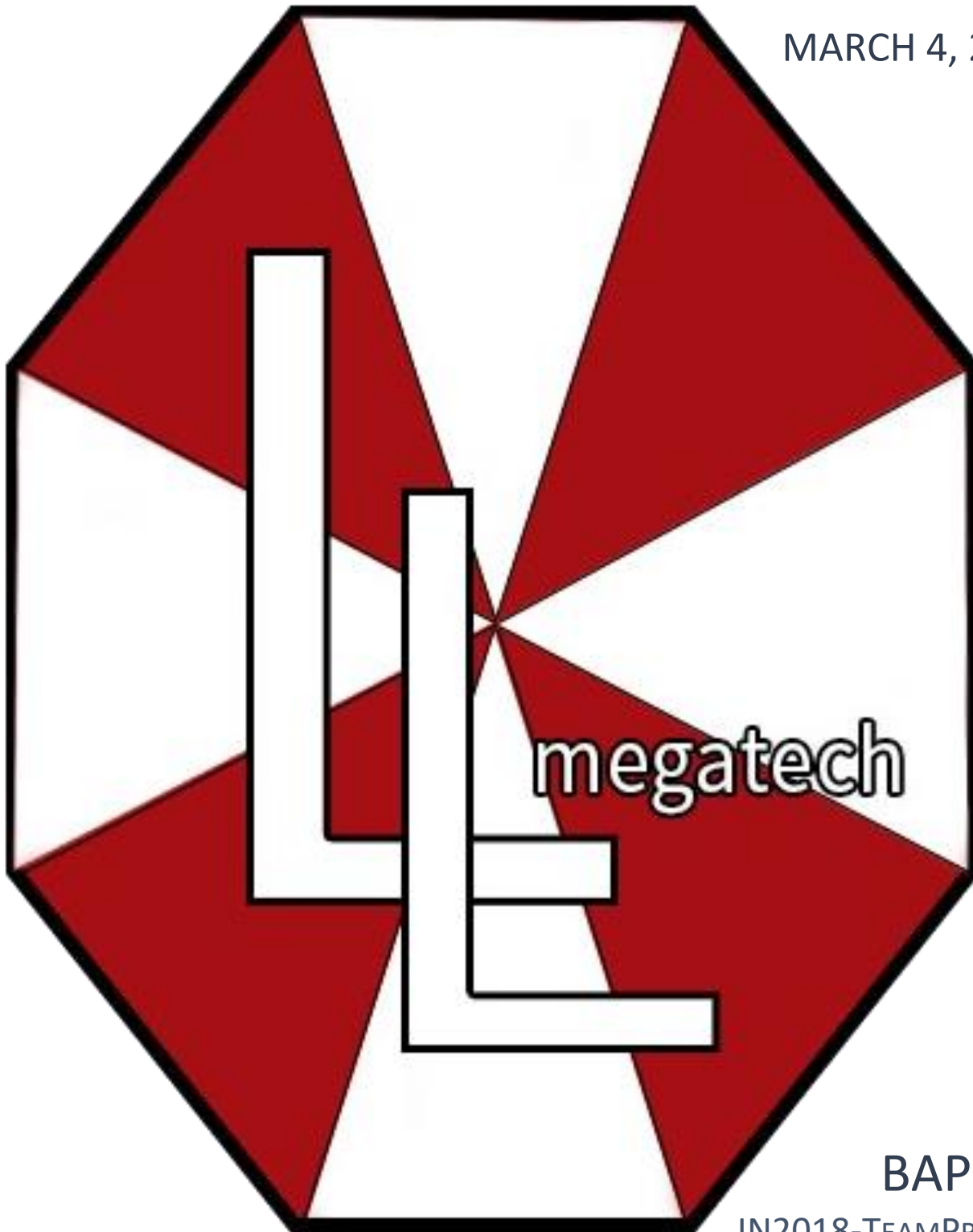


MARCH 4, 2018



megatech

BAPERS

IN2018-TEAMPROJECT

GERDA MOROCKAITE & CO.

