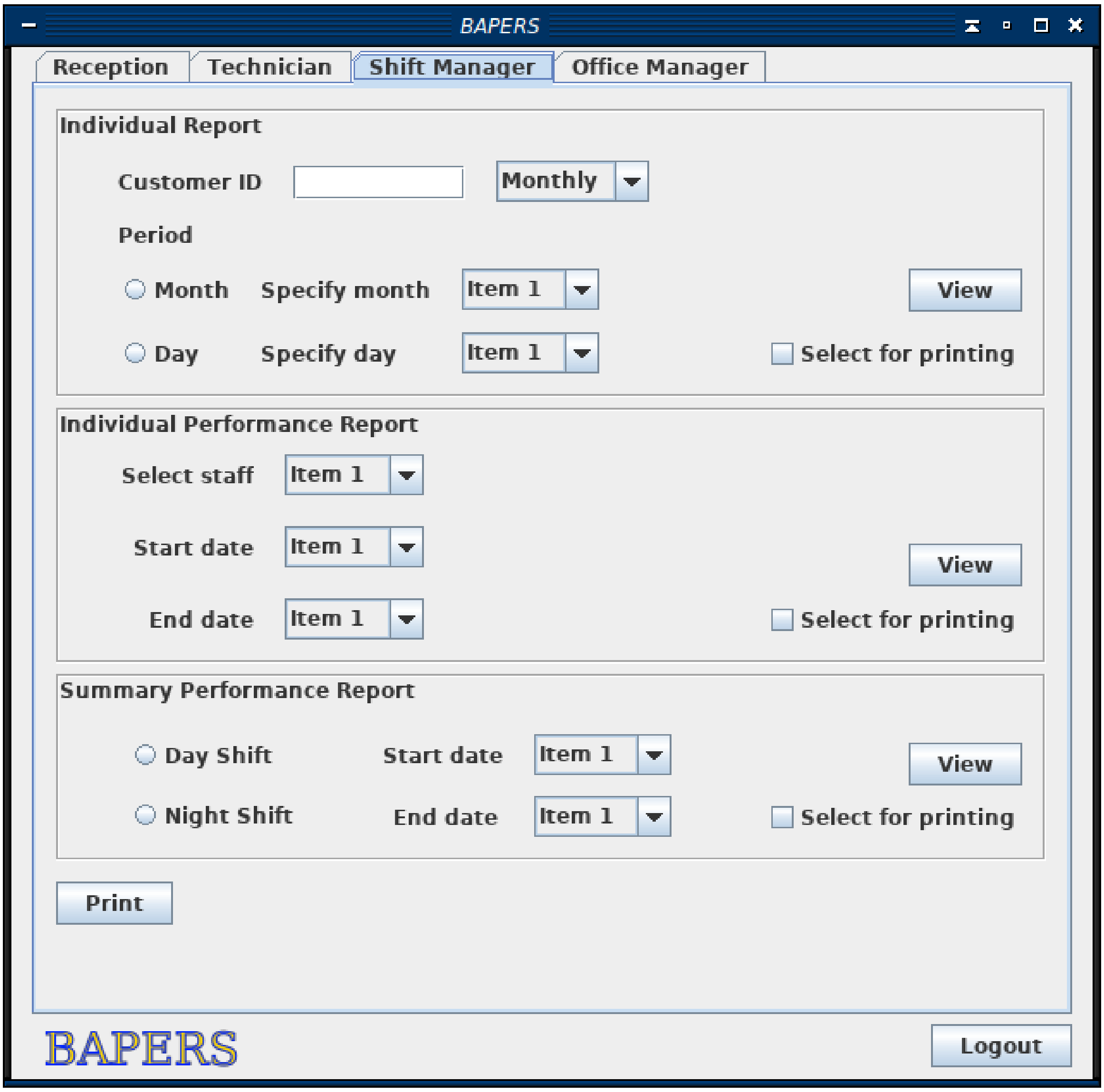
BAPERS

Logout

Technician form



Shift Manager Form



BAPERS

Reception | Technician | Shift Manager | Office Manager

Individual Report

Customer ID [ ] Monthly ▾

Period

○ Month Specify month Item 1 ▾ View

○ Day Specify day Item 1 ▾ □ Select for printing

Individual Performance Report

Select staff Item 1 ▾

Start date Item 1 ▾ View

End date Item 1 ▾ □ Select for printing

Summary Performance Report

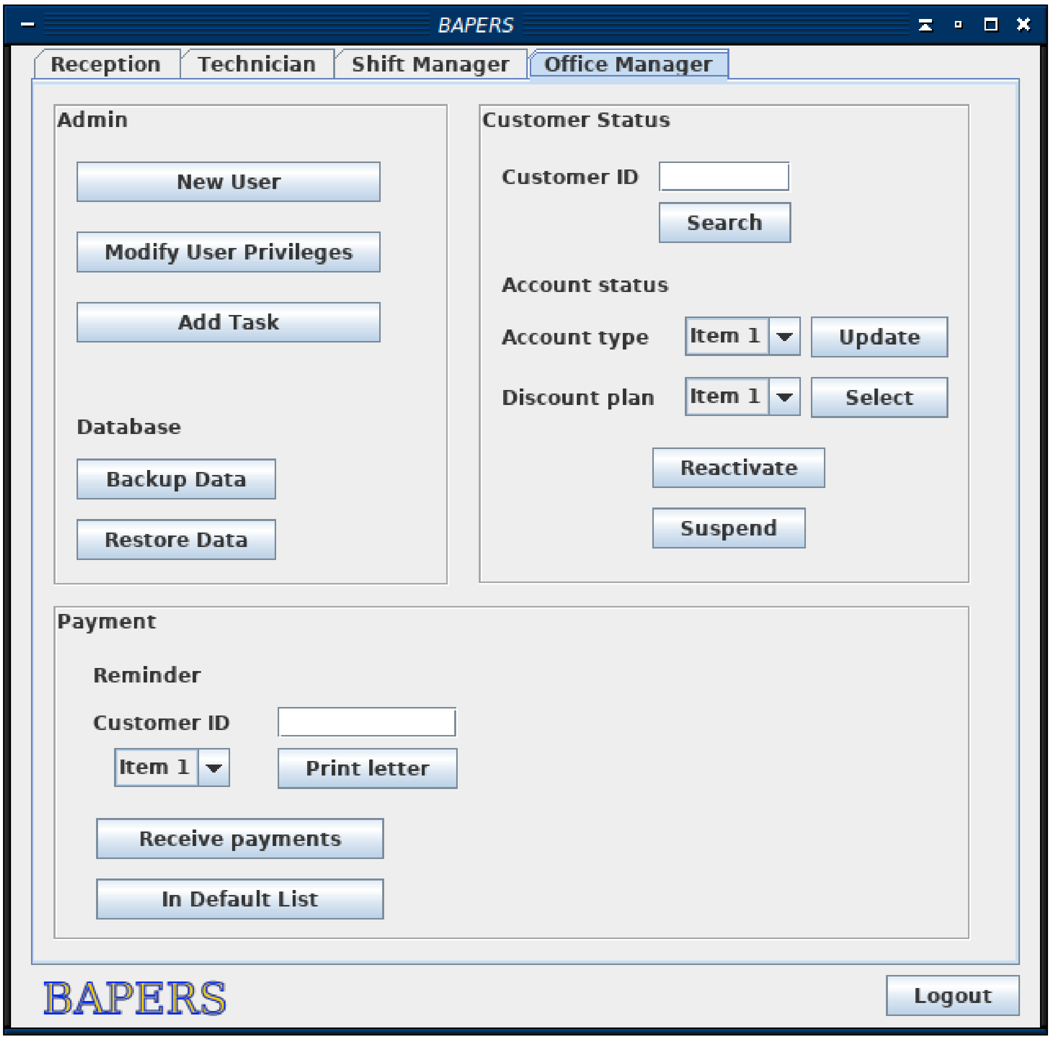
○ Day Shift Start date Item 1 ▾ View

○ Night Shift End date Item 1 ▾ □ Select for printing

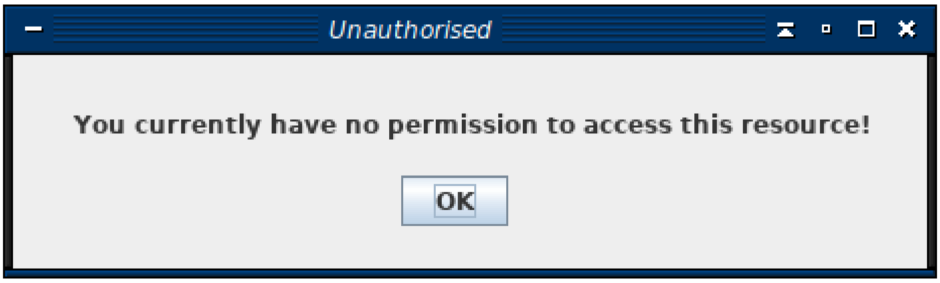
Print

BAPERS Logout

Office Manager

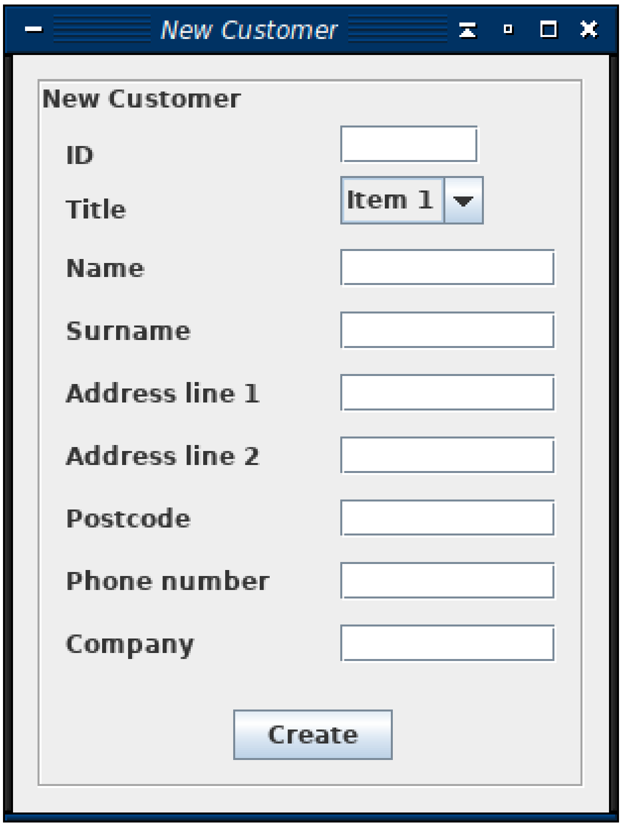


Unauthorised Form


Unauthorised

You currently have no permission to access this resource!

New Customer Form


New Customer

New Customer

ID

Title Item 1

Name

Surname

Address line 1

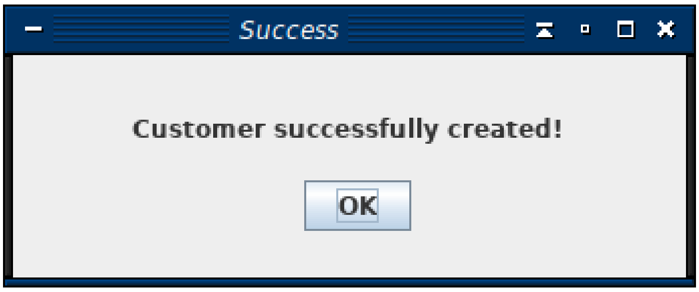
Address line 2

Postcode

Phone number

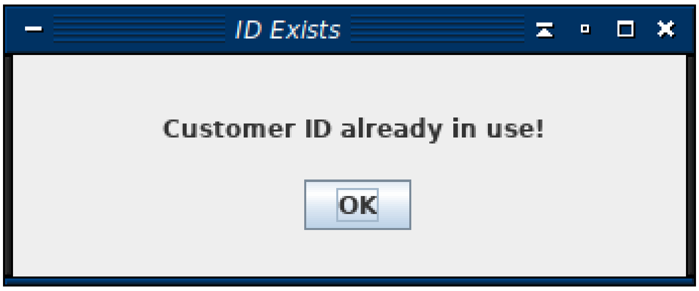
Company

Customer Creation Alert Form


Success

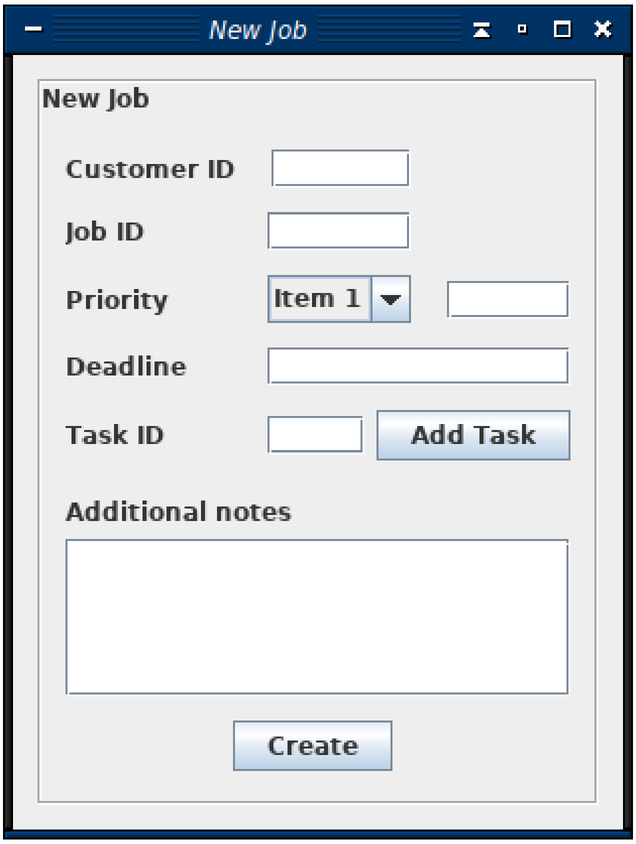
Customer successfully created!

Customer ID Exists form

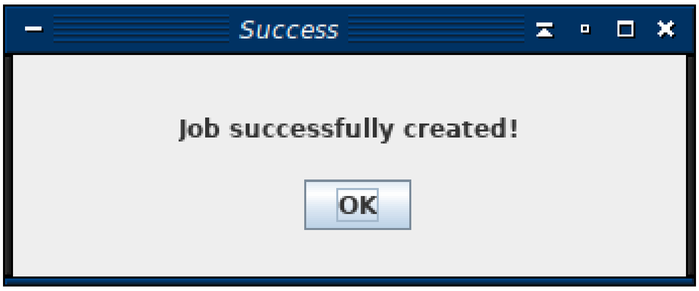


Customer ID already in use!

New job form

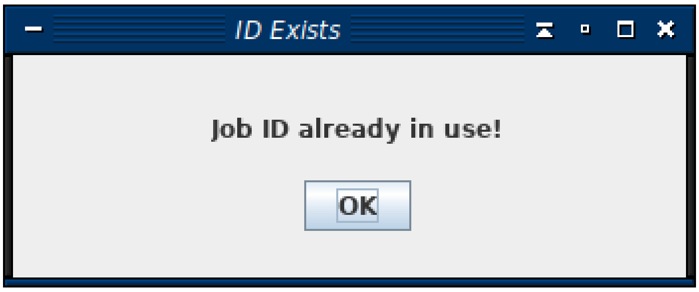


Job created form



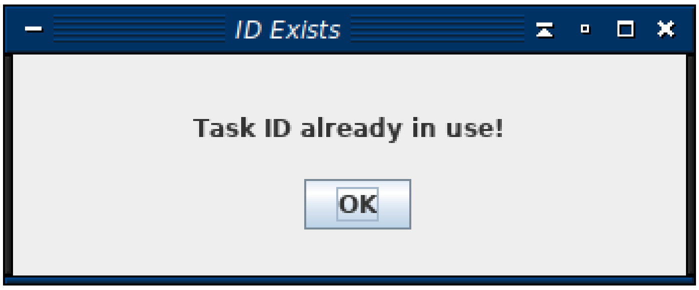
Job successfully created!

