Working Product Implementation Report TEAM 23

Summary

The production and creation of the final software. Various diagrams to increase the clarity and ease of how the system was created. The aspects of the various source files, programming languages and compilers used to create the final solution. Lastly, a detailed test plan has also been included to produce a thorough understanding of the document.

Team 23: Mayur, Myan, Parmveer, Raymun, Anil and Lukas

ABSOLUTE CONSULTANCY

SIMPLE SOLUTIONS FOR COMPLEX CONNECTIONS

Contents

Induction to the document

Introduction

This document will state a full detailed approach to the implementation of BAPERS. The following will be included:

- Component diagram
- Deployment diagram
- Compilation process
- List of software components
- Detailed Test Plan

This document will list:

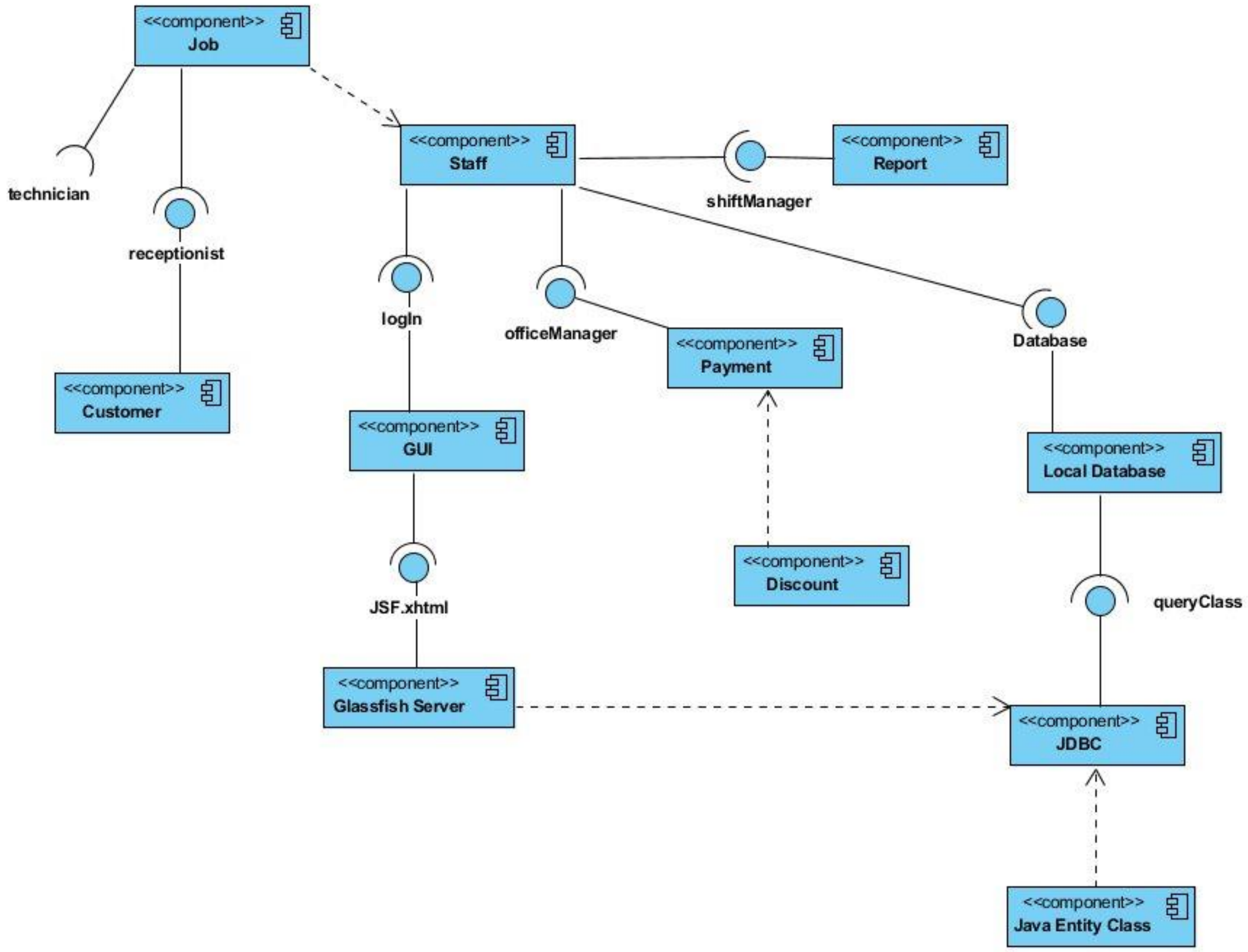
The various diagrams to demonstrate the implementation of the software. Both Component and Deployment diagrams have been introduced to show an accurate representation of the resources used. Furthermore, the compilation process has also been included which will show the various types of source files, programming languages and compilers used throughout this implementation. A list of software components will also be stated to demonstrate what are required to run the software. Finally, a detailed test plan will also be included, covering both functional and Non-functional requirements.

Component Diagram

A Component Diagram has also been included within this documentation to:

Shows how the various components are interlinked with each other to create a software system, which will be used to portray the system’s architecture. The Components diagram consists of all the important components, usage, ports and the relationship between them.
Furthermore, all the provided and required interfaces, dependencies and artifacts have been used efficiently throughout the diagram to display all the essential and required information.

Component Diagram has been shown below:



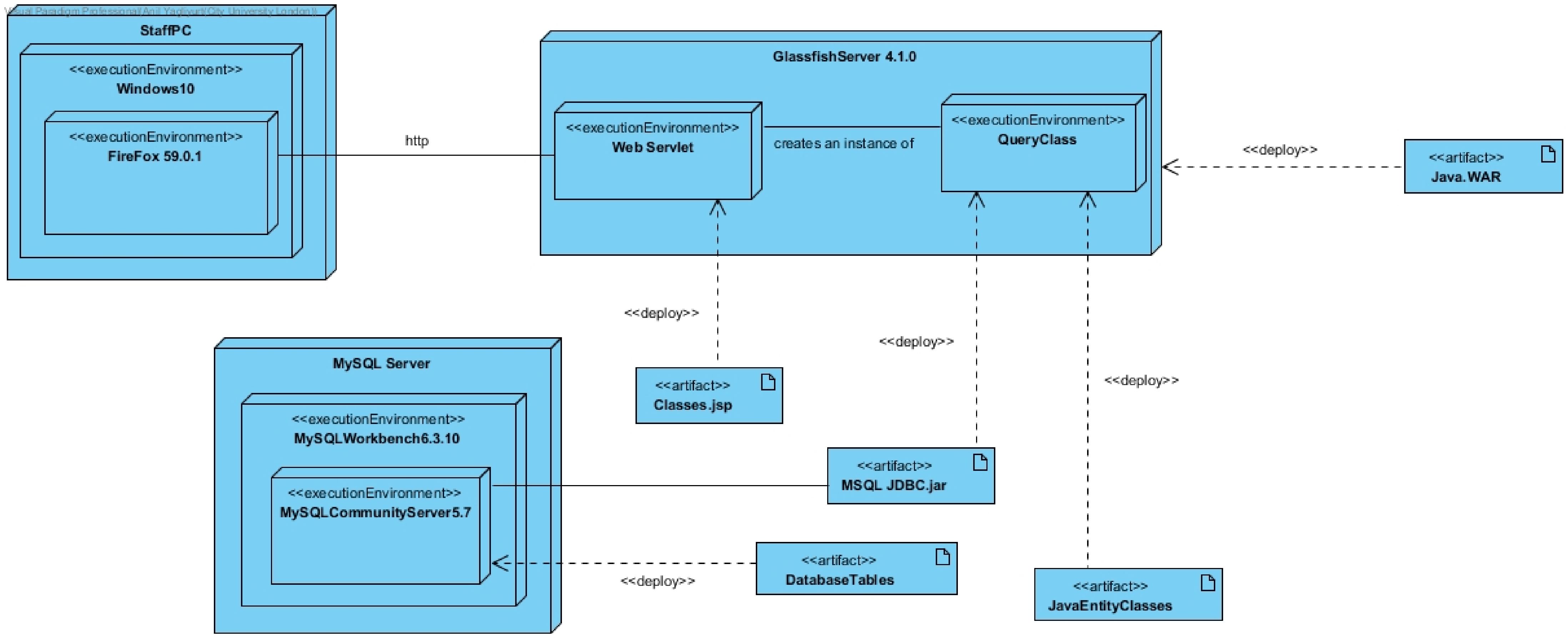
<<component>> Job
technician
receptionist
<<component>> Customer
<<component>> Staff
shiftManager
<<component>> Report
logIn
<<component>> GUI
JSF.xhtml
<<component>> Glassfish Server
officeManager
<<component>> Payment
<<component>> Discount
Database
<<component>> Local Database
queryClass
<<component>> JDBC
<<component>> Java Entity Class

Deployment Diagram

A Deployment Diagram has also been included within this documentation to:

Shows a structural diagram which will be used to portray the architecture of the system. The Deployment diagram consists of all the important execution environment nodes and artifacts. Furthermore, all software elements, association's, dependencies and stereotypes have been used efficiently throughout the diagram to display all the essential and required information.