LIU LIU MEGATECH

[Company address]

Table of Contents

1	Preface	4
1.1	Purpose and scope of the document (what is it and what it is not)	4
1.2	intended audience and Reading suggestions	4
1.3	History of the Document.....	4
2	Introduction	4
3	Glossary.....	5
4	System requirements.....	6
4.1	Purpose and Scope of the System.....	6
4.2	UML use-case model	6
4.3	Use Case Diagram	6
5	System Models	7
5.1	Use Case Diagram	7
6	Design Class.....	15
6.1	Package Diagram.....	15
6.2	Simplified Design Class Diagram	16
6.2.1	16
6.3	Detailed Design	17
7	Use Case Indexing	18
	ADMIN.....	20
	ACCT	22
	CUST	24
8	Use Case Specifications.....	26
	Record Payment.....	26
	Generate Letter.....	27
	Create User	28
	Automatic Backup	28
	Update Existing Task	29
	Upgrade Customer Account.....	30
	Print Late Reminder	31
	Login.....	31
	Generate Reports.....	32
	Update Job Status	33
9.	Systems Evolution	34
10.	Appendices.....	34
11.	Hardware and Software configuration	34

Hardware minimal requirements.....	34
Operating Systems:	34
Additional Software	34
12. Database	35
12.1 SQL schema	35
12.2 Reports.....	51
12.2.1 Summary Performance Report	51
12.2.2 Individual performance report.....	52

1 Preface

1.1 Purpose and scope of the document (what is it and what it is not)

This is a software requirements document for the BAPERS Software and covers the functional and non-functional requirements with various implementation constraints defined in the initial statement of the requirements by the marketing of LIU LIU TECH

This Document explains what the system shall do but not how the functionality can be implemented. Various models are used to convey in more detail to system.

1.2 intended audience and Reading suggestions

The staff at BAPERS, Software Developers and Mr Lancaster

1.3 History of the Document

This is a new document, v 1.0

2 Introduction

BIPL is a photographic laboratory that deals with the work of professional photographers by providing a number of standard jobs

The jobs are made up of one or more standard tasks that are performed in their professional laboratory operated by the staff. Special requests can be added to the job as well as the “urgent status” which is a request by the customer for the job to be completed faster than the original deadline which is usually, within 24 hours. The usual “urgent job” must be completed within 6 hours but the customer can also choose 3 hours for a 100% surcharge. The customer can choose an even faster job but at higher rates.

Currently, **BIPL** can perform 30 standard tasks, each task being uniquely identified and carried out in a specific location in the laboratory.

Each task has an ID, a description, a location in the laboratory, a price and a duration. Once a task is completed, the product is placed in a specific shelf, from where it can be taken for further processing or given to the customer, if completed. Each task has a specific shelf slot.

Most **BIPL** jobs are priced below 400 GBP, operating at a high-turnover, low profit margin level, with hundreds of jobs operating or pending at the same time.

Whenever a new job is placed, it has to be associated to an existing customer or a newly created one.

3 Glossary

BAPERS:

Actor

BIPL

CUST

ACCT

ADMIN

4 System requirements

4.1 Purpose and Scope of the System

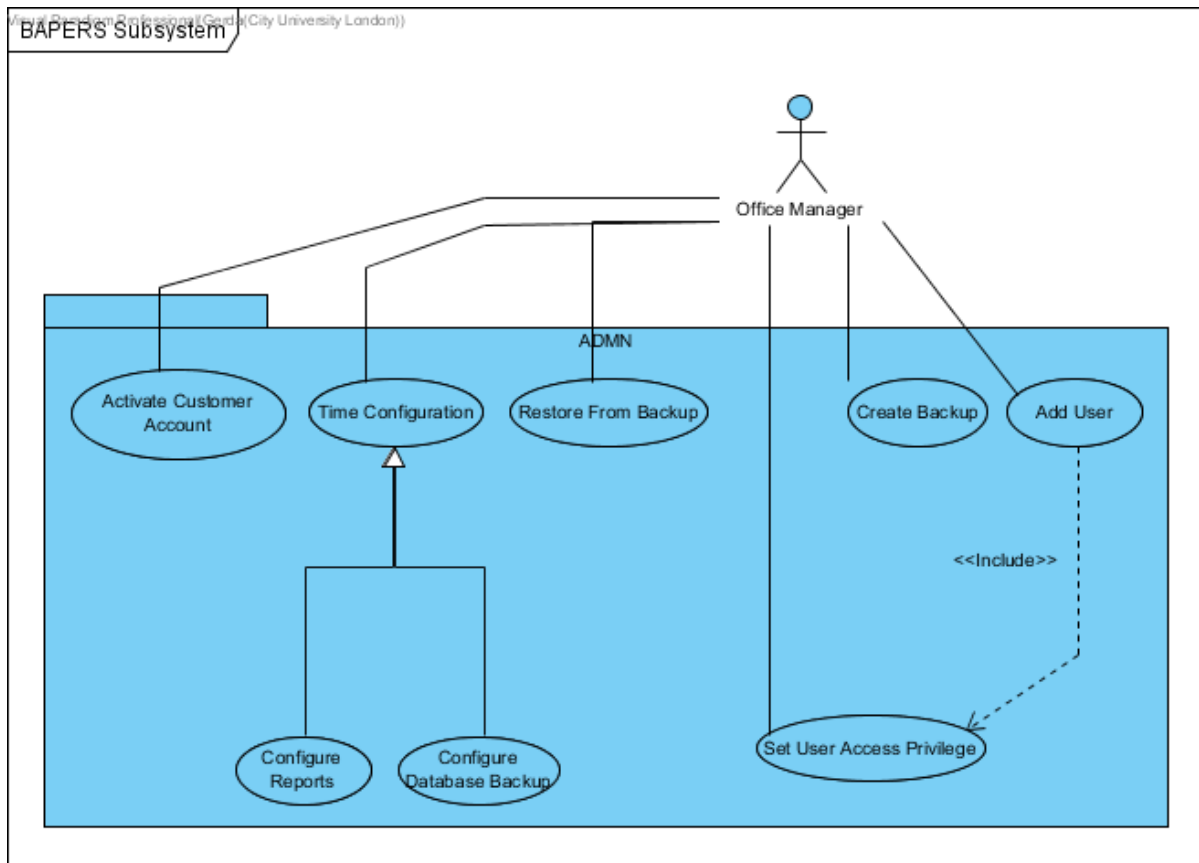
4.2 UML use-case model

4.3 Use Case Diagram

5 System Models

5.1 Use Case Diagram

In the following diagrams, it has been created in packages to represent the different subsystems of BAPERS. We have 6 packages which include ADMIN, CUST, PROC, ACCT, PAYMENT and REPORTS.



ADMIN

Activate Customer Account: Reactivates customer account marked as 'in default'.

Time Configuration: Configures the periodicity of automatic functions.

Configure Reports: Changes the frequency of reports.

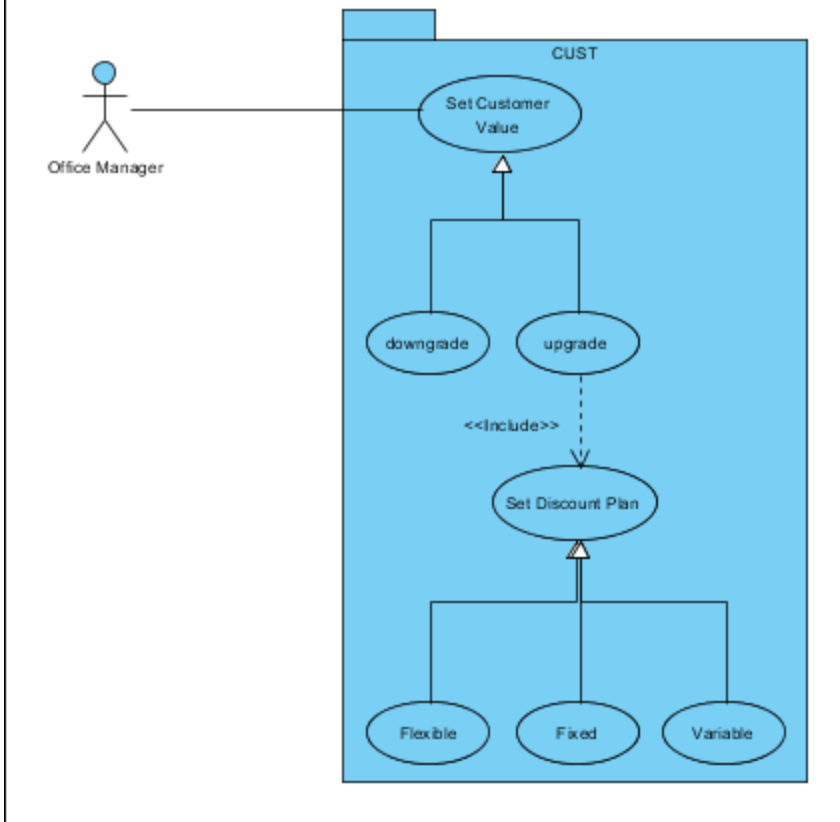
Configure Database Backup: Changes the frequency of automatic backups.