Job ID Exists Form



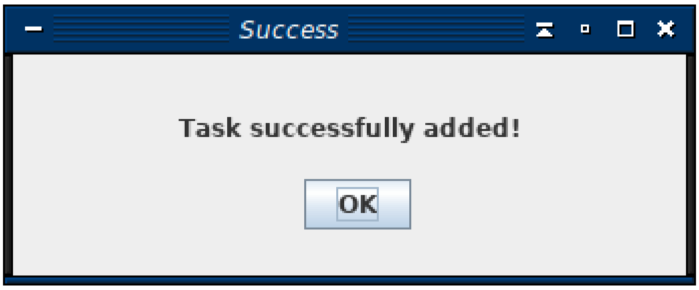
Job ID already in use!

Task ID Exist Form



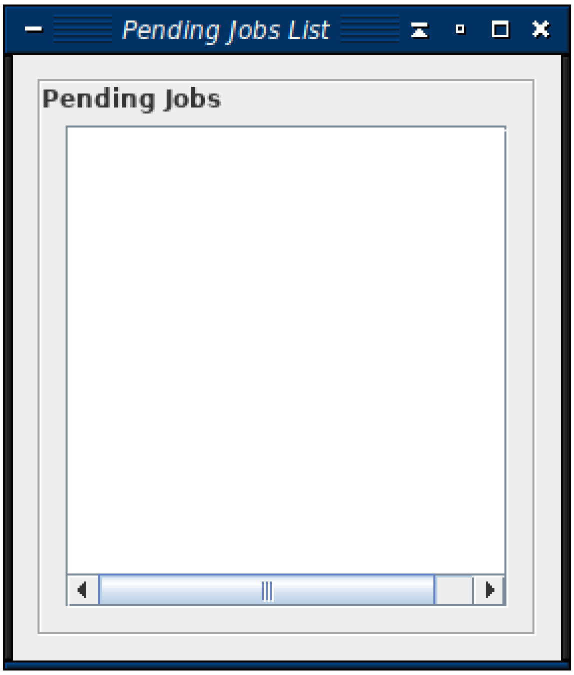
Task ID already in use!

Task Created Form

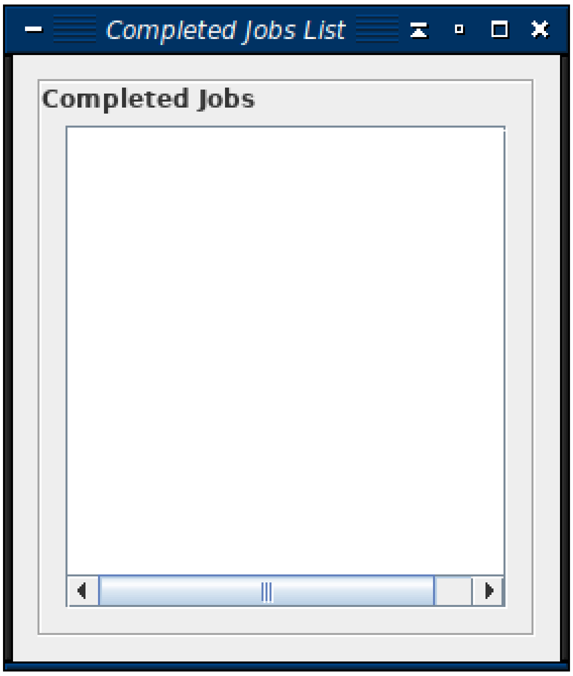


Task successfully added!

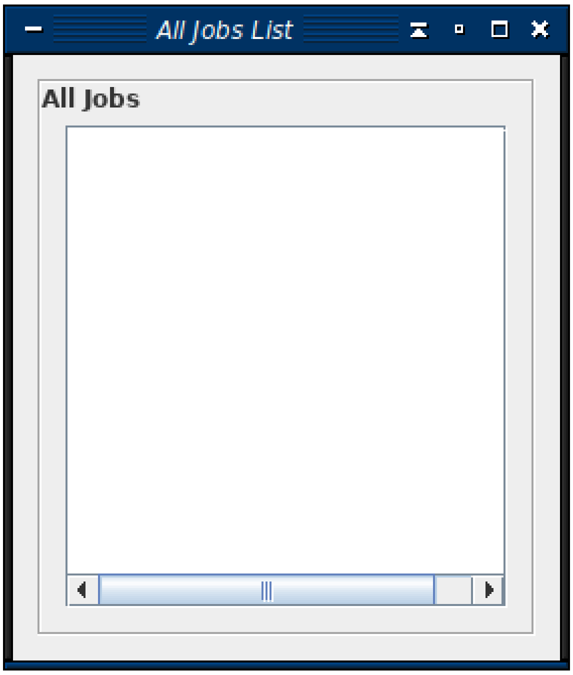
Pending Jobs List Form



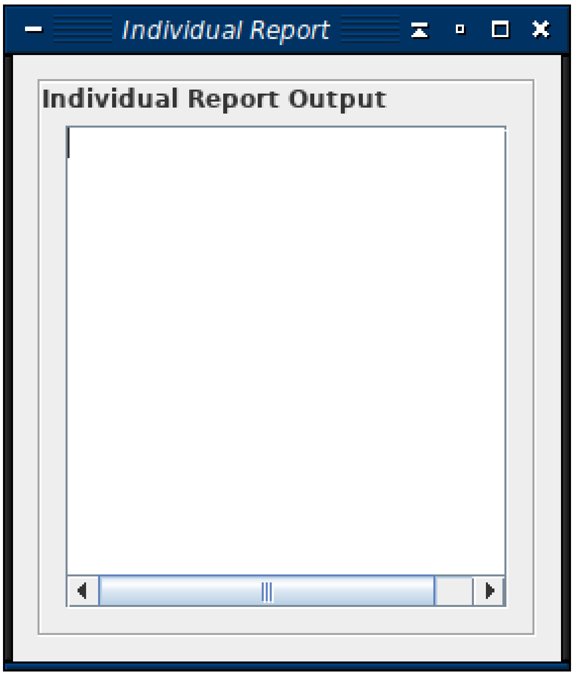
Completed Jobs List Form



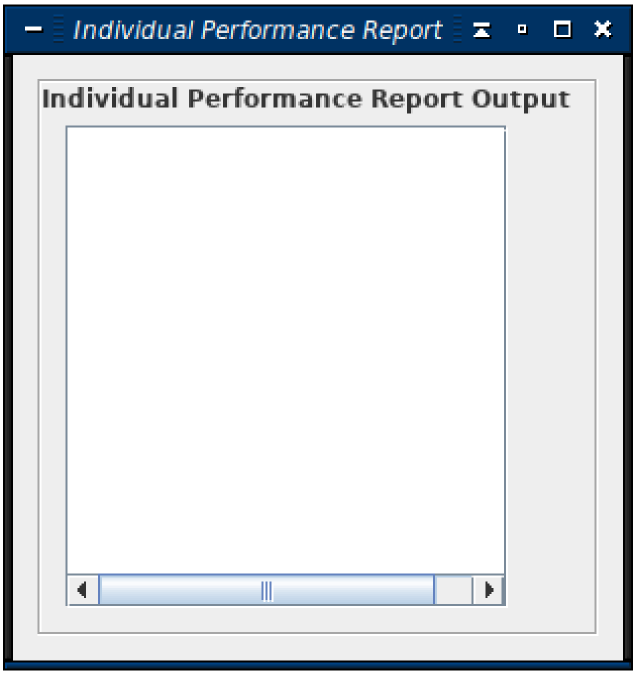
All Jobs List Form



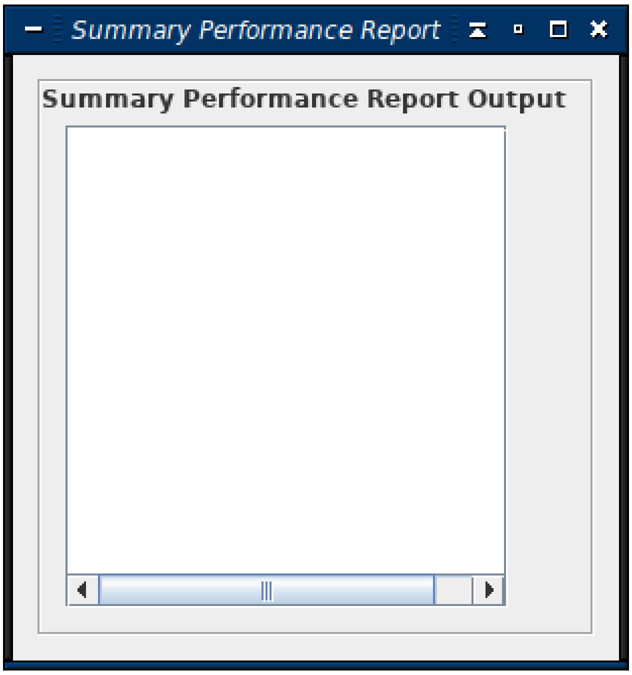
Individual report Output form



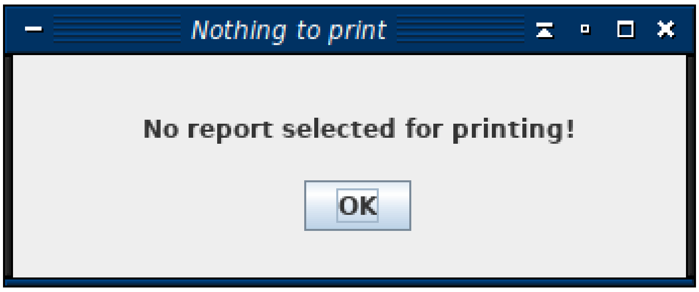
Individual Performance Report Output form


Individual Performance Report
Individual Performance Report Output

Summary Performance Report Output form

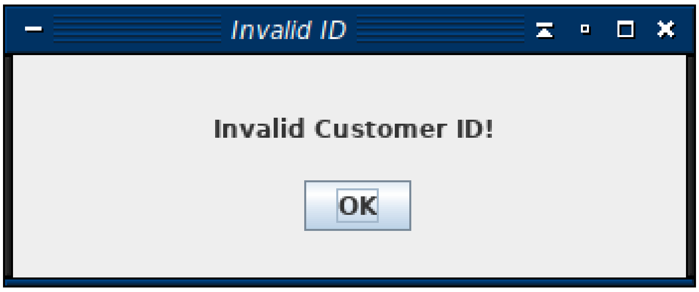


No Printing Target



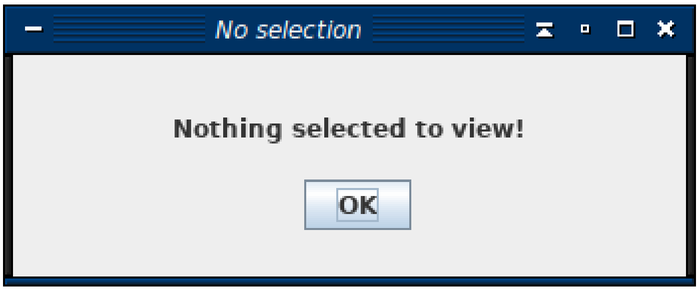
No report selected for printing!

Invalid Customer ID



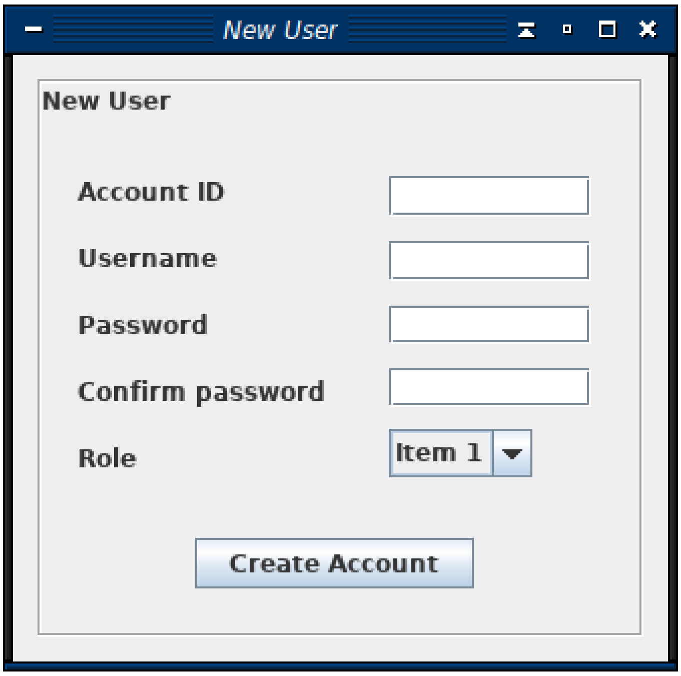
Invalid Customer ID!

No Selection



Nothing selected to view!

New User



New User

Account ID

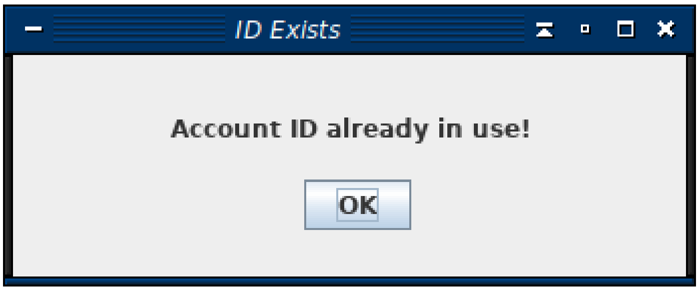
Username

Password

Confirm password

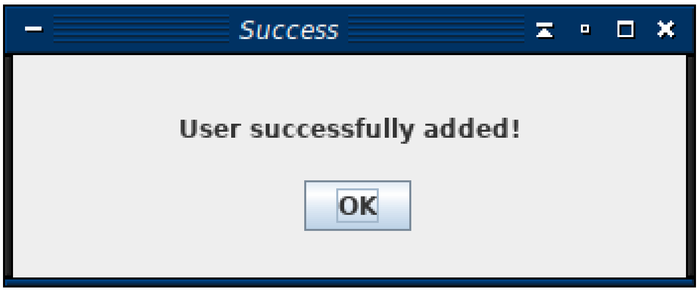
Role: Item 1

Account ID Exists Form



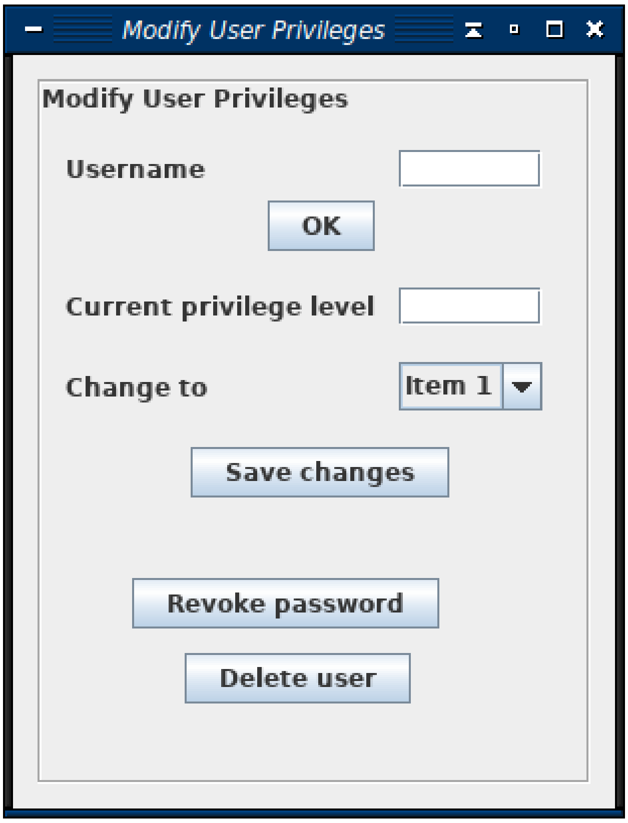
Account ID already in use!