Deployment Diagram has been shown below:


Visual Paradigm Professional(Anil Yagiyurt(City University London))
StaffPC
<<executionEnvironment>>
Windows10
<<executionEnvironment>>
FireFox 59.0.1
http
GlassfishServer 4.1.0
<<executionEnvironment>>
Web Servlet
creates an instance of
<<executionEnvironment>>
QueryClass
<<deploy>>
<<artifact>>
Java.WAR
<<deploy>>
<<artifact>>
Classes.jsp
<<deploy>>
<<deploy>>
MySQL Server
<<executionEnvironment>>
MySQLWorkbench6.3.10
<<executionEnvironment>>
MySQLCommunityServer5.7
<<artifact>>
MSQL JDBC.jar
<<artifact>>
DatabaseTables
<<deploy>>
<<artifact>>
JavaEntityClasses

Source files, programming language and compiler

Class source files:

Class Source Files	Description
Band	This is linked to the single flexible discount rates which can be applied to the system.
Cardpayment	This provides the ability to pay for the job once the orders have been completed. Debit Card or Credit card are accepted as part of this payment process.
Customer	The individual who will be purchasing the goods provided by BAPERS.
DefaultCustomer	This is a customer who has not paid an outstanding balance after a 1-month period. The customer must be a valued customer originally in order for their account to be marked as default.
Discount	Various BAPER's customers have the ability to receive a deduction from the usual cost of a product. There are also various types of discounts, such as; Fixed discount, Variable discount and Flexible discount.
Enquire	To ask information regarding a certain task. This can be asked through various methods. Such as; taskID, engquireNumber and by findingInformation.
EnquirePK	This is a unique primary key which is linked to the tasktaskID and enquireNumber
fixeddiscount	This is a specific type of discount factor for 'valued customers'. Whereby a set discount can be applied to the products they have purchased. Value of discount is calculated and applied when the job is accepted.
Basetask	This consists of the tasks BAPERS can do, e.g. make use of a large copy camera. The task consists of the base task and a job with for example, start time being recorded.
Job	The job holds all the vital information regarding the order the customer has placed. This includes the Job ID, the date of the order, collection date, special instructions involved in the order, deadline, sub charge and the value of the job.

Payment	Process of paying for the Job order, once it has been completed. This can be done through various payment methods, such as; cash, credit or debit card.
Staff	These are the individuals employed who will be using BAPERS system. The staff's personal information, such as; Staff ID, Name, Surname and role is used to identify the specific staff member.
Suspendedcustomer	The valued customer will become a 'Suspendedcustomer' if they have not paid an outstanding balance within a 1-month period.
Task	These are the various types of services that must be undertaken by technicians in order to complete a job. For example: A job may involve the prospect of using a large copy camera task in order to complete the job. Each task has a unique identifier.
Valuedcustomer	This is a customer who has a special discount plan, whereby they have the ability to pay by the 10th of each month for their jobs being completed by the end of the previous calendar month. Allocated the status of being a Valuedcustomer if regular purchases are made frequently and jobs are paid on time.
Variablediscount	A discount plan only allocated to valued customers. A percentage discount which has been set to each task and may vary according to certain tasks. Discount is deducted when job is accepted. The information regarding a Variablediscount, can be asked through various methods. Such as; DiscounteddiscountID, Amount and by BasetaskbaseTaskID.
VariablediscountPK	This is a unique primary key which is linked to the getDiscountdiscountID and TasktaskID.
addCustomerQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. Customer allowed to provide a query if an issue occurs. This allows customer queries to become prioritised and therefore allocated to a higher level within the list.
loginQuery	Any staff within BAPERS can hold a query if they hold any problems regarding login. For example; forgotten login details.
viewCustomerQuery	This allows the staff and/or work colleague to view all the appropriate customer queries to help give positive and reliable feedback.

viewTasks	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. These are the various types of services that must be undertaken and viewed by staff members in order to complete a job. A typical example of a job may include, using a copy camera to finish the job. Each task viewed had a specific identifier.
addCustomerServlet	this is to make a new customer in the servlet, when the user submits or clicks on a form.
addTaskServlet	This is to add and make a new task in the servlet.
loginServlet	Login servlet is the Java class, written to be run on a server and produce the login web page. Output from running Login servlet is to produce a login form via HTML. A request for the Login page comes from the web browser, web server recognises this request as one that requires a servlet to be run to produce the Login page. Output is HTML.
selectedCustomerServlet	selectedCustomerServlet is the Java class, written to be run on a server and produce the ability to select Customer Servlet web page. Output from running selectedCustomerServlet is to produce a customer servlet form via HTML. A request for the selecting customer page comes from the web browser, web server recognises this request as one that requires a servlet to be run to produce the select customer page. Output is HTML.
selectedTasksServlet	selectedTasksservlet is the Java class, written to be run on a server and produce the select staff web page. A request for the Tasksservlet page comes from the web browser, web server recognises this request as one that requires a servlet to be run to produce the Tasks servlet page. Output is HTML.
viewCustomerServlet	viewCustomerSerlvlet is the Java class, written to be run on a server and produce the ability to see the Customer web page. Output from running viewCustomerServlet is to produce a customer servlet page via HTML. A request for viewing the customer page comes from the web browser, web server recognises this request as one that requires a servlet to be run

	to produce the view customer page. Output is HTML.
addCardServlet	addCardServlet is the Java class written to be run on a server and produce the ability to add a card Servlet web page. Output from running addCardServlet is to produce a card and/or invoice page via HTML. A request for adding a card comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to add a card. Output is HTML.
addCashServlet	addCashServlet is the Java class written to be run on a server and produce the ability to add cash Servlet web page. Output from running addCashServlet is to produce a cash payment and/or invoice page via HTML. A request for adding a cash payment comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to add cash. Output is HTML.
addJobServlet	is the Java class written to be run on a server and produce the ability to add a job Servlet web page. Output from running addJobServlet is to produce a job, and add tasks and relevant priority page via HTML. A request for adding a job comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to add a job. Output is HTML.
selectedPaymentCustServlet	selectedPaymentCustServlet is the Java class written to be run on a server and produce the ability to select the payment for the customer Servlet web page. Output from running selectedPaymentCustServlet is to produce a selected customer for payment page via HTML. A request for selecting a payment for customer comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to select the customer for payment. Output is HTML.
totalPaymentServlet	totalPaymentServlet is the Java class written to be run on a server and produce the ability to add up the total payment Servlet web page. Output from running totalPaymentServlet is to produce a total for the entire payment page via HTML. A request for totalling up the payment comes from the web browser; web server recognises this request as one that requires a

	Servlet to be run to produce ability to total up the payment. Output is HTML.
viewInvoiceServlet	viewInvoiceServlet is the Java class written to be run on a server and produce the ability to view the invoice of the payment Servlet web page. Output from running viewInvoiceServlet is to produce an invoice for the payment page via HTML. A request for viewing the invoice payment comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to view the total invoice. Output is HTML.
viewPaymentCustServlet	viewPaymentCustServlet is the Java class written to be run on a server and produce the ability to view the payment of the appropriate customer Servlet web page. Output from running viewPaymentCustServlet is to produce a payment for the customer page via HTML. A request for viewing the payment comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to view the total payment for customer. Output is HTML.
viewReceptionistServlet	viewReceptionistServlet is the Java class written to be run on a server and produce the ability to view the receptionist Servlet web page. Output from running viewReceptionistServlet is to produce a receptionist page via HTML. A request for viewing the receptionist comes from the web browser; web server recognises this request as one that requires a Servlet to be run to produce ability to view the receptionist. Output is HTML.
Query	Query languages were used to make queries in the database. Queries were primarily used to find specific data by filtering the criteria. This query class sets up JDBC and more specific query classes inherit from the initial, first one (connects to JDBC) and runs the SQL database.
AddBandQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it adds a discount ID into flexible discounts, such that it sets up the flexible discount. Furthermore, it also creates a band with the given flexible discounts and information provided.

AddCustomerQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it adds a new customer.
AddFixedDiscountQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it adds a new fixed discount.
AddJobQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it has the privilege to add a new job as well as has the ability to return any keys generated through auto increment which are required to create a task. Furthermore, this query created a valued job which records the dates for reminders, alerts and suspensions.
AddPaymentQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it holds and adds the standard cash and card payment system. Furthermore, it also checks for flexible discounts (if required) and adds on the value of a new job to an existing flexible discount plan. Lastly, removes all suspended and default customers.
AddStaffQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it adds a new member of staff to BAPERS. It also adds a technician to BAPERS for specific criteria.
AddTaskQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it adds a task to BAPERS (not base task). Along this, it returns the description of a base task given as a parameter.
AddVariableDiscountQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it inserts a new variable discount into BAPERS.
CheckForLatePaymentsQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it checks for the tenth of the month with alert and return relevant information about the customer to be used within the alert. Furthermore, this query has first reminder alert along with the suspend accounts which have not met the second reminder deadline. Lastly, resets all appropriate flexible discounts every month.
CollectJobQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it notifies and marks a job as collected.
CreateBackUpQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it creates backup as well as changes the path of

	the utility and the second one of where you want it to be stored.
DisplayReminderQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it generates the information needed for the second reminder letter alert. Alongside this, it gets information for the alert for the first job reminder.
GetAlertQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it gets information from new jobs to display in a alert box. Similarly, it also returns the descriptions of the tasks associated with the job ID.
GetCurrentJobsQuery	This inherits from the initial query class and runs SQL using JDBC.
GetCustReportQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it returns the number of jobs and value of the given customer in a time period per day (used for the generated report). This query is the same as the day query however on a monthly/annual basis.
GetDataForInvoiceQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it returns a customer given a job ID, also returns a set of tasks given a job ID and returns the rest of the job information given a job ID. Furthermore, it gets a discount plan attached to a job (if applicable).
GetFinishedJobsQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it returns all the jobs in the system marked as finished.
GetLateJobsQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it gets jobs which will not meet the deadline set to it/them. Alongside this, it stops the alert for a job which will not meet the deadline.
GetPaymentGivenCustomerQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it gets all jobs from a customer which has not been paid. Furthermore, it gets customer who have, an invoice produced.
GetPerformanceReportQuery	This inherits from the initial query class and runs SQL using JDBC. Within this query it gets the jobs in the morning shift in a correct time period………………
getStaffQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get