Create Backup: Manually creates/overwrites system backup.

Add User: Creates a user account for BAPERS.

Set User Access Privilege: Sets up user access privileges.

Restore from Backup: Restores BAPERS from an existing backup.



CUST

Set Customer value: office manager chooses to set a customer to a valued customer

Downgrade: office manager can choose to downgrade a customer's valued status

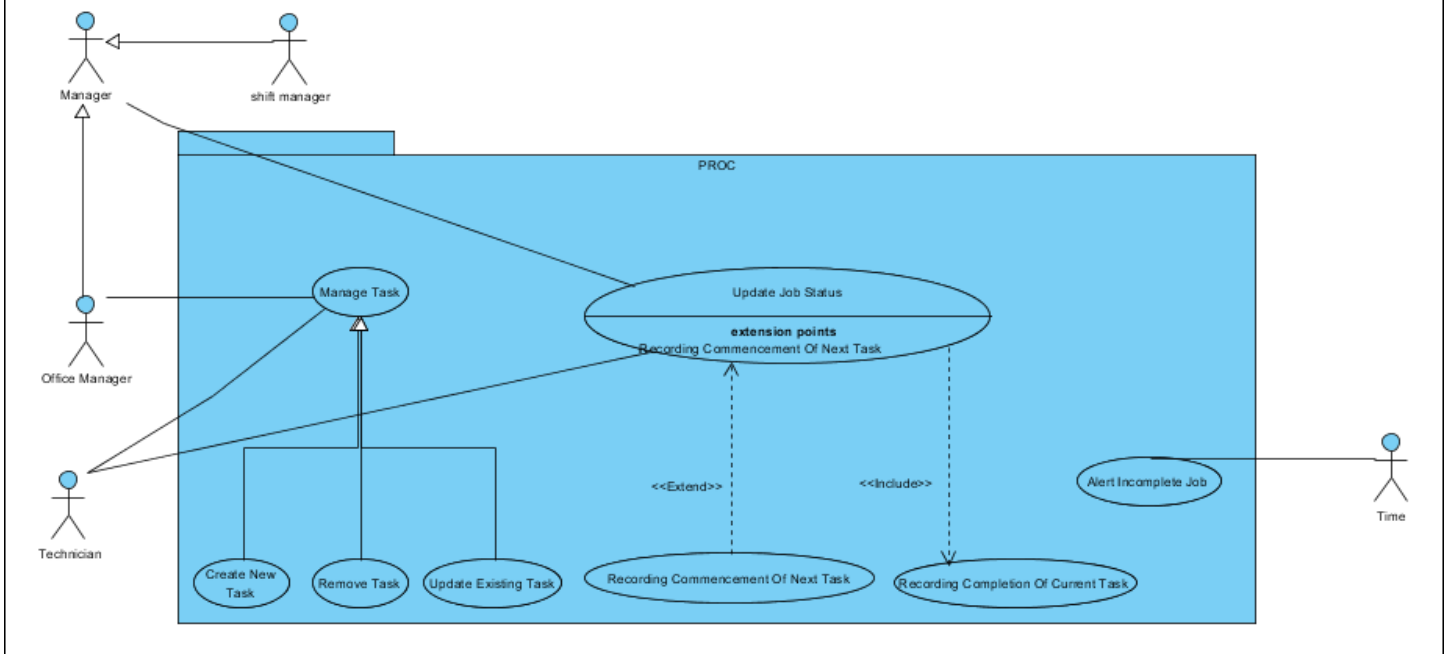
Upgrade: office manager can choose to upgrade a customer's valued status

Set discount plan: when a customer is upgraded, the office manager has the option to set up a discount plan according to how many Jobs/tasks a customer has placed.

Flexible: this use case defines the percentage of the discount that depends on the values of the jobs by the same customer accumulated within a calendar month.

Variable: this use case defines the percentage of the discount that is set for each task and may vary between the tasks. The value of this discount is calculated and deducted from the value of the job at the time of accepting the job.