User created form



User successfully added!

Modify User Privileges



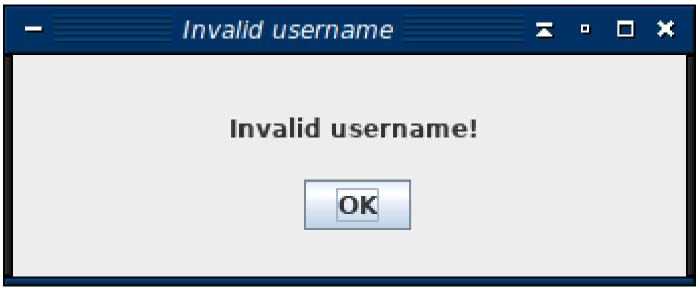
Modify User Privileges

Username

Current privilege level

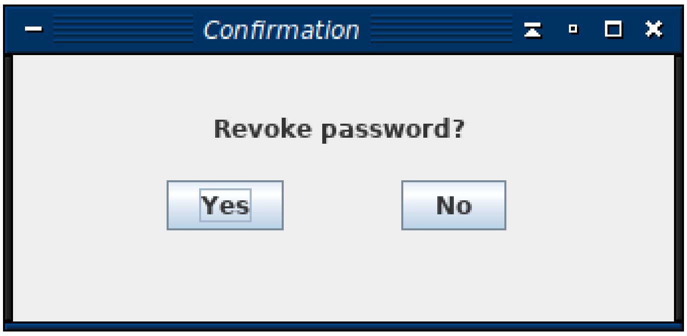
Change to Item 1

Invalid Username



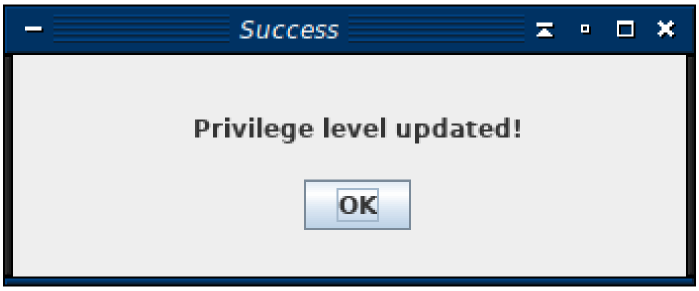
Invalid username!

Revoke password form


Confirmation

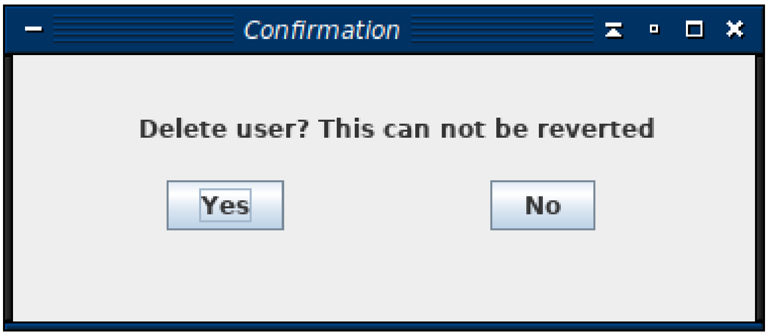
Revoke password?

Privilege Level Updated


Success

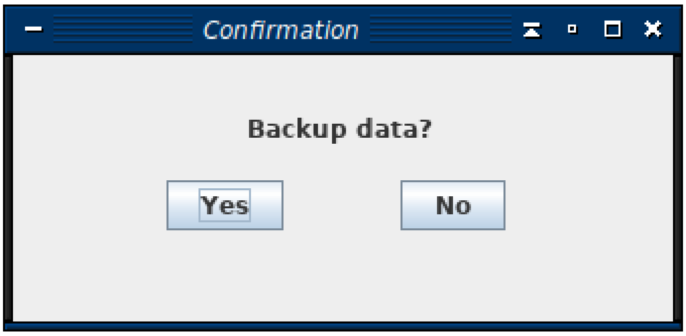
Privilege level updated!

Delete User Confirmation


Confirmation

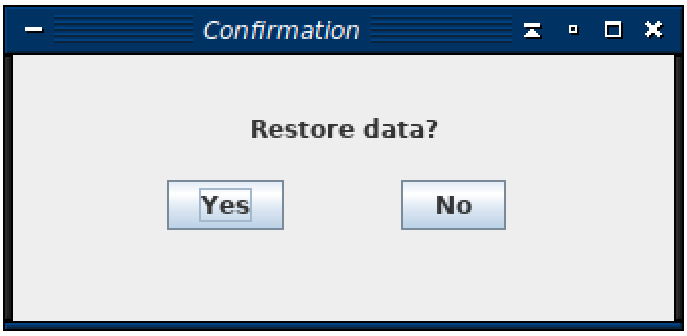
Delete user? This can not be reverted

Backup Data


Confirmation

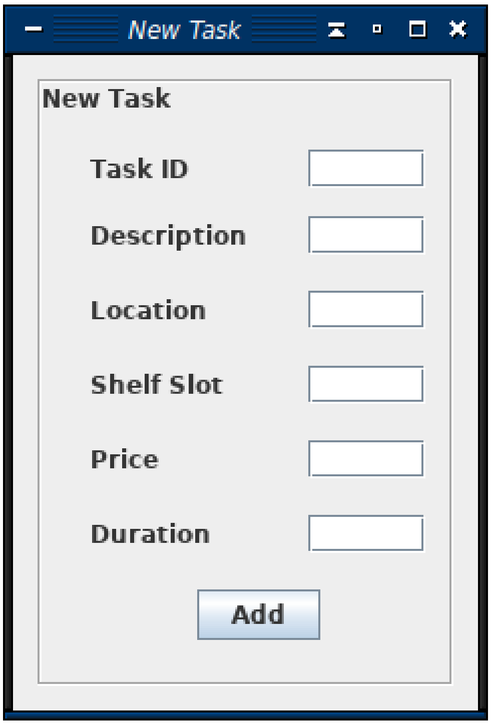
Backup data?

Restore Data form

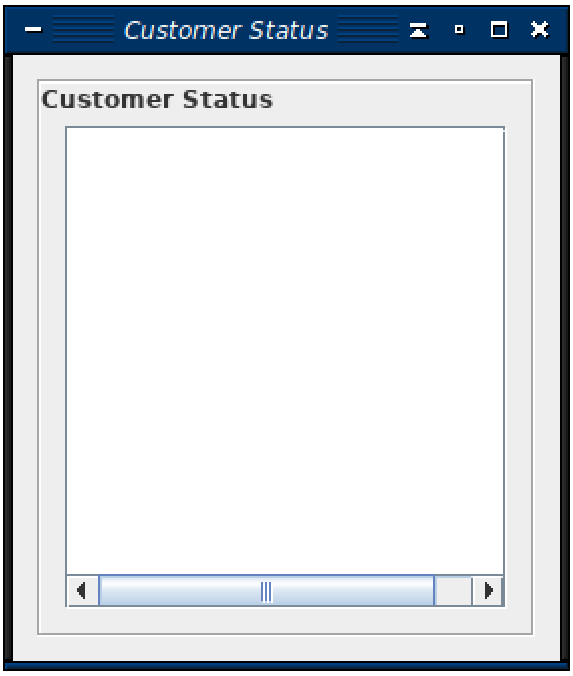


Restore data?

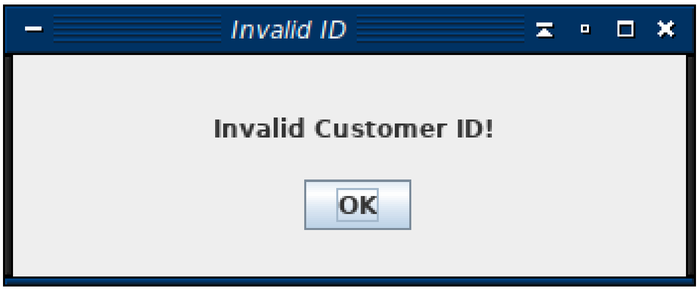
New Task Form



Customer Status form

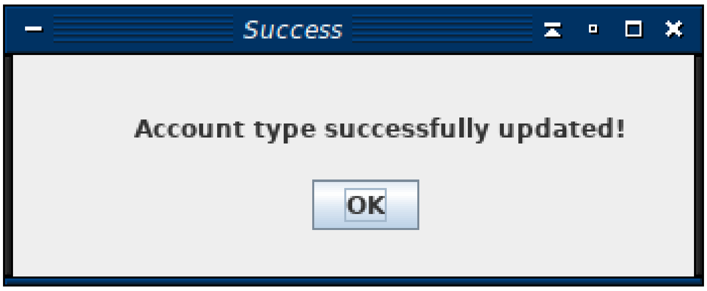


Invalid Customer ID Form



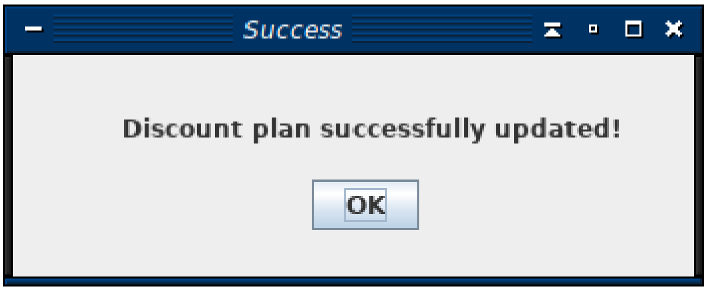
Invalid Customer ID!

Account Type Updated



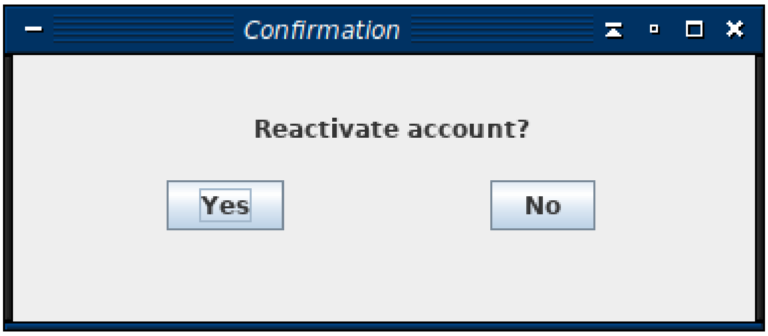
Account type successfully updated!

Discount Plan Updated



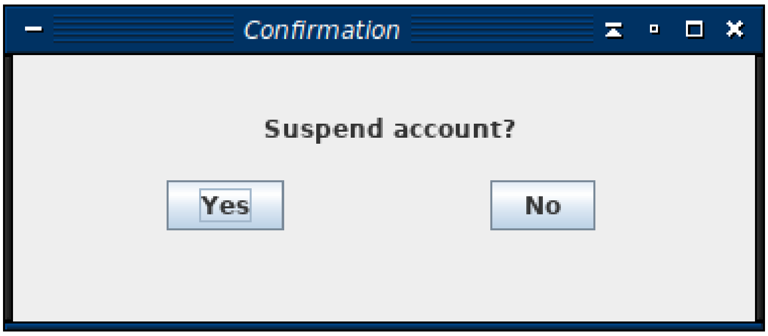
Discount plan successfully updated!

Reactivate Account



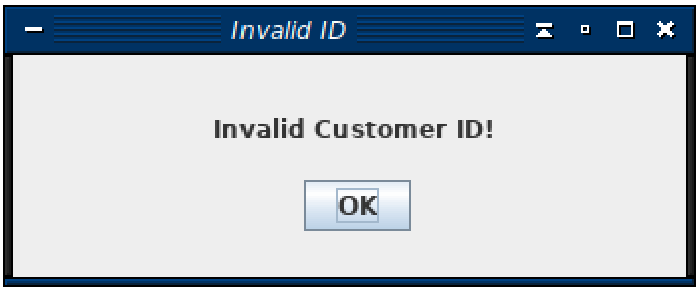
Reactivate account?

Suspend Account



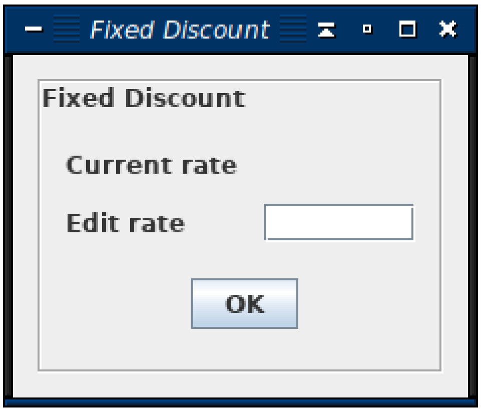
Suspend account?

Invalid Customer ID

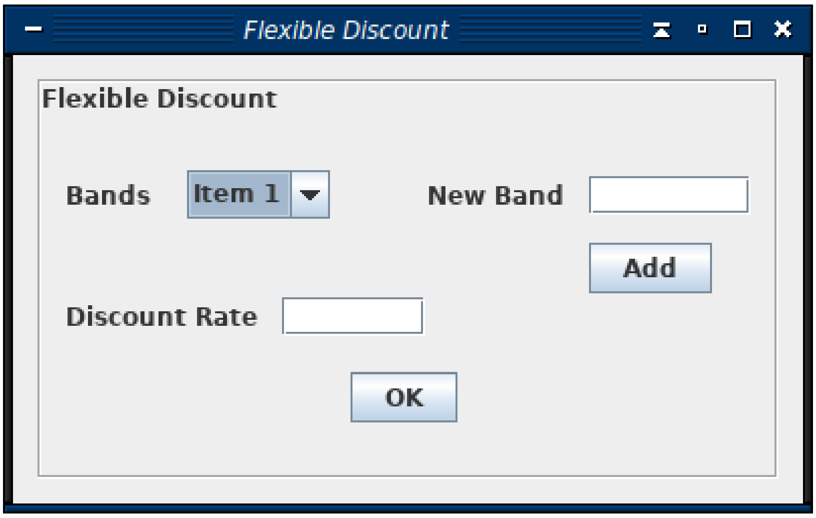


Invalid Customer ID!