	data from the SQL database using JDBC. This query is responsible for setting the type of staff details (e.g. employee type, name, surname, staff ID) and getting all the staff details.
getTechnicianJobQuery	Inherited from the Query class and the data is retrieved from the SQL database using JDBC. Consists of technician receiving specific job details, such as the deadline of the job and the customer it is based for.
getStaffReportQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for retrieving the staff report which is based on: time, date, task, staff ID, base task and department.
sendEmails	This class consists of various operations, it's main purpose is the intention of sending emails and reminders. Also includes the detailed message information which is specific based on the first reminder letter and the second reminder letter.
getVariableDiscountQuery	The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for ensuring that variable discount is identified and only applied to valued customers, who have a specific customer ID. This also sets the discount amount applied to the specific task.
loginQuery	An extension to the Query class, allowing extra features to be added. The code is fundamentally used to communicate and get data from the SQL database using JDBC. The query is responsible for the area of logging in. Requests the database for the correct first name, surname and password to allow authorisation.
removeDefaultQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. Within this query there is an update made to the customer account to change it from suspended to default. Also updates jobs when customer pays for them.
removeLateAlertQuery	Inherited from the Query class and the data is retrieved from the SQL database using JDBC. The primary function of this query is to ensure an alert is removed and the job or customer is updated to the correct status after this has

	been done. Consists of the removal of late payment alerts, first reminder alerts, suspended alerts and default alerts.
restoreFromBackupQuery	The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for ensuring that a restore can be made from a previous backup which was made as a SQL file.
searchForCustQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for searching the type of customer and is based on the getting this through customer details (e.g. ID, name, surname, phone, email, address, postcode)
setDiscountQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for setting the type of discounts, creating new discounts for customers based on customer ID and removing discounts based on customer ID.
stopAlertQuery	The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for ensuring that the system no longer produces an alert once the job has been updated.
updateBulk	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. The primary function of this query is to ensure that an update is made to make an account default or suspended. Reminders, bulks and dates are used concurrently in order to determine this.
updateCustomerQuery	An extension to the Query class, allowing extra features to be added. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is responsible for allowing the customers details to be updated (e.g. customer ID, name, surname, phone, email, address, postcode and holder)
updateJobQuery	An extension to the Query class, allowing extra features to be added. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This

	query is responsible for allowing the job query to be successfully updated or ended when completed. This is based primarily on the staffID and taskID to provide an update onto this.
valuedCustomerQuery	Inherited from the Query class and the data is retrieved from the SQL database using JDBC. The primary function of this query is to ensure that a customer is set as a valued customer based on their customer ID or to remove the customer from being valued.
viewCustomerQuery	This is an inheritance of the Query class, to add existing and additional features. The code is fundamentally used to communicate and get data from the SQL database using JDBC. This query is used to view the type of customers based on various functionalities, such as selecting payment customers, all customers, invoice customers, suspended customers, bulk suspended customers, default customers and reminder customers.
viewTasksQuery	Inherited from the Query class and the data is retrieved from the SQL database using JDBC. The primary function of this query is to set and get tasks based on specific options. Consisting of the ability to view tasks, add tasks, disable tasks, start tasks and finish tasks.
viewTechQuery	An extension to the Query class, allowing extra features to be added. The code is fundamentally used to communicate and get data from the SQL database using JDBC. The query is responsible for the viewing of the jobs accounted in the technician's room. It also gets the jobs information and is able to see the results which have been set for them, such as customer name, task and department.
Servlets.Task **(addNewTaskServlet)** **(disableTask)** **(viewTask)**	Within the servlets.Task folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example. addNewTaskServlet in order to add new tasks to the job, disableTaskServlet to remove tasks from the job and viewTaskServlet which is responsible for getting the information regarding all the tasks. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletTask is an interface that implements for creating any servlet. The servlet

	class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.admin (addFixedDiscountServlet) (addFlexibleDiscountServlet) (addStaffServlet) (addVariableDiscountServlet) (completeCustomerServlet) (createBackUpServlet) (lateJobAlertOfficeServlet) **(latePaymentAlertStopperServlet) (manageBandServlet) (newJobAlertServlet) (removeDefaultServlet) (restoreFromBackupServlet) (updateCustomerServlet) (viewAdminServlet) (viewOfficeManagerServlet)**	Within the servlets.admin folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example: addFixedDiscountServlet is responsible for allowing a discount to be added. addFlexibleDiscountServlet is used to allow a flexible discount to be added. addStaffServlet in order to add staff to the system. addVariableDiscountServlet allowing a variable discount to be implemented. completeCustomerServlet to complete customer details. createBackUpServlet to allow a backup to be created. lateJobAlertOfficeServlet to alert of a job being late. latePaymentAlertStopperServlet to prevent alerts from persisting when payment has been received. manageBandServlet to inform of the various bands available. newJobAlertServlet to send a alert informing of a new job. removeDefaultServlet in order to remove default from a customer once payment has been received. restoreFromBackupServlet to restore data from SQL regarding system which was backed up previously. updateCustomerServlet to change account type information to updated version. viewAdminServlet- to view administrators screen. viewOfficeManagerServlet to see the screen that the Office manager has capability to see. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletadmin is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.shiftManager (selectCustForReportServlet) (generateCustReportServlet) (generatePerformanceReportServlet) (generateStaffReportServlet) (lateJobAlertServlet) (newJobAlertShiftServlet)	Within the servlet.shiftManager folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example. selectCustForReportServlet to select the appropriate customer for their report, generateCustReportServlet in order to generate

(selectedStaffShiftServlet) (viewShiftManagerServlet) (viewStaffServlet)	the relevant customers report, generatePerformanceReportServlet to process the request and generate the performance report of the staff, generateStaffReportServlet in order to generate and create the report for the relevant staff, lateJobAlertServlet to process and acknowledge the late job request, newJobAlertShiftServlet in order to alert the shift manager with a new job, selectedStaffShiftServlet to show and process the staff selected for the shift, viewShiftManagerServlet in order to simply view the shift manager and viewStaffServlet to view the appropriate staff. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletshiftmanager is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance, portability and security.
Servlets.technician (selectedJobTechnician) (taskCompletedServlet) (updateTechnicianJobServlet) (viewTechnicianServlet)	Within the servlet.Technician folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example, selectedJobTechnician to ensure the appropriate job is selected from the technician, taskCompletedServlet in order to acknowledge the completed task, updateTechnicianJobServlet to update the appropriate job for the technician and viewTechnicianServlet in order to simply view the technician information. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servlettechnician is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.login (loginServlet)	Within the servlet.Login folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example, loginServlet to ensure all the relevant details for an appropriate log in. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression

	Language is the fundamental language which has also been used on the JSF pages. The servletlogin is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance, portability and security.
Servlets.email sendFirstEmailServlet sendSecondEmailServlet	Within the servlet.Email folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example, sendFirstEmailServlet to send the customer a special email for appropriate information and sendSecondEmailServlet to go through the same process, again. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletemail is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.Invoice (selectedInvoiceCustServlet) (totalInvoiceServlet) (viewInvoiceCustServlet) (viewInvoiceServlet)	Within the servlet.Invoice folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example, selectedInvoiceCustServlet to select the appropriate invoice for the customer, totalInvoiceServlet in order to tally up and total the entire payment/invoice, viewInvoiceCustServlet to view the appropriate invoice for customer and viewInvoiceServlet to simply view the invoice. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletInvoice is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.latepayments (secondServlet) (bulkSecondServlet) (bulkFirstServlet) (firstServlet) (latePaymentServlet)	Within the servlet.latepayents folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example, secondServlet to produce and process the second batch of late customers, bulkSecondServlet in order to – as a

	whole – produce the second late customers payment, bulkFirstServlet to as a whole, produce the first batch of late customers, firstServlet in order to request the process of the first batch of late payments and latePaymentServlet to view all late payments. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletlatepayments is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability and more secure.
Servlets.payment (addCardPayServlets) (addCashPayServlets) (generateReceiptServlet) (selectedPaymentCustServlet) (totalPaymentServlet) (viewPaymentCustServlet)	Within the servlets.payment folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example. addCardPayServlets in order to add card payments to the jobs available, addCashPayServlets to add a form of cash payments for jobs, generateReceiptServlet which is responsible for getting the information regarding all the jobs, payments and creating a receipt for this. selectedPaymentCustServlet to request payment from a specific customer. totalPaymentServlet to calculate total of payments. Lastly, viewPaymentCustServlet to view if payment from customers that have paid through cash or card format. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletpayment is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.
Servlets.receipt (addCustomerServlet) (addJobServlet) (addTaskServlet) (collectJob) (searchForCustServlet) (selectedCustomerServlet) (selectedTasksServlet) (viewCustomerServlet) (viewFinishedJobsServlet)	Within the servlets.receipt folder there are various classes which have been used to extend the capabilities of servers that host applications accessed. For example. addCustomerServlet in order to add new customers to the records, addJobServlet to add new jobs to the list, addTaskServlet which is responsible for adding new tasks to jobs, collectJob to inform customers to collect their jobs when completed. searchForCustServlet to find a

	specific customer by ID, selectedCustomerServlet to select a specific customer, selectedTasksServlet to select a specific task, viewCustomerServlet in order to see the details of the existing and new customers available, viewFinishedJobsServlet to see which jobs have been completed. These servlets run query classes in order to get the data to produce a JSF page. JSF Expression Language is the fundamental language which has also been used on the JSF pages. The servletreceipt is an interface that implements for creating any servlet. The servlet class extends the capabilities of the servers and responds to the incoming requests. This ensures better performance and portability.