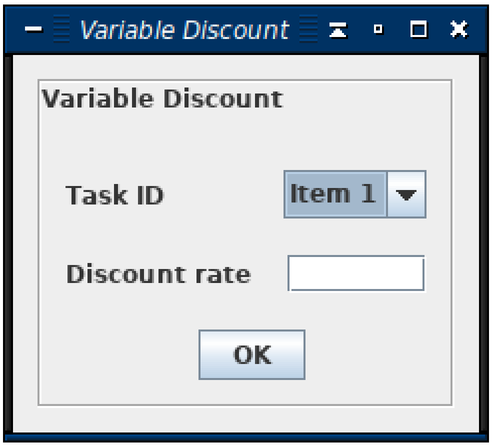
Fixed Discount



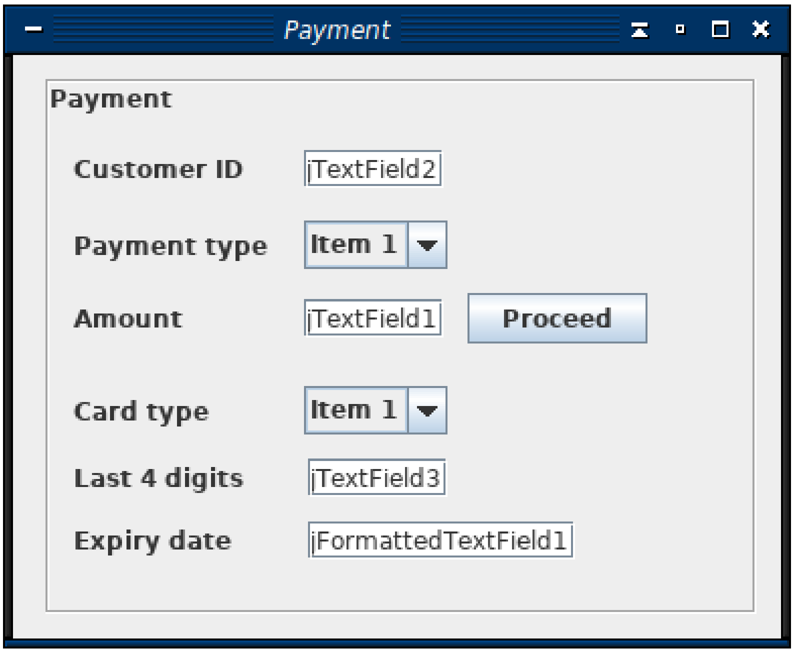
Flexible Discount



Variable Discount



Payment Form



Payment

Customer ID jTextField2

Payment type Item 1

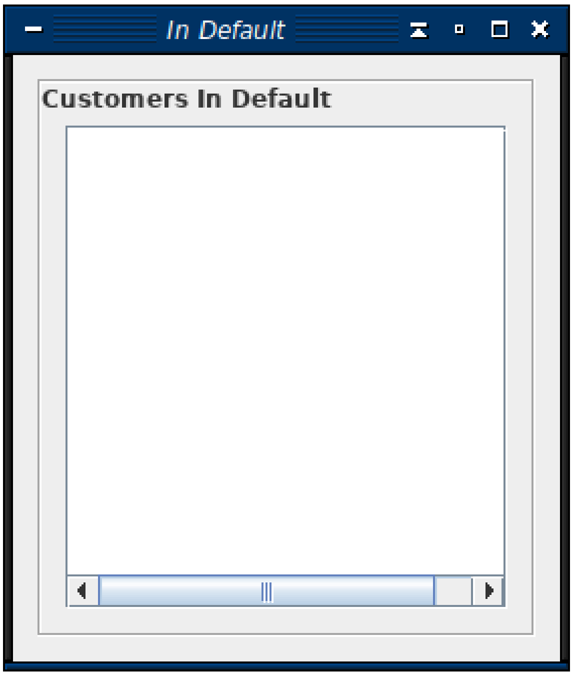
Amount jTextField1

Card type Item 1

Last 4 digits jTextField3

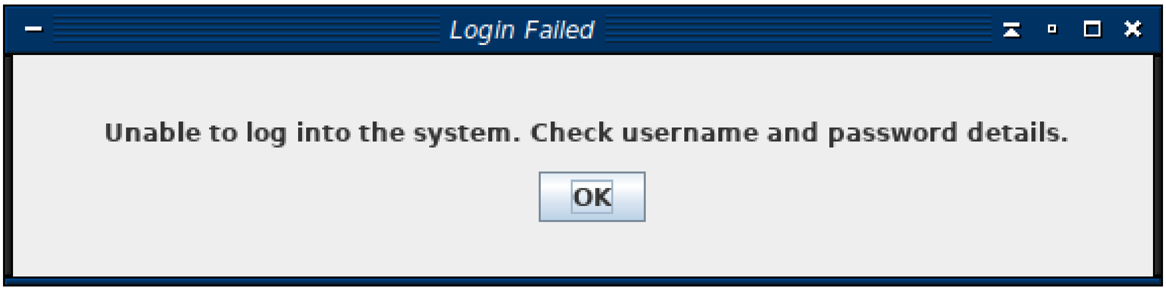
Expiry date jFormattedTextField1

In Default List Form



Customers In Default

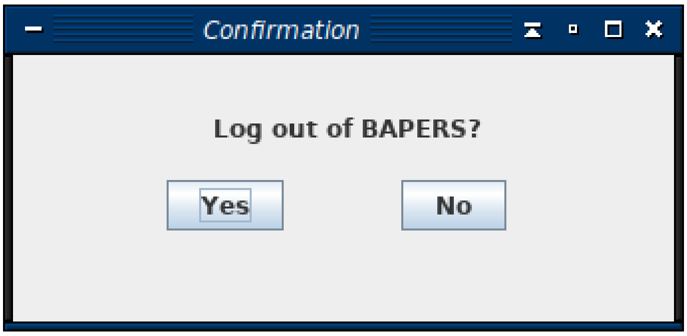
Login Failure



Login Failed

Unable to log into the system. Check username and password details.

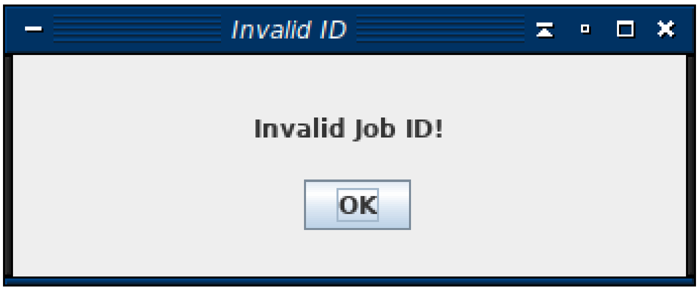
Log Out Confirmation



Confirmation

Log out of BAPERS?

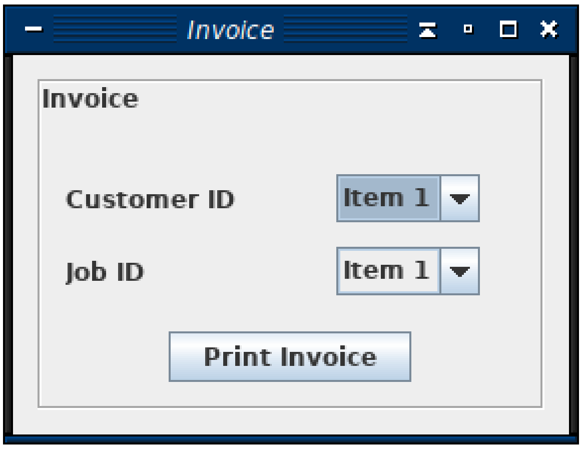
Invalid Job ID



Invalid ID

Invalid Job ID!