Programming language

The production of a webpage is created through various types of programming languages, such as HTML, CSS and JavaScript. Concurrently they are all linked to each other to produce and display these webpages. These are only some of the languages which have been used to allow the creation of the website, there are many more which will be talked about in this section.

Programming Language	Description
HTML (Hyper Text Mark-up Language)	Is a standard mark-up language used for the creation of web pages. HTML has been used across the development of our software to define all the content, text and the images we have used throughout the process. It has been developed with the ability to be displayed on any sort of web pages. Such as Firefox, Internet explorer, chrome and many others. HTML has been used across all the servlets within our program to output the information (within the java classes) to a webpage. Such as the customer servlet java class produces the customer page via HTML.
Java	Java is an object-oriented language. JVM also executes Java code. We have used a Java web application to create server-side web applications. Currently, servlet, JSF etc. technologies are used. In java, JSP is used to create a dynamic web page such as PHP. Java is a fundamental programming language which has been used across our program to create several different Java Class Source files. Lastly, we used a Java IDE, NetBeans that is open source and free
Java Script	JavaScript can change HTML content. JavaScript accepts both double and single quotes. JavaScript can change HTML attributes. JavaScript can change HTML Styles (CSS). JavaScript can hide HTML Elements.
CSS	CSS (Cascading Style Sheets) – describes how HTML elements display on screen, and/or in other media. Ideally it controls the layout of multiple web pages all at once. We used CSS to define different styles for our web pages, including the design, the layout, and variations in display for different devices and screen sizes. We used CSS with an external stylesheet file, as it allowed us to change the look of the entire web application just by changing one file.
SQL (Structured Query Language)	SQL ideally lets you access and manipulate databases. SQL allowed us to execute queries against the database as well as retrieve data from the database. We used SQL to allow us to insert and update records to and from the database. We also used it to create new tables and stored procedures in the database. To build

	the web application, to show the data from the database, we used a Relational Database Management System program (i.e. SQL Server).
JSF (Java Server Faces)	This is a java specification used for building component based, user interfaces for the web application. Servlets populate from the JSF pages and JSP is the language used on JSF pages.
Expression Language	This is also part of JSF and JSP. It provides mechanisms for enabling the presentation layer (web pages) to communicate with the application. The Expression language is used by both JSF and JSP.
Java Servlets	These are used to extend the capabilities of servers that hose applications accessed by requests and responses. Within this project they commonly used to extend the applications hosted by web servers. Servlets show the JSF page, I.e. the contents applied to the JSF pages using JSF Expression Language and embedded Java. Servlets run query classes to get the data to populate the JSF pages.
MySQL server 5	SQL ideally lets you access and manipulate databases. SQL allowed us to execute queries against the database as well as retrieve data from the database. We used SQL to allow us to insert and update records to and from the database. We also used it to create new tables and stored procedures in the database. To build the web application, to show the data from the database, we used a Relational Database Management System program (i.e. SQL Server).
JDBC	JDBC will work alongside MySQL databases. The reason why we chose this is because it is object oriented and also offers a procedural API. It supports prepared statements and protects from SQL injection. Not only that, but also important for web application security. We used this when connecting to MySQL using JDBC and ensured that MySQL was configured to accept external TCP/IP connections. Lastly, it helps connect Glassfish to the MySQL server.
Maven	This is a build automation tool used particularly and primarily for this web application. Maven ideally addresses two aspects of building software, firstly, the description of how the system is built and secondly, it describes the dependencies which we used to import JavaMail as a dependency for the emails (sorts our dependencies and to use JavaMail library)

	This inherits from the initial query class and runs SQL using JDBC. Within this query it generates the information needed for the second reminder letter alert. Alongside this, it gets information for the alert for the first job reminder.

Description of the Compilation

The system was developed using Java EE 7 (enterprise edition) through NetBeans IDE. The user interface originally consisted of HTML pages generated in AxureRP which have been converted to JSP pages in order to make use of JSF Expression Language (version 2.0+) to display dynamic data on the page. The system was compiled using Maven compiler which will automatically import any external libraries. The only external library used was JavaMail, which allows for email reminder letter generation. An Internet connection is required to run BAPERS, due to the email reminders, if BAPERS is to be used without an internet connection, comment out the following lines of code:
1) Lines 176 and 166 on the officemanagerservlet to comment out all first reminder email alerts and second reminder email alerts. Replacing these will disable all the email reminders.

Compiler Settings

Our web application was built using the NetBeans 8.0.2 IDE, contained in a NetBeans project file as a Maven Java EE Application. The project can be opened in the IDE using the ‘Open Project’ option from the menu bar and the project will open as a Maven project. Maven is a build management tool which is used to define how the .java files get complied to the .class packaged into .jar(or .war or .ear) files.

As the system is a web application, the use of glassfish was required. Glassfish is defined as an application server. This means it can be used to add interactive web front end onto Java programs amongst other functionality outside the scope of our system. Glassfish can be added to Netbeans by right-clicking Services > Add Sever on the Netbeans services window(The Add Server instance wizard appears). After being prompted to select the relevant version of GlassFish, the server location page appears. by default, the GlassFish server is installed to “JavaCAPS_Home\appserver.”. Once the correct domain has been chosen from the “Register Local Default” drop down list, GlassFish has been successfully added to the IDE.

For the back-end of the system, MySQL WorkBench was used to create and configure the database used in the system to hold data such as customer information, user log-in credentials, etc. Upon downloading MySQL WorkBench the ‘Empty.sql’ needs to be run on MySQL WorkBench as a query in order to create the tables needed to run the system. The file path for the backups also needs to be specified. Additionally, the “createBackUpQuery”, the path for mysqldump and the location for backups must be set. The MySQL server path also needs to be setup for the restoreFromBackups Query. Without configuring these options correctly, backups up the database cannot be stored or retrieved.

To link the database from MySQL Workbench to the IDE, a connection pool was configured from the Netbeans GlassFish Server to the SQL Schema. Once the connection pool is set up correctly, you are then able to plug the settings into the Netbeans API.
The application makes use of servlets, which are small programs that executes on a server taking in information from HTTP POST and GET methods and manipulating JSP pages according to data given, servlets are also responsible for talking to the Query classes which communicate in.

Apache Maven was used to compile the program, the pom.xml contains all the dependencies required for BAPERS, with the only external dependency being JavaMail. All libraries should be fetched automatically by Maven. Default compilation settings were also used throughout the development of the system.

External Libraries

As the system was being implemented, the use of a particular external library was used to add extra functionality to the system which is not included in the default Java development kit.
To add a new external library to Netbeans, the properties of the project need to be accessed from the Projects window and right-clicking on the name of the project the library needs to be added to. In the Categories section select "Libraries" node > on the right side of the Project Properties window press button "Add JAR/Folder" and select jar files that are needed.

The external library used to build the system was JavaMail. The JavaMail API is a platform-independent and protocol-independent frame which is mainly used to build mail and messaging applications.
Regarding our system, this library was utilised as our system automatically recognises if a customer is behind with a payment and sends them an email reminding them of their outstanding debt.

Run Time Components

The machine running the system must have MySQL Server installed. It is recommended MySQL is used(https://dev.mysql.com/downloads/mysql/) to install all the relevant packages. The packages required are MySQL Server, MySQL Workbench, MySQL Utilities, MySQL Shell. Upon successful installation of these components, in the SQL backups folder, there should be an SQL text file which can be run on workbench which will regenerate the database.

As the system was built using the JDK, the host most machine running the system is required to have the most up to date version of the Java Run-time Environment or JRE installed to run the system correctly.

The JRE is needed because it is part of the JDK, which is a set of programming tools for building Java applications. The JRE provides the minimum amount of resources needed for executing a Java application.

Software

Software	Description
Firefox	Mozilla Firefox is an open source web browser developed by Mozilla. Firefox is available for Windows, macOS, Linux, and Android, hence why we chose this browser. We decided to use Firefox 59 because it has several new features including, FLAC support, WebGL 2.0 support as well as clipboard access for Web Extensions and many more.
MySQLWorkbench	Describes and provides SQL development, data modelling, administration tools for server configuration, backup, user administration and much more. MySQLWorkbench is available on Windows, Linux and Mac OS X. Furthermore, MySQLWorkbench enabled us to design, model, generate and manage databases. Alongside this, we were able to create, execute and optimise SQL queries.
MySQLCommunityServer	This is the database system used on the web. The aim for this is for the database system to run on the server. We used this because it is ideal for a large application like BIPL. The data stored in the MySQLServer were stored in tables, a collection of related data, consisting of columns and rows. Another reason why we used this database server was due to the fact that it is useful for storing information categorically e.g. Employees, Customers, and Products etc.
Glassfish Server	Glassfish is defined as an application server. Which means it can be used to add interactive web front end onto Java programs amongst other functionality outside the scope of our system.
MSQL JDBC	JDBC will work alongside MySQL databases. The reason why we chose this is because it is object oriented and also offers a procedural API. It supports prepared statements and protects from SQL injection. Not only that, but also important for web application security. We used this when connecting to MySQL using JDBC and ensured that MySQL was configured to accept external TCP/IP connections.
JSP page (JavaServer Pages)	This is a document that contains two types of text, static data (e.g. HTML and XML) and JSP elements, construct dynamic conent. We used JSP to tackle problems, for example, instead of

	embedding HTML in programming code, JSP lets us embed specialised code into HTML pages. Java is the default scripting language of JSP, but the JSP specification allows for other language as well such as JavaScript.

Testing

Testing is a fundamentally importance part of the software development process and was used to ensure that all requirements stated were met successfully. We have identified the use of carrying out various forms of testing, such as Use Case Testing and Non-functional testing to ensure high accuracy and reliability of the software's functionality. Tests were also repeated numerous times to ensure the results provided were verified and fully precise.

Testing methods have been shown below:

Use Case Testing

Use Case Testing has also been included within this documentation to:

Perform the technique of identifying various test cases which can occur throughout the use of the whole system. The use case testing consists of all the important primary and alternative flows which transpire throughout the activities. Furthermore, all Objectives, Set Up’s and Expected results have clarified in full detail to demonstrate accurate and precise information for Users.

Use Case Testing has been shown below:

Use Case Testing: cardPay

Use Case ID: 28	**Use Case Name:** cardPay
Test number: 1	
Objective: To test the card payment functionality within BAPERS. This tests the main flow of the use case and addresses all steps of the feature. The tests finalises and approves the use of card payments in BAPERS, and makes sure the main aspects of the use case are applied i.e. payments clearing, job list updating, card details stored and customer list updated.	
Set Up: Customer exists within system and has one or more jobs completed with BAPERS. Customer is ready for collection. When selected, customers job(s) are displayed and priced (including applicable discounts) and the right card details are entered.	
Expected Results: 1. Job is updated as paid in system 2. Card details (expiry, type and last 4 digits) recorded in table 3. Payment has gone through successfully 4. Invoice is printed	
Test: 1. Receptionist logs into system (user: john.smith and password: 999) 2. Customer 'test man' is chosen 3. Specific tasks 'use printer' and '12321' are added to the job 4. Add special instructions and job type, click submit 5. View customer list, select customer 'test man', select appropriate job ID 6. Card payment is selected, and details are filled in 7. Card payment goes through successfully and invoice is printed (by pressing the 'Print' button) 8. Listed job IDs and customer lists are marked as paid and updated in system	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed.	
Date: 13/04/18	**Tester: Parmveer Johal**
Results: Passed.	

Use Case ID: 28.1	**Use Case Name:** paymentDeclined
Test number: 2	
Objective: Testing the alternative flow, whereby incorrect card payment details are inputted to pay for an order.	
Set Up: Similar to the prior main flow test case except the last part; The entered payment details are incorrect to BAPERS.	
Expected Results: 1. Job remains unpaid in the system 2. Payment is unsuccessful at being processed and therefore declined. 3. Error produced, informing card cannot be accepted as a method of payment 4. Everything else is unchanged	
Test: 1. Continuing on from step 6 of the test case (cardPay) 2. Incorrect card details (34567) are inputted into the system for the last 4 digits of the card number, however, system does not allow more than 4 digits to be entered. 3. Incorrect card details (311) are inputted into the system for the last 4 digits of the card number, however, system produces an error stating 'Please lengthen this text to 4 characters or more (you are currently using 3 characters). 4. Error is displayed 5. Payment does not go through 6. Filled out form remains incomplete 7. Selected jobs still remain unpaid	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Jobs remained as unpaid and all forms, including the payment is unchanged.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: Create staff

Use Case ID: 40	**Use Case Name:** createStaff
Test number: 3	
Objective: To test the addition of adding new staff to the BAPERS system. This tests the main flow of the use case to ensure staff have been created and they are assigned specific access levels. The test will also examine the various roles and privileges set to each staff, ensure there are no duplicate staffs (same name and surname) and most importantly the functionality of staff login.	
Set Up: User is logged in as an Office Manager and has the right privileges to add/edit staff in BAPERS. Office manager must be set up with sufficient admin rights, correct details and other relevant information necessary. The new staff values that are being added to BAPERS do not correspond with any staff currently in the system.	
Expected Results: 1. New staff is created and added into BAPERS 2. Staff table is updated 3. Newly created user is able to log in and view their respective screens.	
Test: 1. User logs in as an office manager with username: bob.ross and password: 123 2. Office manager selects admin button from its respective screen 3. Office manager opts to add new staff 4. New staff details are entered (**Name**: Jimmy, **Surname**: Gill, **Role/Privileges**: Receptionist, **Tech Room**: Copy) and 'Add Staff' button is selected. 5. Office manager exits the system 6. Office manager logins with the new the newly created staff details (username: Jimmy.Gill and password: 123) 7. Successfully directed to the Receptionist screen.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts are successful. Staff table is fully updated, the account is able to log in and is directed with the right privileges	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case ID: 40.1	**Use Case Name:** alreadyExists
Test number: 4	
Objective: Testing the alternative flow, whereby the newly created staff already exists within BAPERS.	
Set Up: Similar to the main flow of the use case. The user is logged in as an office manager and hits the admin button from its respective screen. The office manager opts to create a new staff, however enters the detail of an existing staff from the table.	
Expected Results: 1. An error is presented stating the entered staff already exists in the system 2. No duplicate is created on the table	
Test: 1. Continuing from step 4 of the use case test (createStaff) 2. Office manager enters already existing information to the (Add Staff) section. 8. Office manager enters staff details which already exist in staff records (**Name**: Jimmy, **Surname**: Gill, **Role/Privileges**: Receptionist, **Tech Room**: Copy) and 'Add Staff' button is selected. 3. Office manager is presented with an error on screen stating "Account already exists" 4. Duplicate staff record does not appear on the table	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts are successful. An error was presented and the duplicate was not added onto the table	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed.	
Date: 13/04/18	**Tester: Parmveer Johal**
Results: Passed.	

Use Case ID: 40	**Use Case Name:** insufficientPrivileges
Test number: 5	
Objective: Testing the alternative flow, whereby currently logged in user has insufficient privileges to create new staff. This makes sure a level of security is implemented into BAPERS.	
Set Up: The user is logged in as an office manager and views the customer table, looking to create new staff.	
Expected Results: 1. User cannot view the section to create/edit the staff in BAPERS	
Test: 1. User is logged in as a receptionist with username: john.smith and password: 123 2. User is directed to receptionist screen. 3. User selects customer by pressing 'View Customers' and is directed to the customer table, where a customer is selected (Timmy Turner) 4. Receptionist tries to create/alter the current table 5. Receptionist does not have the ability to access the Admin page, to add or edit staff. Therefore, do not have the relevant privileges.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The receptionist does not have the option to view the section of add/edit staff, but to merely view the staff table and select them for reports.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 14/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: setDiscount

Use Case ID: 38	**Use Case Name: setDiscount**
Test number: 6	
Objective: Test the main flow to ensure that the valid discount package can be applied successfully to the correct customer i.e. they must be valued. The test also ensures that the set discount is then successfully applied to all of the customers current jobs.	
Set Up: User is logged in as an office manager and views the valued customers. Customer table is used to identify whether they are a 'valued customer' or not. E.g. Customer ID = 24 and set up the discount packages which can be applied to give this individual a discount from their order.	
Expected Results: 1. Valued customer is correctly selected 2. Customer is not affiliated with any other discount 3. Office manager sets one of the three discounts. 4. Both the customer and his/her jobs are updated with the correct discount.	
Test: 1. User logs in as an office manager with username: bob.ross and password: 123 2. User selects the 'View customer' button and is directed to the respective screen. 3. User opts to edit by selecting a valued customer (Timmy) 4. User selects one of the three discounts types (Fixed) and enters a discount percentage (10%) 5. User saves changes. 6. Office manager navigates to Receptionist screen. 7. User selects customer (Timmy) 8. Screen displays associated discount value for customer (Fixed at 10%)	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts are successful. The user is now affiliated with a discount and all the pending jobs have the discount applied.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case ID: 38.1	**Use Case Name: customerNotValued**
Test number: 7	
Objective: Testing the alternative flow, whereby the office manager selects a non-valued customer and attempts to affiliate a discount plan with it. The system should recognise the error and prevent this from happening.	
Set Up: The user is logged in as an office manager and selects an already existing customer without a discount plan. The selected customer is also not checked as valued.	
Expected Results: 1. No discount is affiliated with customer 2. Customers pending jobs stay the same rate 3. No other changes	
Test: 1. Continuing from step 2 of the use case test (setDiscount) 2. User selects a non-valued customer (Gonzolo) from the table. 3. User affiliates a discount with the selected customer (whom is not a valued customer) 4. User saves changes 5. Error is presented by navigating to a blank screen 6. User is given the opportunity to go back and change value and discount.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The customer is not set with a discount and all the pending jobs have the same previous rate.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case ID: 38.2	**Use Case Name: discountExists**
Test number: 8	
Objective: Testing the alternative flow, whereby the office manager selects a valued customer with an existing discount plan and attempts to affiliate an extra one. The system should prevent this from happening and present an error.	
Set Up: The user is logged in as an office manager and selects a valued customer that already has a discount plan. The customer should exist as valued in the customer table.	
Expected Results: 1. Two discounts should not be applied to a single customer. 2. The changes are not applied 3. An error is presented 4. No changes to the already existing customer discount	
Test: 1. Continuing on step 4 of the use case test (setDiscount) 2. The selected customer already has a set discount type (Fixed at 30%) 3. The user attempts to apply another discount type 4. System examines the saved changes 5. Discount cannot be added, only changed or altered 6. No other changes occurred, and previous existing discount still exists	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The user has its previous discount and is not affiliated with two discounts. Also the users pending jobs have the same rate as before.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: login