Invoice form



Invoice

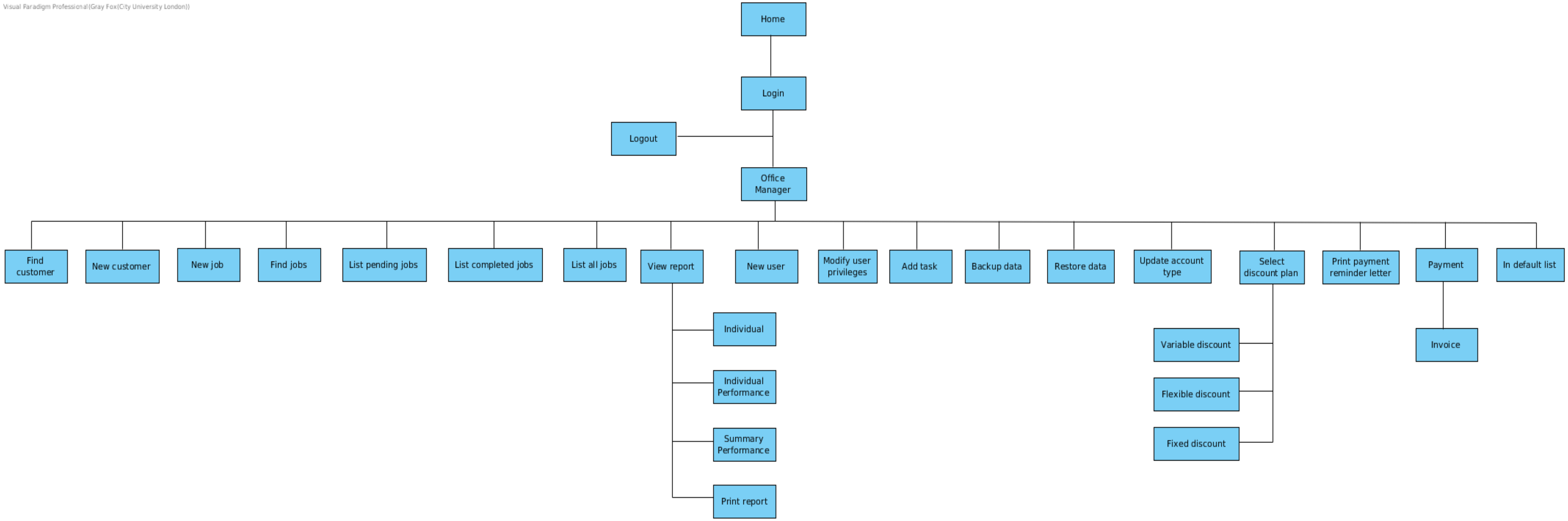
Invoice

Customer ID: Item 1

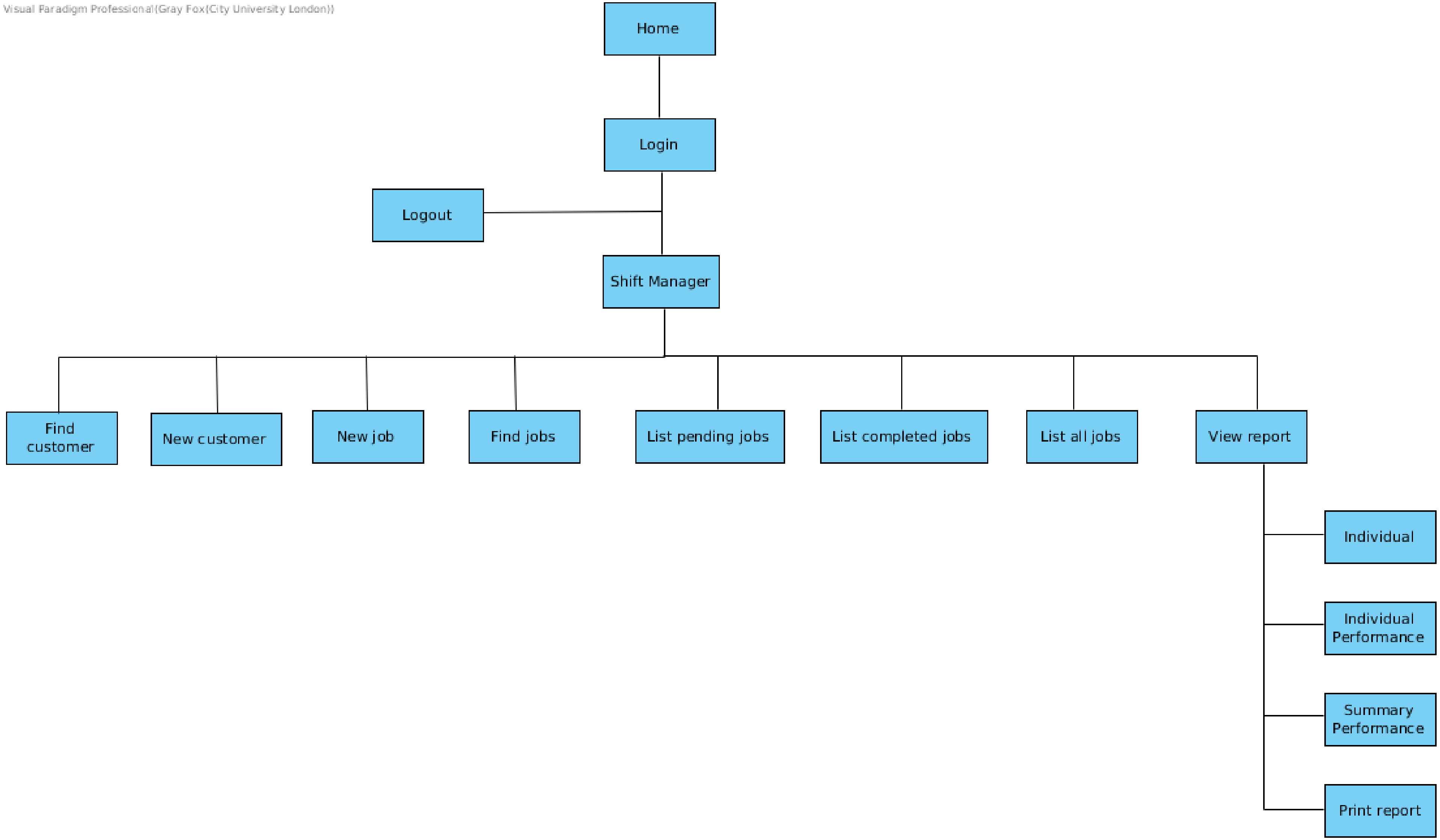
Job ID: Item 1

GUI Site Maps

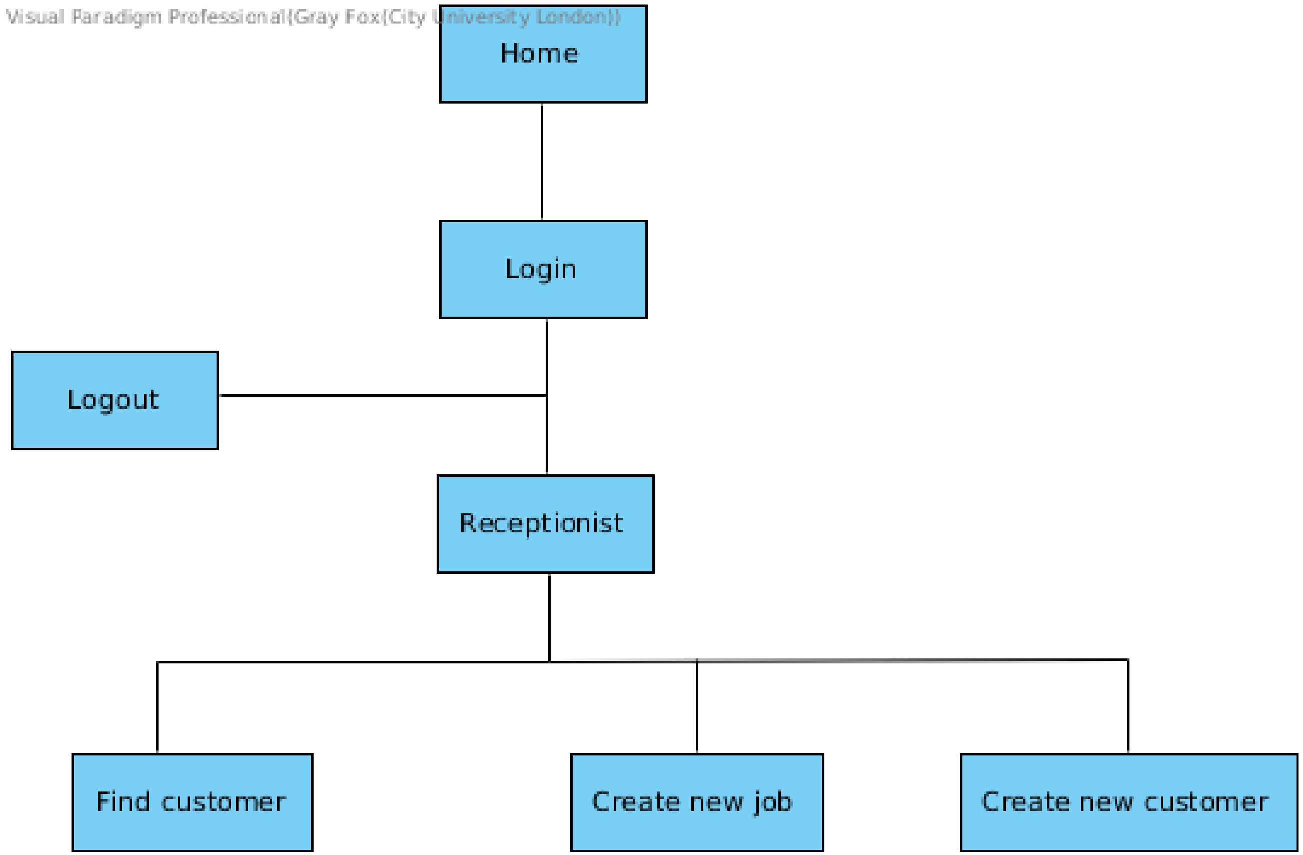
Office manager site map



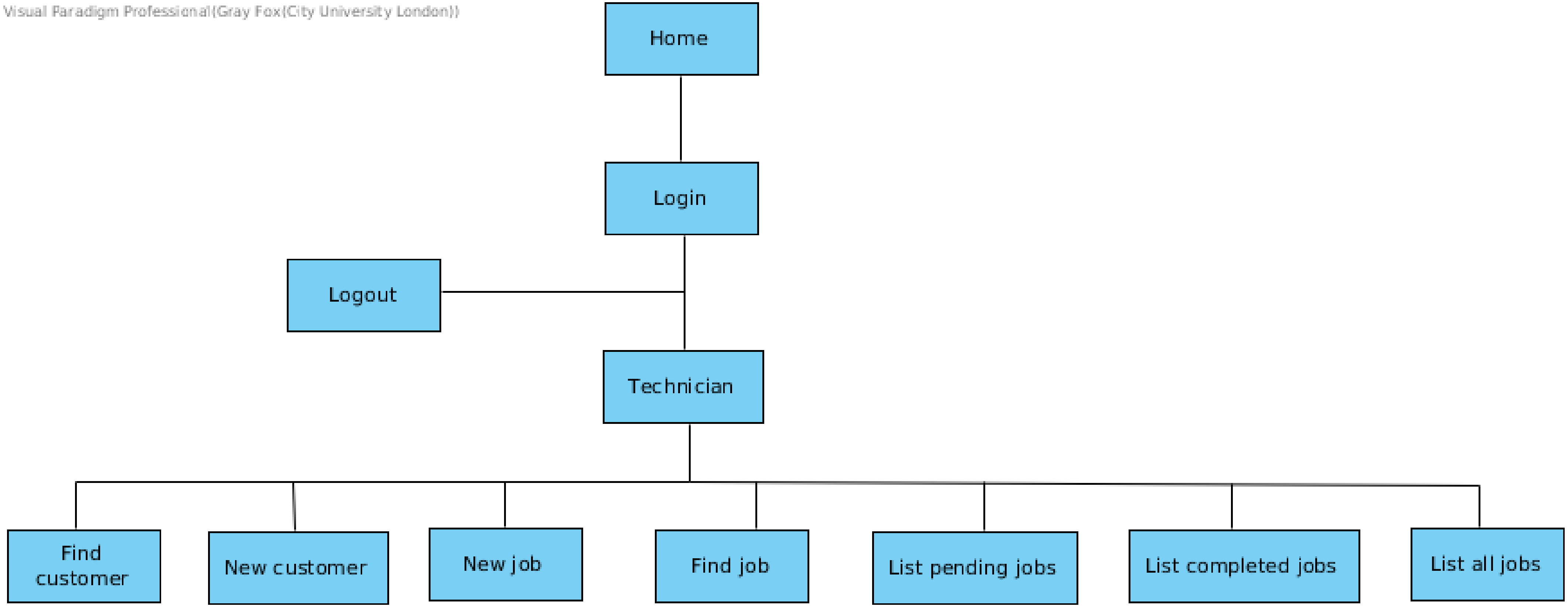
Shift manager site map



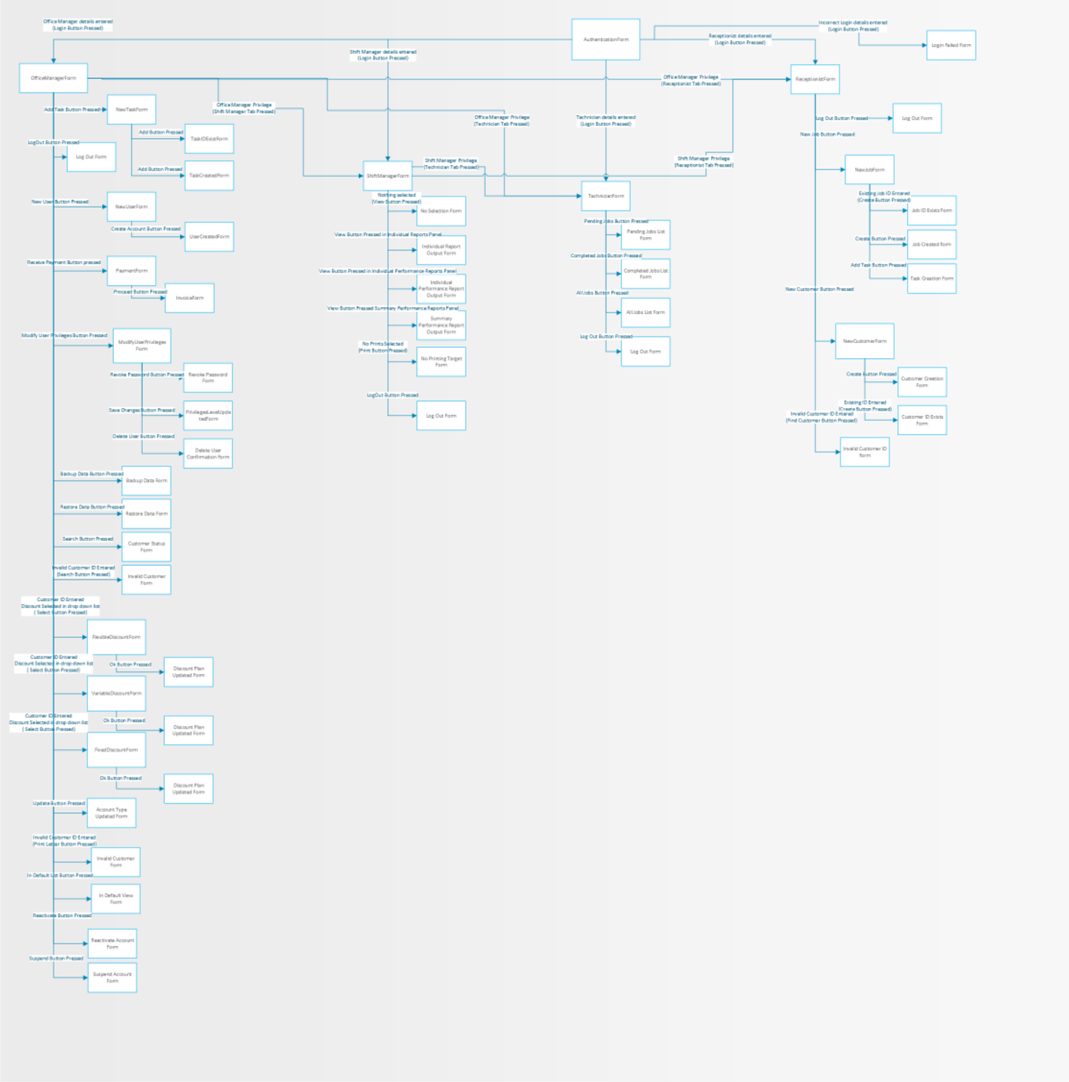
Receptionist Site map

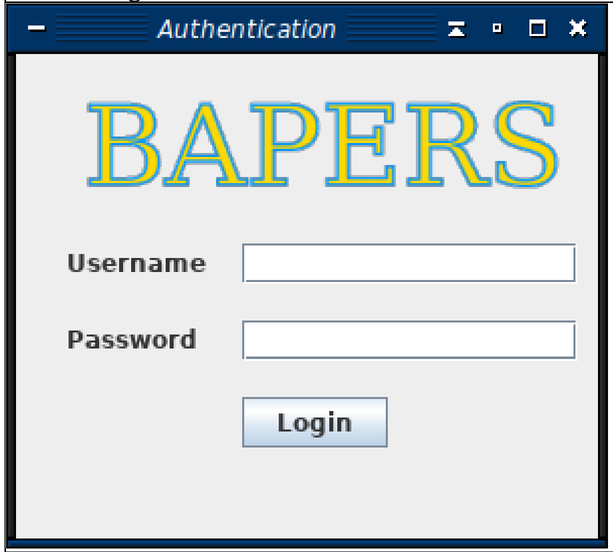
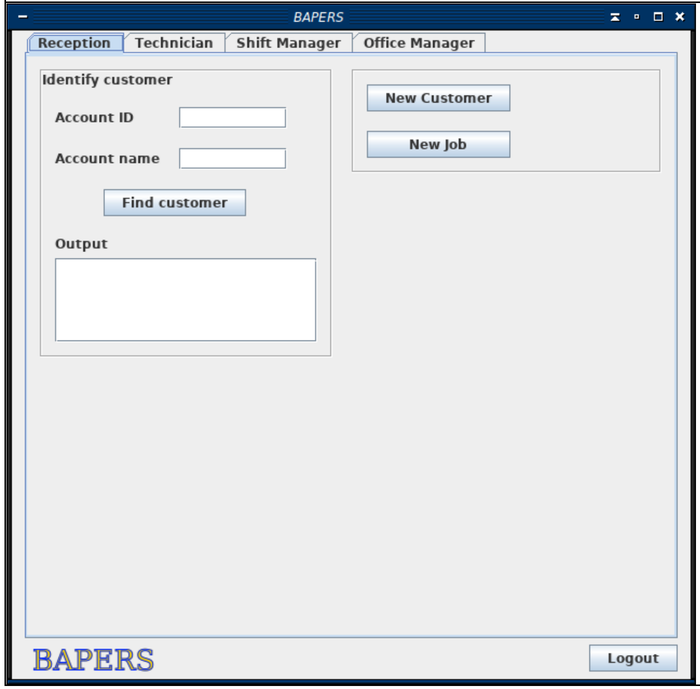


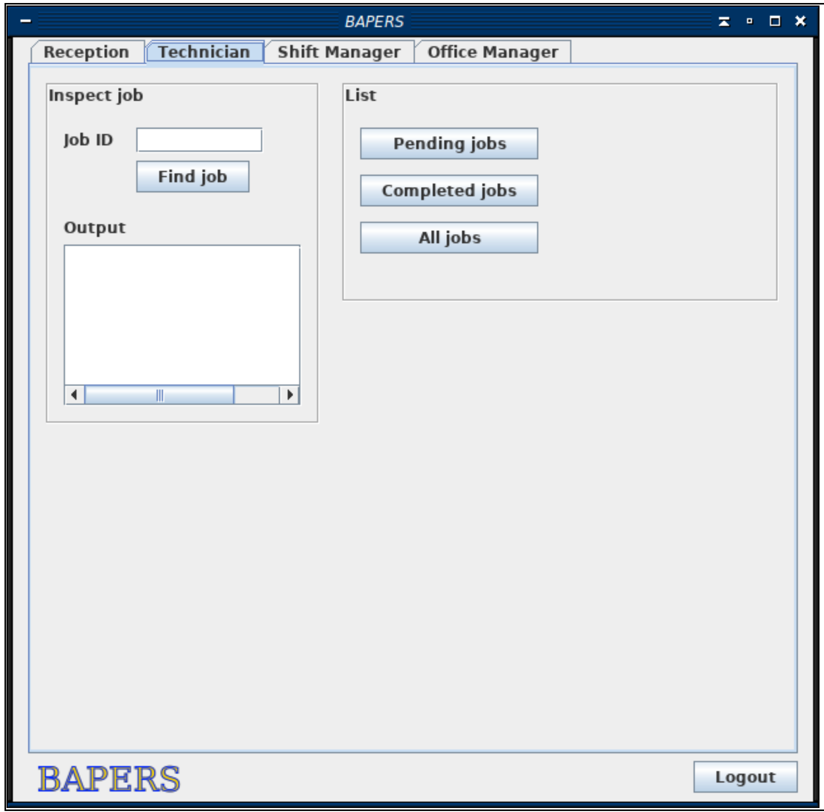
Technician Site map

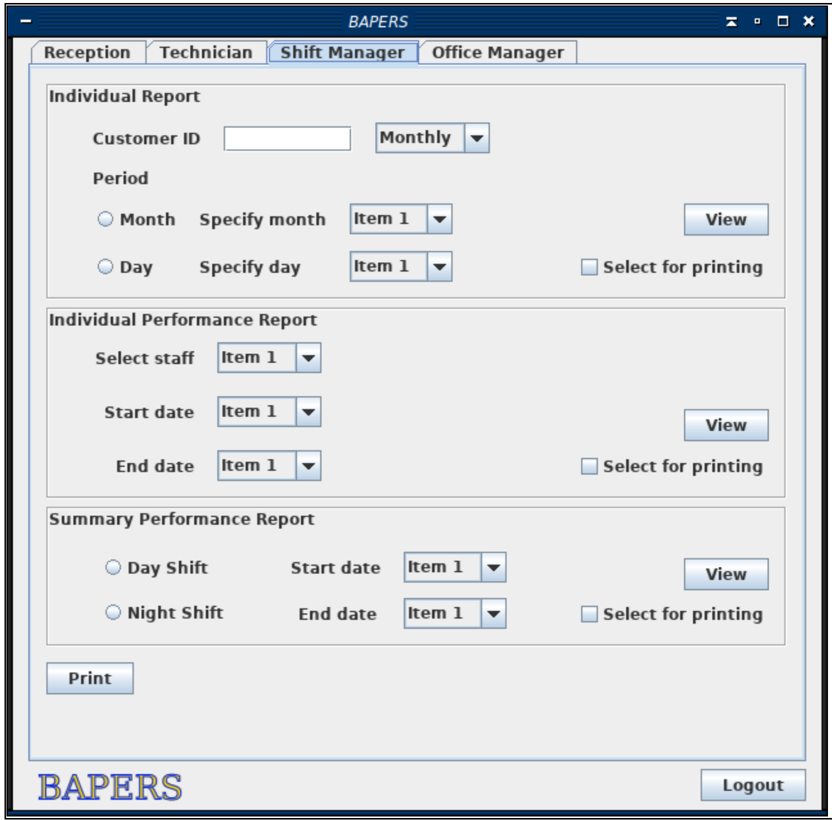


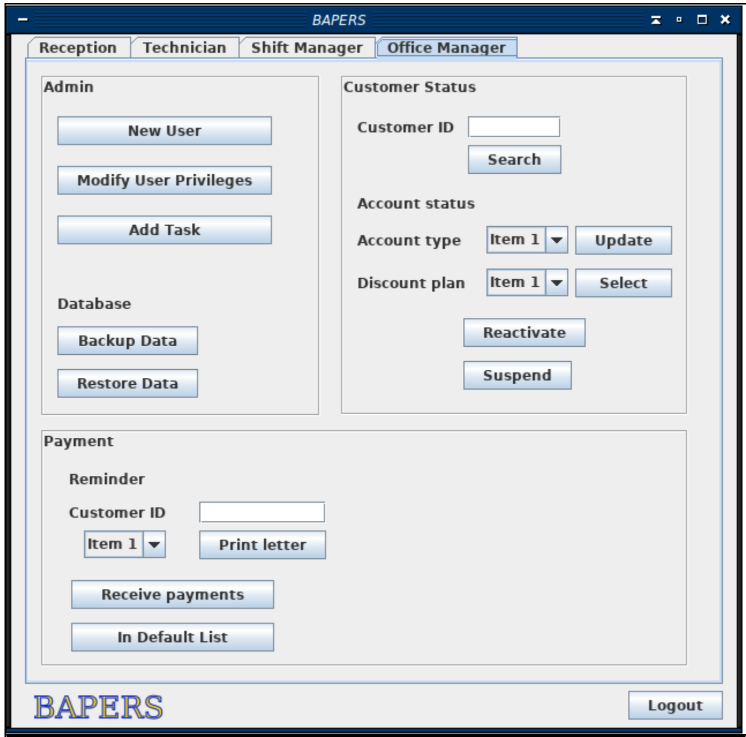
GUI Navigation and Mapping



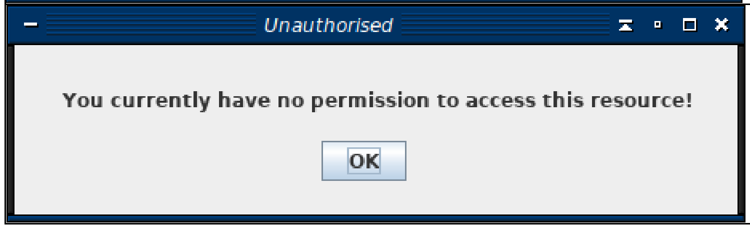
GUI Design	Description	Mapping
	This is the login form for the BAPERS user interface The user enters their username and password and press the Login button If login is successful, the user goes to the main BAPERS interface. If login details are incorrect, the Login Failed popup appears The form shown here is the Reception tab of the main interface.	AuthenticationForm
 BAPERS Reception Technician Shift Manager Office Manager Identify customer Account ID Account name Find customer Output New Customer New Job BAPERS Logout	This tool allows the user to search for a customer by filling in the two boxes and clicking Find customer Afterwards, the customer's details are outputted into this box	ReceptionForm