Use Case ID: 12	**Use Case Name: login**
Test number: 9	
Objective: Test the main flow to ensure that staff can successfully login into the BAPERS system via their specific details. Each user is logged into their respective screens and has the right privileges associated to their accounts.	
Set Up: Not currently logged into BAPERS	
Expected Results: 1. Staff User enters correct login details. 2. System allows authorisation into system. 3. Relevant screen displayed. 4. Specific access privileges and details shown, based on type of staff member.	
Test: 1. User logs in as a receptionist with username: john.smith and password: 999 2. Account details are conveniently processed, and user is successfully logged in 3. Directed to the receptionist screen	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The user is directed to the correct screen and no other system details are updated.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/09/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case ID: 12.1	**Use Case Name: incorrectDetails**
Test number: 10	
Objective: Testing the alternative flow, whereby user attempts to log in with incorrect details. The system should prevent the user from accessing this as a security precaution.	
Set Up: Similar to the main flow test case, the use case tests the login system. However, this time the user alters the values of password when logging in	
Expected Results: 1. Error presented 2. Login not permitted	
Test: 1. User logs in as a receptionist with username: Hello and leaves password blank 2. The system fails to log in and produces an error message "Please fill in this field." 3. User attempts password: Hello123 and system fails to login	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/09/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: addSpecialInstructions

Use Case ID: 7	**Use Case Name: addSpecialInstructions**
Test number: 11	
Objective: Test the main flow to ensure special instructions informed by the customers, can be included when placing an order. The main objective is to ensure the special instructions are added to the job and the technician is notified of the special instructions through the notification bar.	
Set Up: User is logged in as a receptionist. A new job is being processed with an existing customer. The customer requires special instructions to the job.	
Expected Results: 1. Special instructions are attached to the job 2. BAPERS is notified at the arrival of a new job including the special instructions 3. Job successfully gets placed into BAPERS	
Test: 1. Office manager logs in with username: john.smith and password: 123. 2. User selects receptionist screen. 3. User selects existing customer (Timmy Turner) from the view customers list 4. User adds task/tasks to Job. 5. Special instructions (red colour to be used when film processed) and 'Regular' option is selected. 6. User submits job (including special instructions) to BAPERS 7. Notification is sent to BAPERS with special instructions 8. Office manager navigates to technician screen. 9. Screen displays a notification bar with the special instructions (red colour to be used when film processed)	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Special instruction is added to the job spec and notification is sent to BAPERS with special instruction attached to the job	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case ID: 7.1	**Use Case Name: jobMismatch**
Test number: 12	
Objective: Test the alternative flow to ensure special instructions attached to the wrong job order, allow the receptionist to rectify their mistake and apply the special instructions to the correct job order.	
Set Up: Similar to the main flow test case, the alternative flow follows the same footsteps. However, the special instructions get added to the wrong job during processing.	
Expected Results: 1. System informs an information error has occurred 2. Receptionist re-applies instructions to correct Job	
Test: 1. Continuing from step 6 of the use case test (addSpecialInstructions) 2. The user attaches the special instructions to the wrong job. 3. User realises that special instructions have been attached to wrong job order. 4. The system acknowledges that the user has made an error and allows the user an additional chance to rectify the special instruction. 5. Special instruction is now added to correct job, and changes are saved 6. The special instruction is not added to the wrong job 7. The form to add a new job stays the same	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The special instructions did not get added to the wrong job, and the form stayed the same to complete and place the order.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case Testing: payment

Use Case ID: 26	**Use Case Name: payment**
Test number: 13	
Objective: Test the main flow to ensure that the customer is able to successfully pay for their job via card/cash payment method. As well as the recording and storage of the payment on BAPERS.	
Set Up: User is logged in as a receptionist/office manager and a customer is ready to collect and pay for jobs. An existing customer is already in the system with jobs that are completed.	
Expected Results: 1. Payment is received and stored 2. Job is updated as paid 3. Receipt is printed 4. Customer jobs is updated	
Test: 1. Logged in as an receptionist with username: john.smith and password: 999 2. Receptionist navigates through to the payment page by pressing the 'Payment' button 3. Receptionist selects 'View customer list' and selects the appropriate customer (Firstname: nonvalued) 4. Receptionist then selects the relevant Job ID and clicks 'Select Jobs' 5. This navigates through to the payment form 6. User completes relevant payment (via card or cash) 7. Jobs updated as paid within the system 8. User has ability to print receipt. 9. Customers job list is updated	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Customer job is updated as paid and customer job list is also updated.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 13/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case ID: 26.1	**Use Case Name: jobLost**
Test number: 14	
Objective: Test the alternative flow whereby if a Job is lost/damaged during processing, a new collection date can be set up for the customer. The job should change scheduled dates and be placed as urgent for compensation.	
Set Up: User is logged in as a receptionist/office manager to process payments. Customer is selected and jobs listed are not yet completed/not on the shelf location.	
Expected Results: 1. Error produced informing selected job is not completed 2. New collection date alert to customer regarding Job. 3. Job marked as urgent to ensure completed within time frame. 4. Customer collects according to new date.	
Test: 1. Continuing on step 4 of the test case (payment) 2. Office manager accesses the page through the receptionist screen. 3. The user selects the respective customer and proceeds towards selecting payment option for the customer. 4. Screen displays an exceeded deadline next to actual deadline. Informing the user, that the job has not been completed. 5. User selects receptionist screen and selects the specific customer. 6. User inputs special instructions (URGENT, Job must be completed by 15th April 2018) and then selects the Urgent button. 7. Job is automatically made into urgent and compensation is calculated. 8. Customer is informed, and payment is withheld	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Job is now transcribed as urgent with no fee attached and a new scheduled date is provided.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 14/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case Testing: Alert new job

Use Case ID: 6	**Use Case Name: alertNewJob**
Test number: 15	
Objective: To alert the system in BAPERS of a new jobs arrival. The receptionist has the opportunity to submit a job order, which will produce a notification of a new job order to the shift manager.	
Set Up: User is logged in as a receptionist and selects an existing/new user and creates a new job. The job is successfully recorded.	
Expected Results: 1. Notification centre is updated 2. The job is added to the system	
Test: 1. User logs in as a receptionist with username: john.smith and password: 999 2. User clicks 'view customer' and selects an existing customer (Timmy Turner) 3. User selects tasks to add to the job and inputs special instructions. No special instructions are present so (n/a) is inputted and job is selected as regular. 4. User submits the job. 5. Notification is updated onto the shift managers screen. Displaying (New Job Detected, Customer, Instructions, Deadline and Tasks)	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Notification was sent to BAPERS notifying arrival of new job and job was successfully added to the system	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed.	
Date: 14/04/18	**Tester: Mayur Depala**
Results: Passed.	

Use Case ID: 6.1	**Use Case Name: incorrectJobType**
Test number: 16	
Objective: To alert the receptionist that the wrong job type (regular/urgent) is attached to the wrong job. i.e. customers may be valued and normal urgency is selected for their respective jobs.	
Set Up: Similar to the main flow test case, the user is logged in as a receptionist and selected an existing valued customer. The customer is then set to normal urgency on job type and processed	
Expected Results: 1. The job alert is not recorded correctly in the database. 2. Error occurs during process of job creation 3. No delivery date and completion date 4. Job is not completed	
Test: 1. Continuing from step 3 of the use case test (alertNewJob) 2. The receptionist selects appropriate tasks to add to job 3. User sets jobs as either regular or urgent priority 4. User selects 'Submit' button 5. The system acknowledges that the user has made an error and allows the user an additional chance to rectify the job priority type 6. Job priority type is now altered, and changes are saved	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The job is not processed and customers value type is not changed.	
Date:09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 14/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case Testing: First reminder

Use Case ID: 32	**Use Case Name: firstReminder**
Test number: 17	
Objective: Ensure the system generates the correct reminders to be sent to the correct individuals at the appropriate time and the user has the ability to view these late payments.	
Set Up: User is logged in as an office manager and views customers with late payments. The office manager proceeds to print the reminder letter, which is then sent to the specific customer it has been associated for.	
Expected Results: 1. First reminder letter is delivered to a specific customer 2. Office manager has the ability to print and send the letter. 3. Reminder table is updated.	
Test: 1. Logged in as an office manager with username: bob.ross and password: 123 2. User selects 'Manage late payments' from respective screen. 3. User selects the appropriate customer from the reminder table 'First Reminder' and clicks 'view' button. 4. Information is displayed regarding the late payment for that specific customer. 5. User presses the button 'Print NEW first reminders' and the First reminder copy is presented to allow the user to print. 6. First reminder is printed and issued to the customer. 7. Table is successfully updated	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The table is updated and the reminder letter was successfully printed and sent. No other customers were altered in the table.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 16/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case ID: 32.1	**Use Case Name: paidOnTime**
Test number: 18	
Objective: Test the alternative flow whereby if the customer has paid, the database should update to ensure that the payment has been accepted and the customer is not listed on the late payments table.	
Set Up: Similar to the main flow test case, the user is logged in as an office manager and selects the 'manage late payments' button from its respective screen. The table should be updated with the relevant customers who are still yet to pay. If by chance, a customer has paid on time and is now on the table falsely, the customer should be removed, and the table should be updated.	
Expected Results: 1. User clicks view for the appropriate late customer 2. User clicks prints 3. Error is presented 4. Table is updated 5. Reminder is not issued to the customer	
Test: 1. Continuing on step 3 of the test case (firstReminder) 2. After clicking 'view' for the relevant customer, the relevant information is present 3. User attempts to click the 'print' button for the reminder 4. Error occurs and failure to print the reminder 5. The page is refreshed and the table is updated 6. Reminder is not sent or submitted to the customer 7. Customer account is no longer available on the late payment table	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The table quickly updated and removed the customer from there. The reminder was therefore not issued to any customers.	
Date: 04/04/19	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 16/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case Testing: Set Cust Value

Use Case ID: 36	**Use Case Name: setCustValue**
Test number: 19	
Objective: Test the main flow to see if the system will allow a normal customer to be upgraded to a valued customer whilst retaining, updating and keeping all previous details e.g. name, address, jobs etc…	
Set Up: As only the office manager has the privileges to do this, the office manager's account will need to be set up and logged into. An existing customer will be selected and an attempt to upgrade them to a valued customer will be made.	
Expected Results: 1. Unvalued customer wishes to upgrade 2. Customer is set to be valued 3. Customer is eligible for discounts 4. Customer jobs and details aren't affected	
Test: 1. User is logged in as an office manager with username: bob.ross and password: 123 2. User clicks 'View customers' and navigates through to the relevant page 3. User selects the customer "Timmy" 4. User is given the option to edit customer details for "Timmy" if required. 5. User then checks the "Valued" box to make this customer a valued customer by ticking the box 6. User saves changes and details are updated on the table	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. The system was used to promptly edit the details of a customer to upgrade them to a valued customer with no issues. The other customer values remained unaffected as well as the customer is eligible for a discount plan.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 14/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case ID: 36.1	**Use Case Name: custNotExist**
Test number: 20	
Objective: Test the alternative flow of setCustValue whereby the customer has recently been deleted or edited out, however, the list has not been updated to show the customer has been deleted. So, the customer is still present within the table.	
Set Up: Similar to the main flow test case, the user is logged in as an office manager. The deletion of a customer from the database should be updated on the table for the system to process.	
Expected Results: 1. System does not acknowledge customer and customer details cannot be changed 2. The customer table list updates successfully	
Test: 1. Continuing on step 3 of the test case (setCustValue) 2. User selects the customer "Timmy" from the customer table 3. The customer "Timmy" is selected to edit 4. User checks and is given the opportunity to change the valued option and clicks 'Save changes' 5. System fails to acknowledge customer and changes are not saved 6. Table is refreshed and list is updated.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Error was prompted, value was not set and the table was updated after the error message.	
Date: 09/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date:16/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: Backup Database

Use Case ID: 45	**Use Case Name: backupDatabase**
Test number: 21	
Objective: Test the main flow to ensure that backup functionality is fully functioning. This also tests whether the contents backed up are correct, safe and usable.	
Set Up: User is logged in as office manager and there are new changes to the data e.g. customer table. The office manager should have a backup location ready with a sufficient size and allocation to back up the new changes.	
Expected Results: 1. Enter new changes to the table 2. New changes are successfully stored and backed up 3. New changes are accessible	
Test: 1. Logged in as an office manager with username: bob.ross and password: 123 2. User clicks 'Admin' button from their respective screen 3. User adds staff if necessary and selects 'Back up now' 4. User checks to see if current table of customers are backed up and updated 5. User also checks if previous back up is changed to current date 6. Database is successfully backed up	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. User successfully backed up BAPERS, and checked to make sure the most recent back up is correct. The last back up time has also changed on the system.	
Date: 10/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 14/04/18	**Tester: Parmveer Johal**
Results: Passed	

Use Case ID: 45.1	**Use Case Name: memoryFull**
Test number: 22	
Objective: Test the alternate flow to the main use case whereby the storage medium runs out of memory and meaning the database can longer be backed up. It makes sure that BAPERS does not register a back up, or corrupt existing backups.	
Set Up: Similar to the first one, user is logged in as an office manager and takes note of the new customer tables. The backup location however is changed to a full storage space.	
Expected Results: 1. Error is presented 2. Last backup date is not changed 3. Previous backups are not harmed	
Test: 1. Continuing on step 3 of the test case (backupDatabase) 2. Office manager logs into the system (username: bob.ross and password: 123) 3. User selects Admin page 4. User adds new staff details and clicks 'Add Staff' button. 5. User selects 'Back up now' 6. System is backed up and the file (2018-04-13.sql) is shown below. 7. User selects 'Back up now' again 8. Back up error is presented, stating 'Memory is full' and back up does not complete. 9. Last backup date does not change. 10. User views previous backups and they are fully functioning (no file corruption or deletion)	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Back up is not completed, last backup date does not change and previous backup files are not corrupted or deleted.	
Date: 10/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 16/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case ID: 45.1	**Use Case Name: memoryFull**
Test number: 22	
Objective: Test the alternate flow to the main use case whereby the storage medium runs out of memory and meaning the database can longer be backed up. It makes sure that BAPERS does not register a back up, or corrupt existing backups.	
Set Up: Similar to the first one, user is logged in as an office manager and takes note of the new customer tables. The backup location however is changed to a full storage space.	
Expected Results: 1. Error is presented 2. Last backup date is not changed 3. Previous backups are not harmed	
Test: 1. Continuing on step 3 of the test case (backupDatabase) 2. Office manager logs into the system (username: bob.ross and password: 123) 3. User selects Admin page 4. User adds new staff details and clicks 'Add Staff' button. 5. User selects 'Back up now' 6. System is backed up and the file (2018-04-13.sql) is shown below. 7. User selects 'Back up now' again 8. Back up error is presented, stating 'Memory is full' and back up does not complete. 9. Last backup date does not change. 10. User views previous backups and they are fully functioning (no file corruption or deletion)	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Back up is not completed, last backup date does not change and previous backup files are not corrupted or deleted.	
Date: 10/04/18	**Tester: Anil Yagliyurt**
Results: Passed	
Date: 16/04/18	**Tester: Mayur Depala**
Results: Passed	

Use Case Testing: reactivateDefault

Use Case ID: 30	**Use Case Name:** reactivateDefault
Test number: 23	
Objective: To test the account functionality within the BAPERS system. This tests the use case, and reactivates, approves/declines and finalises the account of a customer in BAPERS. It also makes sure that main aspect of the use case is applied i.e. default account and suspend account.	
Set Up: Customer exists within the system and has ability to reactivate default account or suspend with BAPERS. Customer is set up to choose between the two. Once selected, customers are displayed back within the system or suspended and relevant privileges are applied..	
Expected Results: 1. Customer is either suspended or reactivated as default in the system 2. Account details are recorded and stored in the table 3. System is updated and relevant privileges are applied	
Test: 1. Office Manager logs into system (user: bob.ross and password: 123) 2. Receptionist screen is selected 3. User selects customer they wish to re-activate 4. User suspends customer or default account is created 5. Receptionist saves details and updates system	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Pass	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Pass	

Use Case Testing: respondToEnq

Use Case ID: 18	**Use Case Name:** respondToEnq
Test number: 24	
Objective: To test the enquiry functionality within the system. This tests the use case and requests and addresses the job and receptionist of the enquires in BAPERS. It also makes sure the main aspects of the use case are applied, i.e. processing the job, updating status, transfer job location etc.	
Set Up: The technician within the system has one or more enquiry to respond to with BAPERS. The technician updates and addresses appropriate information to receptionist about the job and the relevant details are entered and recorded	
Expected Results: 1. Job enquiry is sent to technician 2. Technician updates and responds appropriately 3. Receptionists records details on table 4. Job process carries on as usual	
Test: 1. Office Manager logs into system (user: bob.ross and password: 123) 2. Technician screen is selected for user to view 3. Technician views relevant enquiries and responds appropriately 4. Receptionist receives notification 5. System updates enquiry on job	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	

Use Case Testing: assignJobID

Use Case ID: 8	**Use Case Name:** assignJobID
Test number: 25	
Objective: To test the correct job ID number used for the appropriate job within BAPERS. This tests the use case and addresses and updates the customer's job ID number in BAPERS. It also assigns the main aspect of the use case that are applied when it accept the job, i.e. assigning the correct job ID when the job is marked as urgent	
Set Up: The receptionist accepts job within the system and assigns a relevant job ID to the customer. The receptionist updates each task and job completed and sets up the job IDs complete. The relevant details are stored and updated.	
Expected Results: 1. Job is accepted by receptionist 2. Job is assigned to the customer 3. Receptionist records details on table	
Test: 1. Receptionist logs into system (user: john.smith and password: 999) 2. Receptionist views and selects customer 3. Appropriate job ID is created for customer and the job	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	

Use Case Testing: setAccountDetails

Use Case ID: 42	**Use Case Name:** setAccountDetails
Test number: 26	
Objective: To test the functionality for appropriate accounts within the system. This tests the use case and sets and updates the account details for the staff within BAPERS. It also makes sure the main aspects of the use case are applied, i.e. creating the staff.	
Set Up: The office manager within the system has the ability to set up an account for a staff. The office manager creates, adds and sets up appropriate staff details/accounts. The system stores the account for the staff to use.	
Expected Results: 1. Office manager opts to create staff 2. Staff details are inputted 3. Account is created and set for use 4. Records are updated on table	
Test: 1. Office Manager logs into system (user: bob.ross and password: 123) 2. User clicks on 'Admin' page 3. User opts to include relevant details for staff 4. Account details are saved and updated	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	

Use Case Testing: printReminder

Use Case ID: 29	**Use Case Name:** printReminder
Test number: 27	
Objective: Test the main flow to ensure that the various types of print reminders can be successfully printed for customers who have failed to pay within a certain time period. The test will also ensure that only the office manager has priority to be able to do so, and an update is made to the system to inform that a reminder has been printed.	
Set Up: Office manager logs into the system to view late payments. Specific customer is selected, and an exact payment reminder, is printed.	
Expected Results: 1. Customer fails to pay for job within provided date. 2. System processes information and produces a print reminder. 3. Print reminder is printed and sent to customer.	
Test: 1. Office manager logs into the system (username: bob.ross and password: 123) 2. User selects Late payments page 3. User views late payments 4. User selects 'first payment reminder' 5. User presses 'print' button 6. Payment reminder is printed	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	