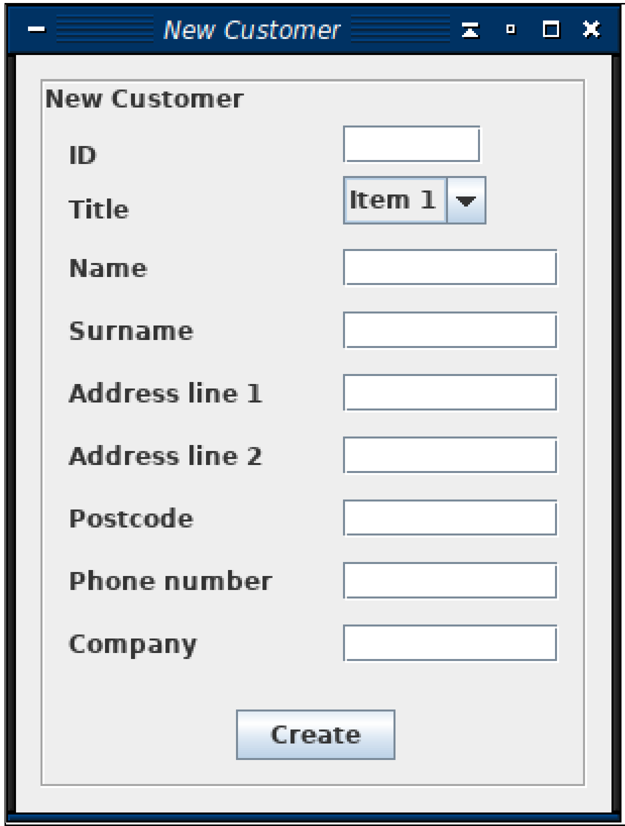
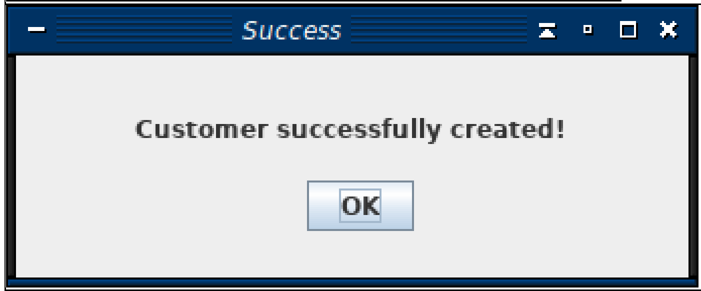
 BAPERS Reception Technician Shift Manager Office Manager Inspect job Job ID Find job Output List Pending jobs Completed jobs All jobs BAPERS Logout		This tool allows the user to see details of a Job by inputting the ID and clicking Find Job The details (if correct) are printed into this box. If the inputted Job ID is invalid, the Invalid Job ID popup appears These three buttons output a list as a popup – Pending Jobs, Completed Jobs, and All Jobs.	TechnicianForm

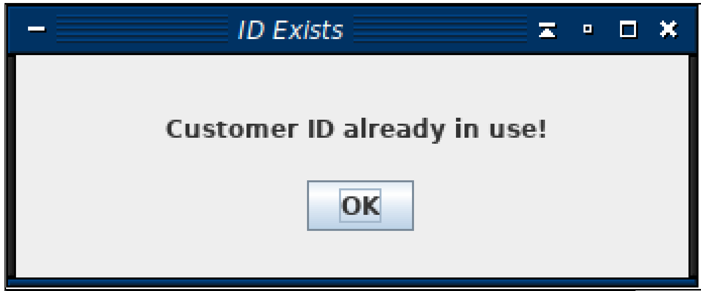
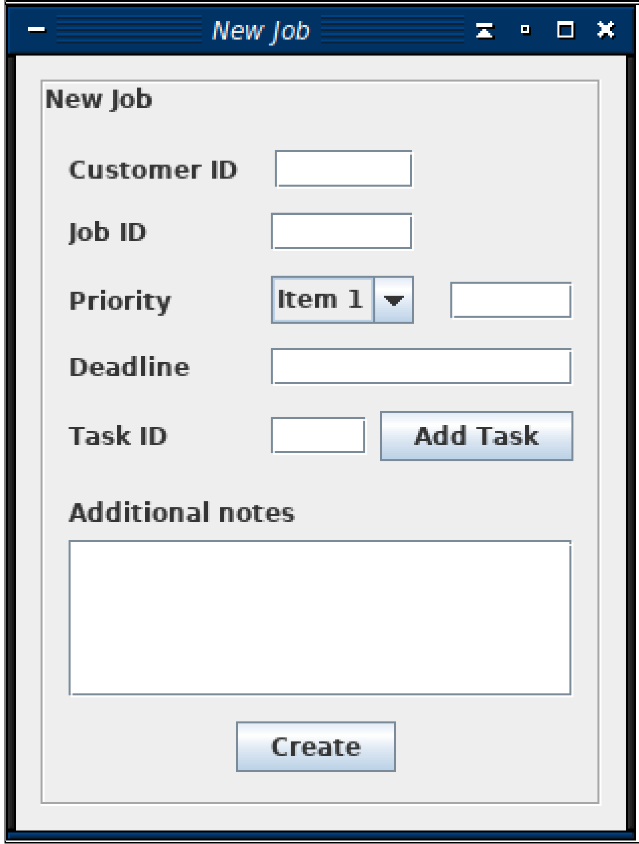
 BAPERS Reception Technician Shift Manager Office Manager Individual Report Customer ID Monthly Period ○ Month Specify month Item 1 View ○ Day Specify day Item 1 □ Select for printing Individual Performance Report Select staff Item 1 Start date Item 1 View End date Item 1 □ Select for printing Summary Performance Report ○ Day Shift Start date Item 1 View ○ Night Shift End date Item 1 □ Select for printing Print BAPERS Logout	These are administrative options. These are database options - the first button is for backing up data and the second for restoring. Both have confirmation popups	ShiftManagerForm

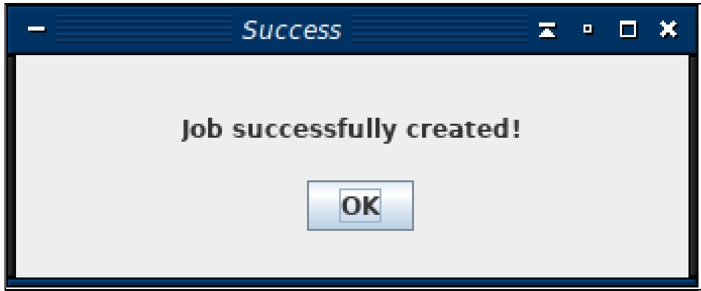
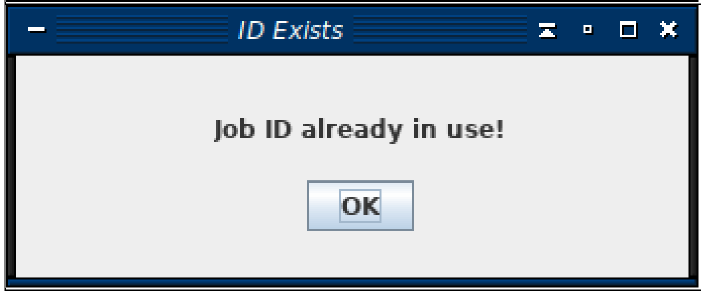
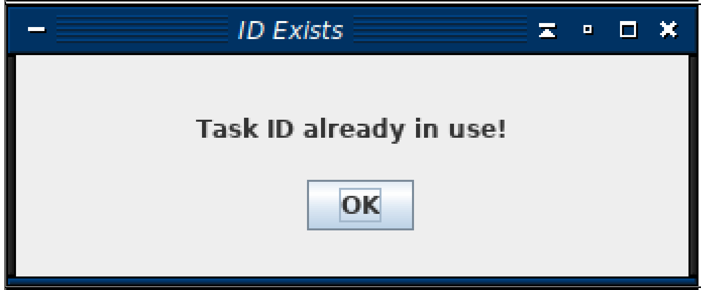
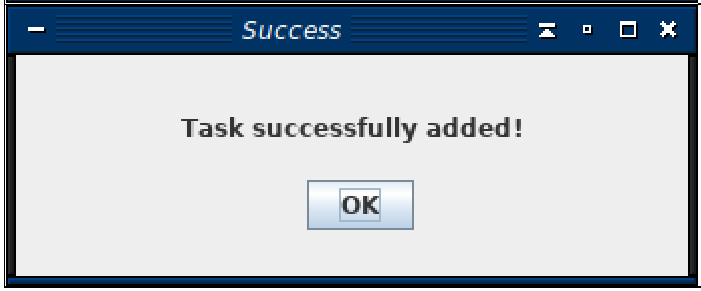
<table>
<tr>
<td>

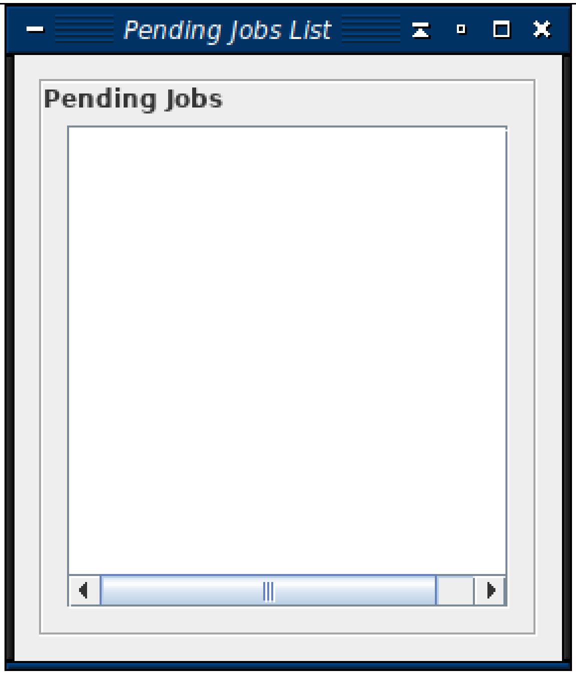
BAPERS
Reception
Technician
Shift Manager
Office Manager
Admin
New User
Modify User Privileges
Add Task
Database
Backup Data
Restore Data
Customer Status
Customer ID
Search
Account status
Account type
Item 1
Update
Discount plan
Item 1
Select
Reactivate
Suspend
Payment
Reminder
Customer ID
Item 1
Print letter
Receive payments
In Default List
BAPERS
Logout

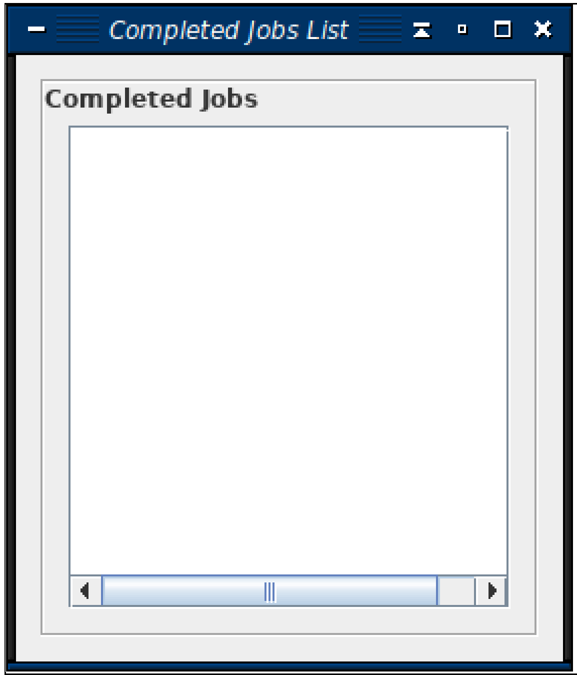
</td>
<td>Entering a valid customer ID and clicking search will output the Customer Status popup
These options are for customer accounts. The account type can be changed, and a discount plan can be selected. In addition, the selected account can be reactivated or suspended.</td>
<td>OfficeManagerForm</td>
</tr>
<tr>
<td>
You currently have no permission to access this resource!
 </td>
<td>User does not has authorised access</td>
<td>MessageForm</td>
</tr>
</table>

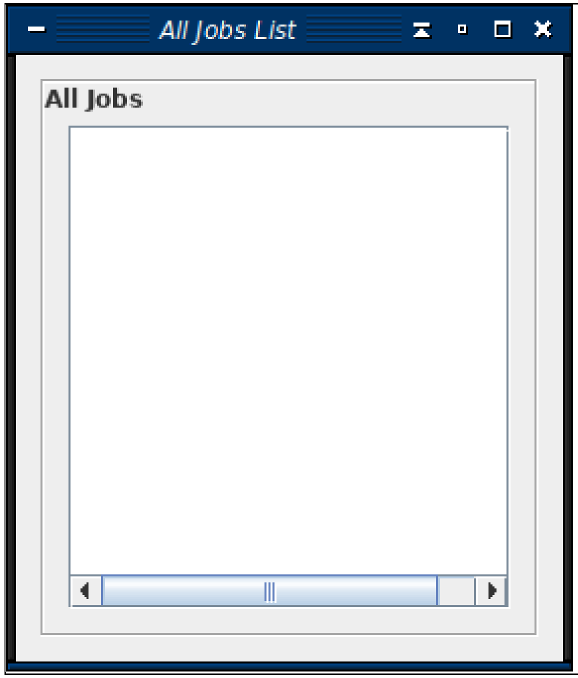
 New Customer New Customer ID Title Item 1 Name Surname Address line 1 Address line 2 Postcode Phone number Company Create		From within the Receptionist form, the user can create a new customer. These details have to be filled in, including name and address. When details have been inputted and are successful, the Create button has to be clicked, which will output a new popup.	NewCustomerForm
 Success Customer successfully created! OK		If all goes well after clicking the Create button, this popup message will appear saying that the customer has been successfully created.	MessageForm