Use Case Testing: deadlineExceedAlert

Use Case ID: 15	**Use Case Name:** deadlineExceedAlert
Test number: 28	
Objective: Ensure the system produces a deadline alert to inform the technician that a job has now gone overdue, and therefore exceeded its deadline day. This test will also enable the technician to update the table once the job has been completed.	
Set Up: Technician logs into the system and views details of jobs which are still to be completed. An alert message is shown on the screen, regarding a specific job being overdue and by how many days.	
Expected Results: 1. Tasks for job have not been completed within date. 2. Job has not been processed and exceeds deadline. 3. Error presented informing of deadline exceed.	
Test: 1. Office manager logs into the system (username: bob.ross and password: 123) 2. Office manager accesses Technician page. 3. Alert is produced informing 'Job 34 is Overdue, please check ASAP' 4. User views details of incomplete job. 5. User marks job as complete. 6. User presses 'Update' button to save new changes.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	

Use Case Testing: transferJobLocation

Use Case ID: 17	**Use Case Name:** transferJobLocation
Test number: 29	
Objective: Test the main flow to ensure that a job is successfully transferred to the right department once it has been completed, and ensure it is transferred within deadline date. The test will also ensure that the Technician and office manager are the only individual who have priority to do so, and the system is updated successfully when this process is complete.	
Set Up: Technician logs into the system and updates the job as complete once it has been finished. Specific job is then selected and transferred to the correct location.	
Expected Results: 1. Job has successfully been completed. 2. Job is transferred to the correct location (Department) 3. Technician updates this onto the system.	
Test: 1. Office manager logs into the system (username: bob.ross and password: 123) 2. Office manager accesses Technician page. 3. User selects job from table and marks as updated. 4. User selects 'Available to be transferred to department' 5. User selects 'Update' button to save changes	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	

Use Case Testing: latePaymentDetect

Use Case ID: 33	**Use Case Name:** latePaymentDetect
Test number: 30	
Objective: Ensure the system is able to produce a notification alert informing the receptionist that the payment received is over the expected deadline, and therefore committing an additional sub charge to be applied over original payment.	
Set Up: Receptionist logs into the system and accepts card/cash payment for Job which has exceeded payment date, system informs user of late payment and Receptionist applies additional sub charge.	
Expected Results: 1. Late payment is received, which surpasses original deadline. 2. Notification produced to inform of deadline exceeding. 3. Allows user to apply additional sub charge over original payment. 4. New updated payment is requested from customer.	
Test: 1. Receptionist logs into the system (username: john.smith and password: 999) 2. Receptionist selects job (ID: 52) and proceeds to payment. 3. User receives alert informing 'Late payment, please apply additional sub charge' 4. User selects 'Sub charge' button and enters '£5' 5. Receptionist selects 'Card payment' 6. System produces new total. 7. User pays for Job. 8. Payment goes through successfully with updated changes.	
Test Records: (e.g. all parts run successfully, or what executed incorrectly) All parts successful. Receipt printed, customer updated, jobs updated and details are recorded in system.	
Date: 23/04/18	**Tester: Parmveer Johal**
Results: Passed.	
Date: 23/04/18	**Tester: Mayur Depala**
Results: Passed.	

Non-Functional Testing

Non-Functional Testing has also been included within this documentation to:

Test the software application based on the way it operates and to evaluate the readiness of the system, rather than just checking the specific behaviours through it. Hence, allowing us to demonstrate how well the product behaves to specific circumstances as opposed to simply stating what the product does. The Non-functional testing consists of 2 important requirements of security and reliability Furthermore, accurate descriptions, rationale, source, fit criteria, priority, conflicts, history, and supporting material has been included in full detail to clarify and demonstrate accurate information for Users.

Non-Functional testing has been shown below:

Security

Non-functional testing has been an absolute necessity to evaluate the readiness and accuracy of BAPERS systems in terms of security. Security was highlighted as a key aspect and was therefore chosen as one of the key NFR requirements. Security is responsible for preventing unauthorised access being made to the system and thus result in keeping customer data confidential and secure from external threats. In order to ensure efficient and quality Security, BAPERS system has a highly secure login page which only allows access to members of the system. The system only allows individuals who produce the correct login credentials authorisation into the system and denies unauthorised people from getting into the system. Furthermore, BAPERS has taken high measures to identify any sort of suspicious activities regarding individuals who try to make unauthorised access to the system. The system informs its creators when numerous incorrect login details are entered into the login page, allowing the creators to identify any sort of suspicious activities and prevent unauthorised access being made into the system. However, despite this security being in place there is still a chance of the system being infected by malicious malware. Therefore, further measures have been taken to ensure the system remains fully functioning and data is not corrupt or lost throughout the process. In order to do this, various database backup versions have been created and stored on an external server to ensure data is available even if deleted or corrupted by an external threat. Furthermore, efficient training has been provided to members of BIPL as a contingency play, so they are aware on how to deal effectively with various unforeseen circumstances. Lastly, due to security being an aspect of fundamental importance. It is important to ensure the system is fully protected from malicious malware to a high extent, in which case a professional hacking team is desirable to test the system to a full scope.

Requirements ID:	**1**	**Requirements Type:**	**NFR**	**Event / Use Case #**	N\A
Description:	The BAPERS system shall only allow specific users to access and gain authorisation into the system.				
Rationale:	Authorisation shall only be allowed by users of the system. Whereby, the correct login credentials (User ID & Password) permit this, and Incorrect details produce denial. Therefore, preventing unauthorised access into the system				
Source:	The initial statement of requirements has come from an Interview with the consultant and BAPERS documentation.				
Fit Criteria:	The system shall effectively process whether the correct login details have been entered to allow access into the system and incorrect details do not allow authorisation. Testing with various attempts of different login credentials and ensuring if correct and accurate, produces desired result of logging into the system. If Incorrect, provides user with ability to re-attempt until successfully logged in with correct details.				
Customer Satisfaction:	**5**	**Customer Dissatisfaction:**	0		
Priority:	Important requirement	**Conflicts**	No conflicts are involved with this requirement.		
Supporting Material:	No supporting material involved.		Volere Source: BAPERS		
History:	No previous history has been entitled, therefore it is a New Requirement.				

Reliability

Reliability is another Non-functional testing method used to ensure that BAPERS system works effectively during certain conditions for a specific period of time. We have accounted Reliability as one of the NFR, as it is responsible for reducing the frequency of failure when actual users interact with the system and maintaining a system which is able to cope and work effectively whilst processing large amounts of data. The BAPERS system has been tested with the use of a range of data, starting from small amounts leading to medium amounts of data. However, in order to ensure the system remains functioning efficiently and in a reliable manner it is recommended that the system is tested with large quantities of data. So that when BAPERS expands and the organisation grows within the next couple of years, leading to a potential increase in customer orders, new customers and new employees. The system is still able to cope and work effectively with processing large amounts of data and more complex data which will come from the daily use of this system and its growth. Furthermore, unexpected naming conventions for data must be extensively tested to ensure the system is able to successfully handle tasks of anticipated magnitude. In order to achieve this, it is recommended that a solution is developed which will pour the test database automatically with various types of data to test the system's ability of effective handling.

Requirements ID:	**2**	**Requirements Type:**	**NFR**	**Event / Use Case #**	N/A
Description:	To Test if the system can perform as expected under specific conditions, for a specified period of time.				
Rationale:	It needs to be clearly evident that the system is reliable because it is paramount the client using the system, has confidence that the system will perform the way they expect it to. Also, to not have to worry about problems such as unexpected loss of data.				
Source:	The initial statement of requirements has come from an interview with the consultant and the BAPERS documentation.				
Fit Criteria:	Once the system has been developed, test inputs/data given by the client will be used and passed into the system. Observing the performance of the system when adding/deleting/upgrading a customer, printing reports, adding a new job into the system will highlight any existing faults in the system. Depending on how the system performs, will be a clear indicator to if the initial requirements of the system have been fully met.				
Customer Satisfaction:	**5**	**Customer Dissatisfaction:**	0		
Priority:	Important requirement	**Conflicts**	No conflicts are involved with this requirement.		
Supporting Material:	No supporting material involved.	Volere Source: BAPERS			
History:	No previous history has been entitled, therefore it is a New Requirement.				

Requirements ID:	**3**	**Requirements Type:**	**NFR**	**Event / Use Case #**	N/A
Description:	The BAPERS system shall avoid losing and mistreating customer material to avoid confusion under normal or urgent jobs.				
Rationale:	Too unreliable, BAPERS will cause delays and will affect the confidence of staff and customers in the operation by the job.				
Source:	Interview with the Office Manager of BIPL				
Fit Criteria:	The system shall effectively process whether the correct job priority has been chosen. Testing the BAPERS system the equivalent of regular, normal and urgent job operations (new customers, special instructions, reservations etc) and observe flexible scheduling during this period.				
Customer Satisfaction:	**5**	**Customer Dissatisfaction:**	0		
Priority:	An essential requirement	**Conflicts**	None		
Supporting Material:	None		Volere Source: BAPERS		
History:	A new requirement				

END OF DOCUMENTATION

ABSOLUTE CONSULTANCY

SIMPLE SOLUTIONS FOR COMPLEX CONNECTIONS