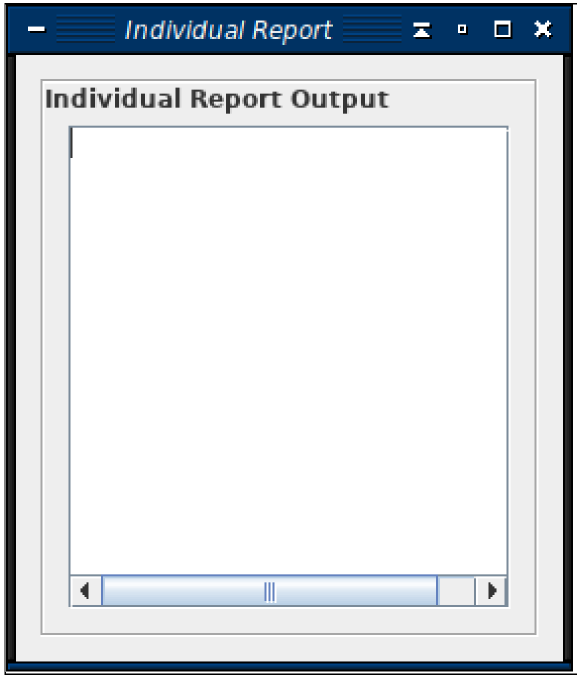
 ID Exists Customer ID already in use! OK	If the given ID for the new customer is already existing, this popup message appears. Clicking OK will return back to the New Customer form.	MessageForm
 New Job New Job Customer ID Job ID Priority Item 1 Deadline Task ID Add Task Additional notes Create	This form is for the Receptionist to create a new job. The customer and job details have to be inputted. The additional notes allows the user to input extra information about the particular job.	NewJobForm

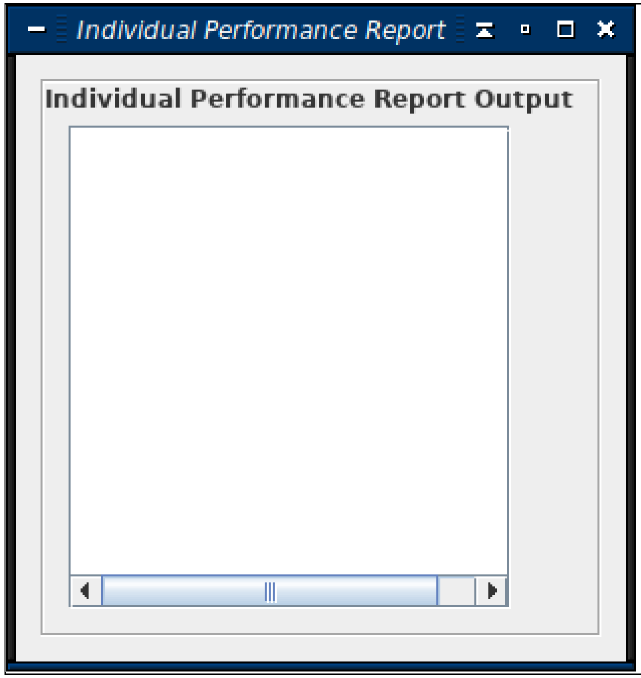
 Job successfully created!		Job has been created form	MessageForm
 Job ID already in use!		If the given Job ID already exists, this popup message shows up, and clicking OK returns the user to the New Job form.	MessageForm
 Task ID already in use!		If the given Task ID already exists, this popup message shows up, and clicking OK returns the user to the New Task form.	MessageForm
 Task successfully added!		This popups up when adding the task is successful	MessageForm

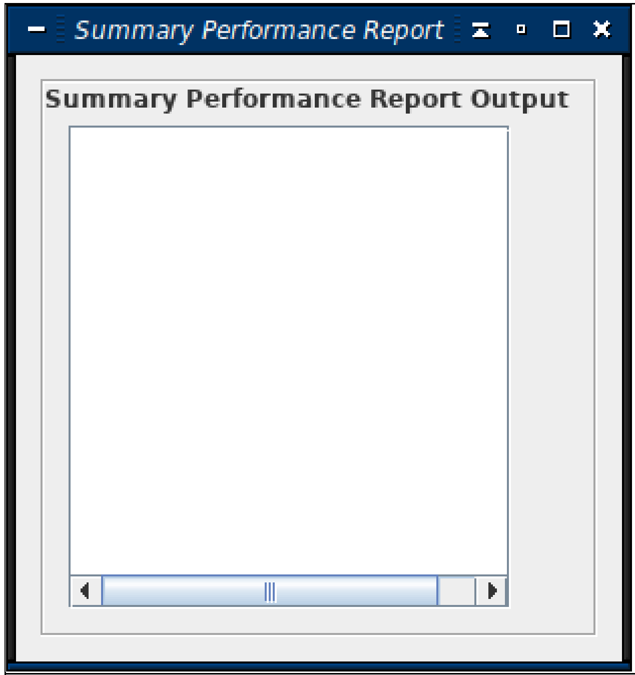
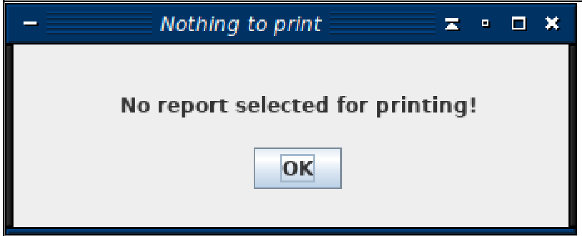
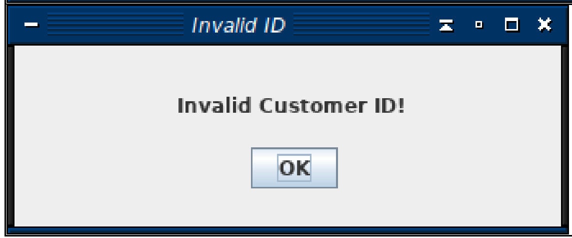
	This is the output list created when the Technician clicks on Pending Jobs.	JobOutputForm

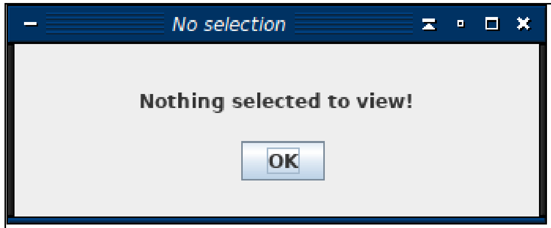
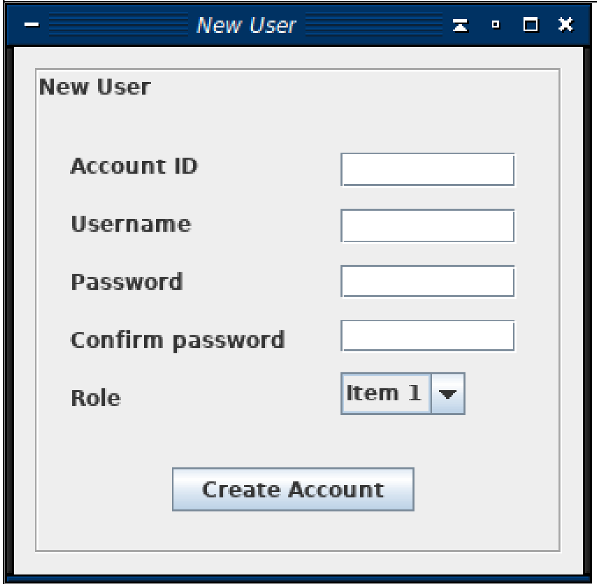
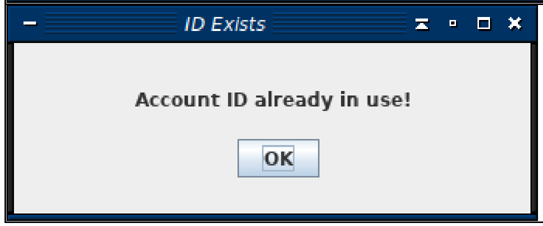
		This output list is outputted when clicking Completed Jobs in the Technician tab.	JobOutputForm

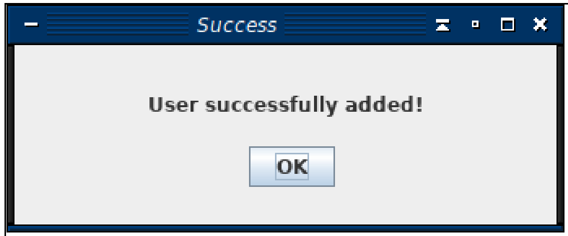
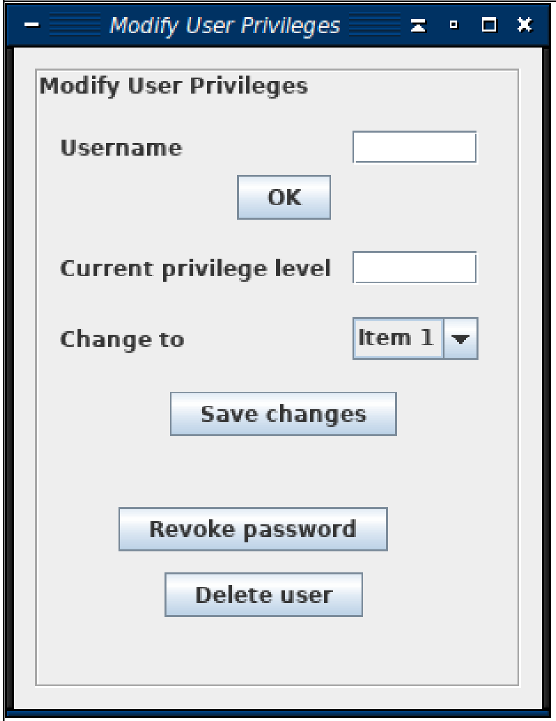
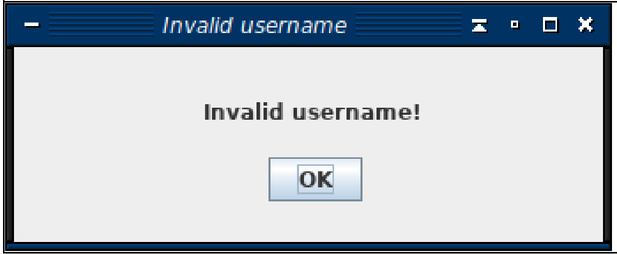
		This output list is created when clicking All Jobs in the Technician tab.	JobOutputForm

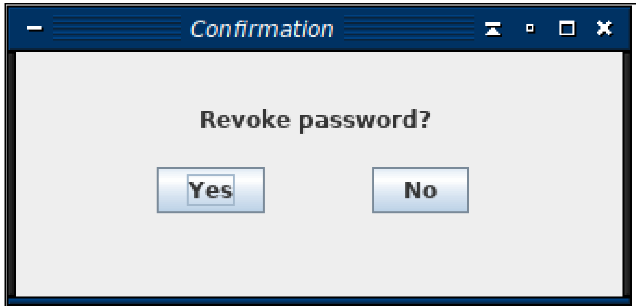
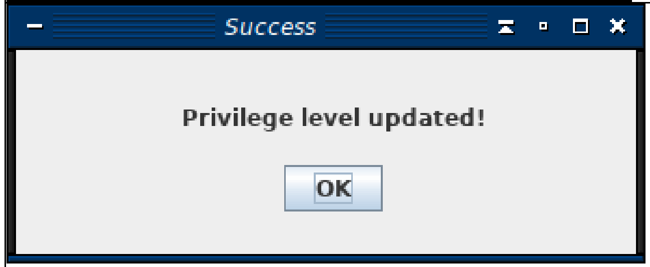
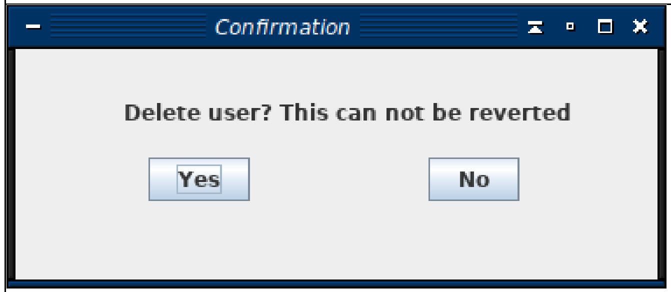
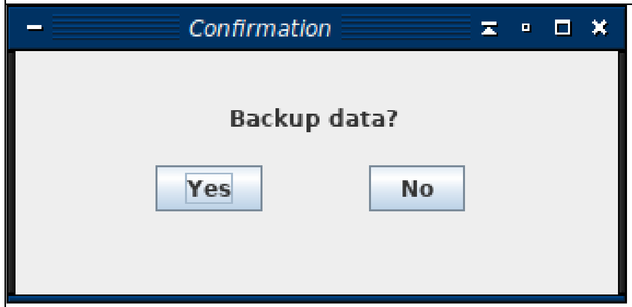
		In the Shift Manager tab, this is the output when clicking View in Individual Report.	ReportOutputForm

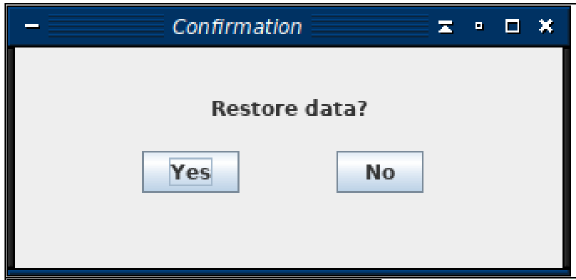
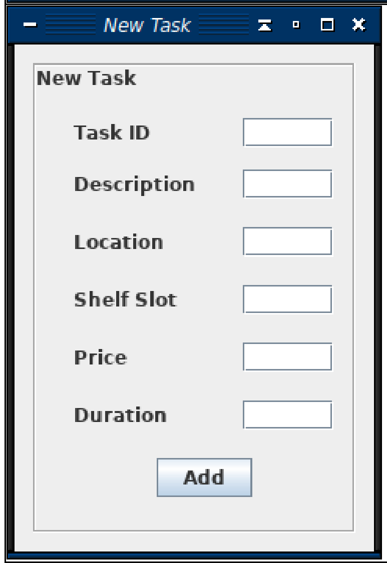
 Individual Performance Report Individual Performance Report Output		Individual performance report displayed	ReportOutputForm

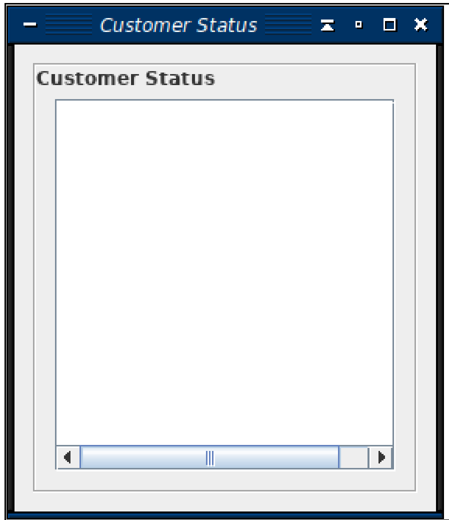
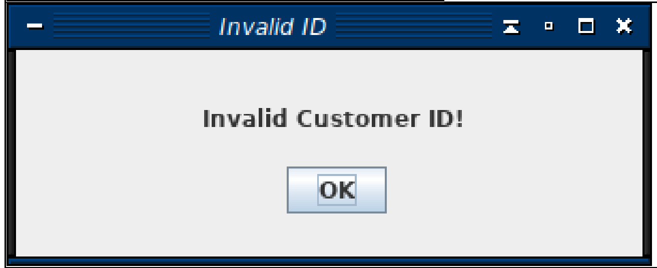
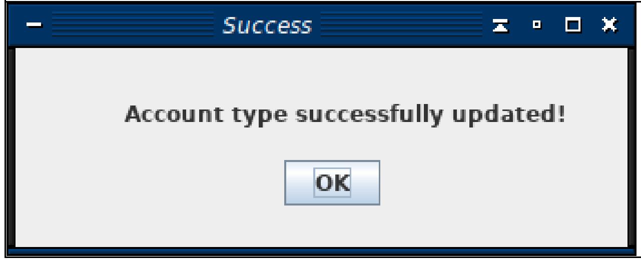
 Summary Performance Report Output	Summary performance report displayed	ReportOutputForm
 No report selected for printing! 	Nothing has been selected to print	MessageForm
 Invalid Customer ID! 	If an invalid Customer ID has been inputted in the Shift Manager tab, this popup warning appears. OK returns back to the tab.	MessageForm

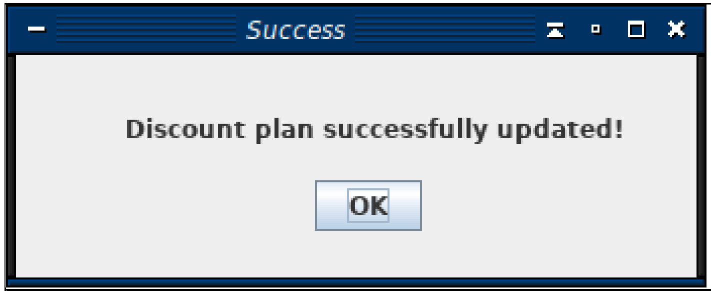
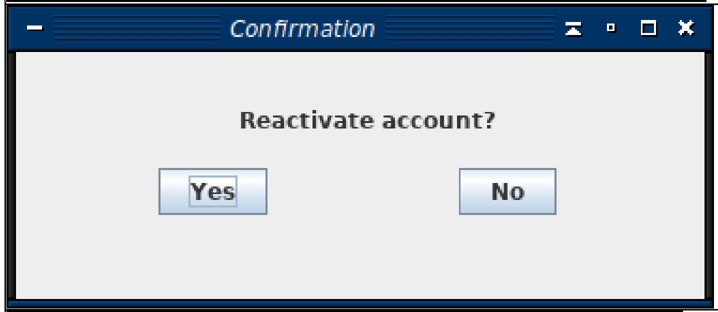
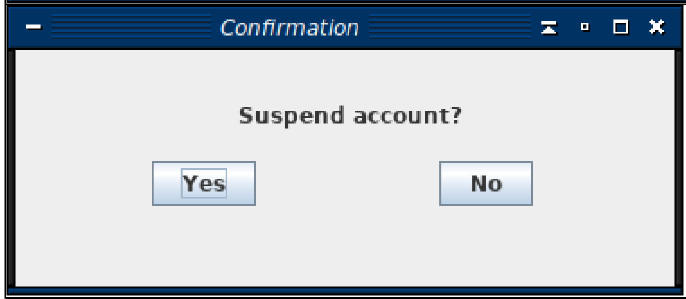
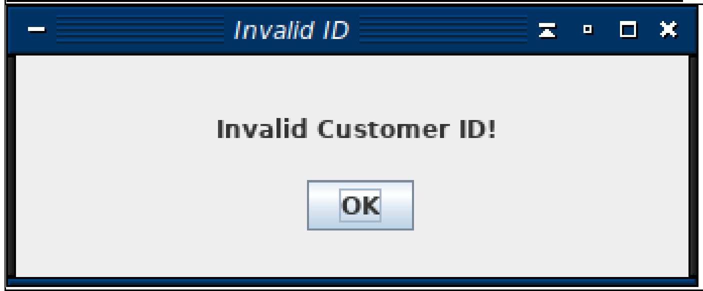
 Nothing selected to view! 	If no options in a report section has been selected, and the View button is clicked, this popup warning appears. Pressing OK returns back to the Shift Manager tab.	MessageForm
 New User Account ID Username Password Confirm password Role: Item 1 	An administrative option in the Office Manager tab, this form appears after clicking the New User button. A new user of the BAPERS system can be made this way. Clicking on Create Account will give a confirmation message if the user has been successfully created or if there is an error.	NewUserForm
 Account ID already in use! 	Account ID is already being used	MessageForm

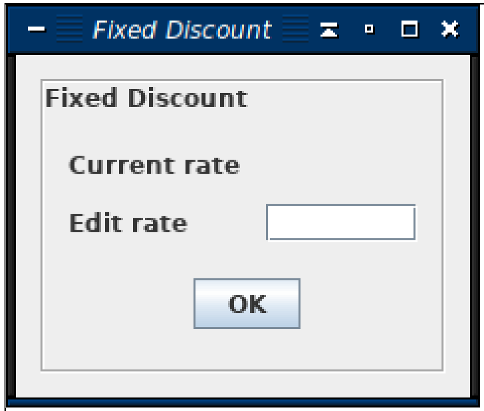
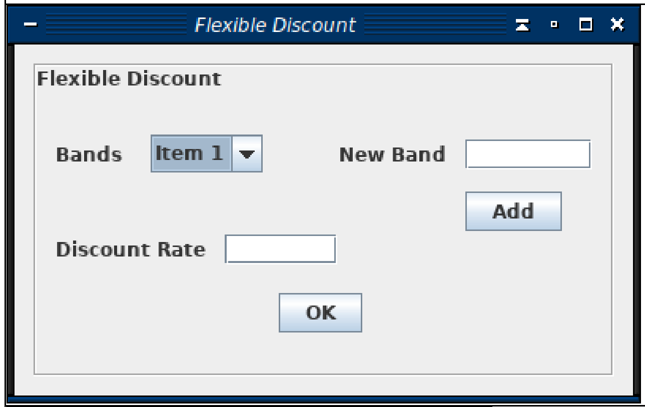
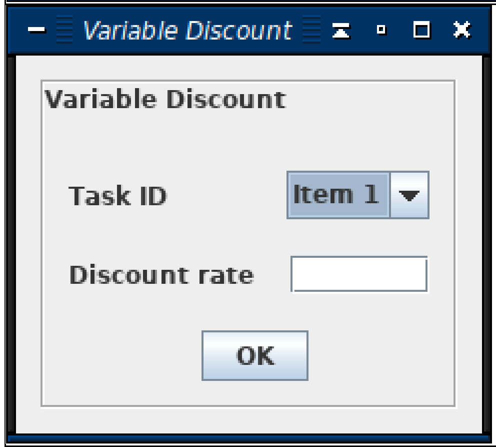
 User successfully added!	If all goes well in the New User form, this popup message appears after clicking Create User. Clicking OK returns to the Office Manager tab.	MessageForm
 Modify User Privileges Modify User Privileges Username OK Current privilege level Change to Item 1 Save changes Revoke password Delete user	Another administrative option, the user privileges can be edited. A username has to be given and clicking OK will give information about that user, if valid The user's privilege level can be edited here. The Save changes button should be clicked to save the changes for the user. Revoke password allows the password to be reset in case the user forgets it. Delete user allows the admin to delete the username in question. Both have confirmation popups.	ModifyUserPrivilegesForm
 Invalid username!	This is a warning popup appearing after clicking OK if there is the given username is invalid.	MessageForm

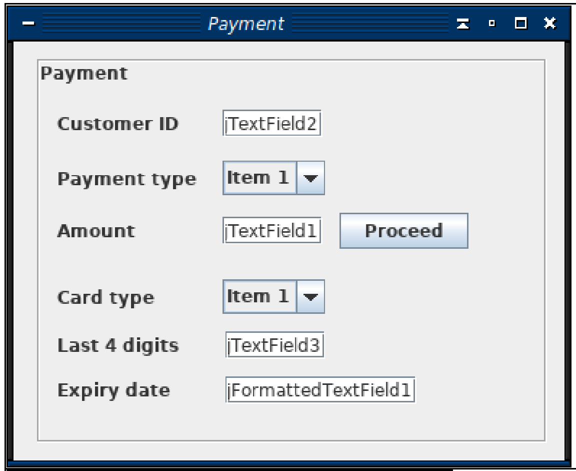
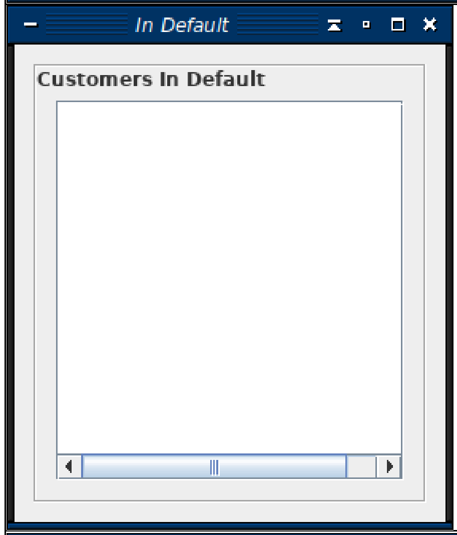
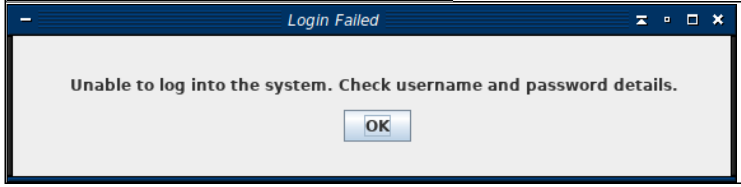
 Revoke password? 	This is a confirmation warning popup message for revoking the user’s password.	ConfirmMessageForm
 Privilege level updated! 	After clicking Save changes, the privilege level for the user in question is changed and this confirmation popup message appears.	MessageForm
 Delete user? This can not be reverted 	This is a warning popup message appearing after clicking Delete user. Clicking on Yes will permanently delete the user in question from the database.	ConfirmMessageForm
 Backup data? 	This is a confirmation popup message in the Office Manager tab when clicking on Backup data.	ConfirmMessageForm

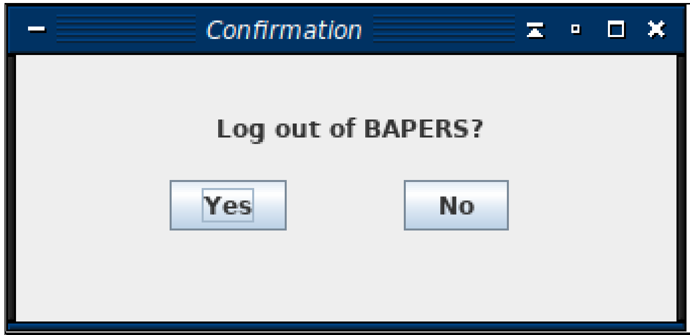
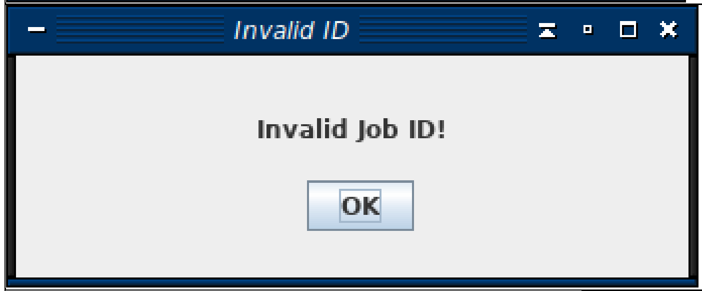
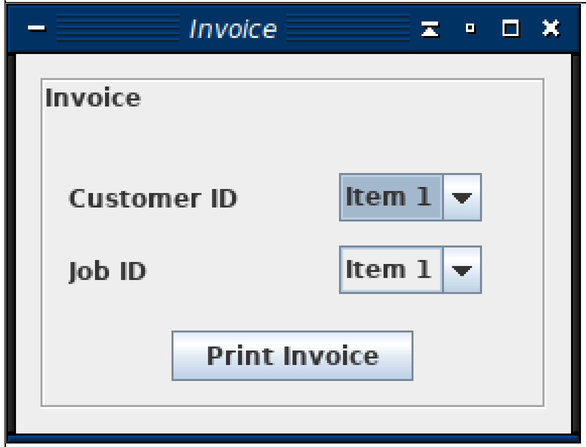
 Confirmation Restore data? 	This is a confirmation popup message in the Office Manager tab when clicking on Restore data.	ConfirmMessageForm
 New Task New Task Task ID Description Location Shelf Slot Price Duration 	This form is used to create new tasks	NewTaskForm

 Customer Status	Customer status will appear here for the office manager	CustomerStatusForm
 Invalid Customer ID! 	Invalid ID has been entered	MessageForm
 Account type successfully updated! 	Account type has been updated	MessageForm

 Success Discount plan successfully updated! 	This popup message appears when the account type has been successfully changed after clicking Update in the Customer Status section of Office Manager for the Customer in question.	MessageForm
 Confirmation Reactivate account? 	This popup message appears when OK is clicked in either the Fixed Discount, Flexible Discount, or Variable Discount forms and will update the discount plan for the customer in question.	ConfirmMessageForm
 Confirmation Suspend account? 	This is a confirmation popup message when the Office Manager clicks on the Reactivate button.	ConfirmMessageForm
 Invalid ID Invalid Customer ID! 	This is a confirmation popup message when the Office Manager clicks Suspend for the customer account in question.	MessageForm

 Fixed Discount Fixed Discount Current rate Edit rate OK	This allows a fixed discount plan to be created or modified for a customer account. It appears when Fixed Discount is chosen for a customer in question and Select has been clicked afterwards. Clicking OK will save the changes for the customer.	FixedDiscountForm
 Flexible Discount Flexible Discount Bands Item 1 New Band Add Discount Rate OK	This is for creating a flexible discount plan for a customer in question, after choosing Flexible Discount and clicking Select.	FlexibleDiscountForm
 Variable Discount Variable Discount Task ID Item 1 Discount rate OK	This creates a variable discount plan for the customer in question.	VariableDiscountForm

 Payment Payment Customer ID jTextField2 Payment type Item 1 Amount jTextField1 Proceed Card type Item 1 Last 4 digits jTextField3 Expiry date jFormattedTextField1	Payment form which would store details of the payments and customers	PaymentForm
 	This is a list of customers in default, outputted when the user clicks on In Default List in the Office Manager tab.	InDefaultAccountsForm
 Unable to log into the system. Check username and password details. 	Login has fail if details are incorrect	MessageForm

 Confirmation Log out of BAPERS? 	This warning popup message appears when the user clicks Log Out in the BAPERS main screen, It is a confirmation for if the user wants to log out of the system.	ConfirmMessageForm
 Invalid ID Invalid Job ID! 	This warning popup message appears if the Technician has inputted an invalid Job ID in the Inspect Job section.	MessageForm
 Invoice Invoice Customer ID: Item 1 Job ID: Item 1 	This form appears when clicking Proceed in the Payment form. It is for creating an invoice for a receiving payment from a customer and its job. The selection is from the list of completed jobs, hence why the selection is limited to a list in a combo box. Clicking Print Invoice will get the inputted information from the database and send a request to the operating system to print the document.	InvoiceForm