Fixed: this use case defines the same percentage of discount given to the valued customer for each job. The value of this discount is calculated and deducted from the value of the job at the time of accepting the job.



PROC

Manage Task: Edit the tasks that exist in the system, add or delete them

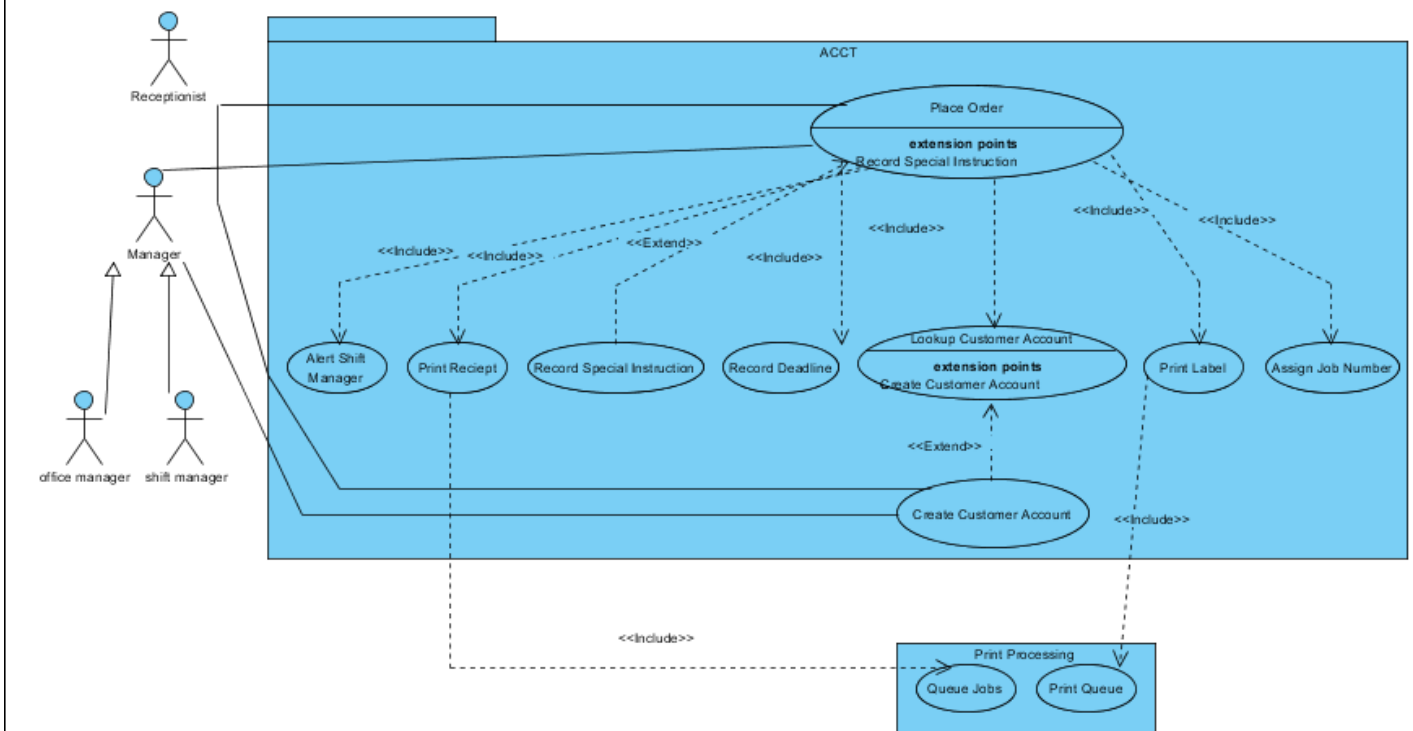
Create New Task: Add a new task to the list of existing tasks

Remove Task: Delete a task from the list of existing tasks

Update Existing Task: Update any changes made to an existing task

Update Job Status: Update the status of an ongoing job and the tasks it has performed or are about to be performed. The commencement and completion of tasks are recorded here

Alert Incomplete Job: Alerts the customer if the job they placed cannot be completed of the job at the time of accepting the job.



ACCT

Place Order: The actor (Managers or Receptionist) placing the order on behalf of the customers.

Alert Shift Manager: To alert the shift manager there is a new job arrive and should be process.

Print Receipt: Print a receipt for the customer about the job he/she order.

Record Special Instruction: Record additional requirement from the customer.

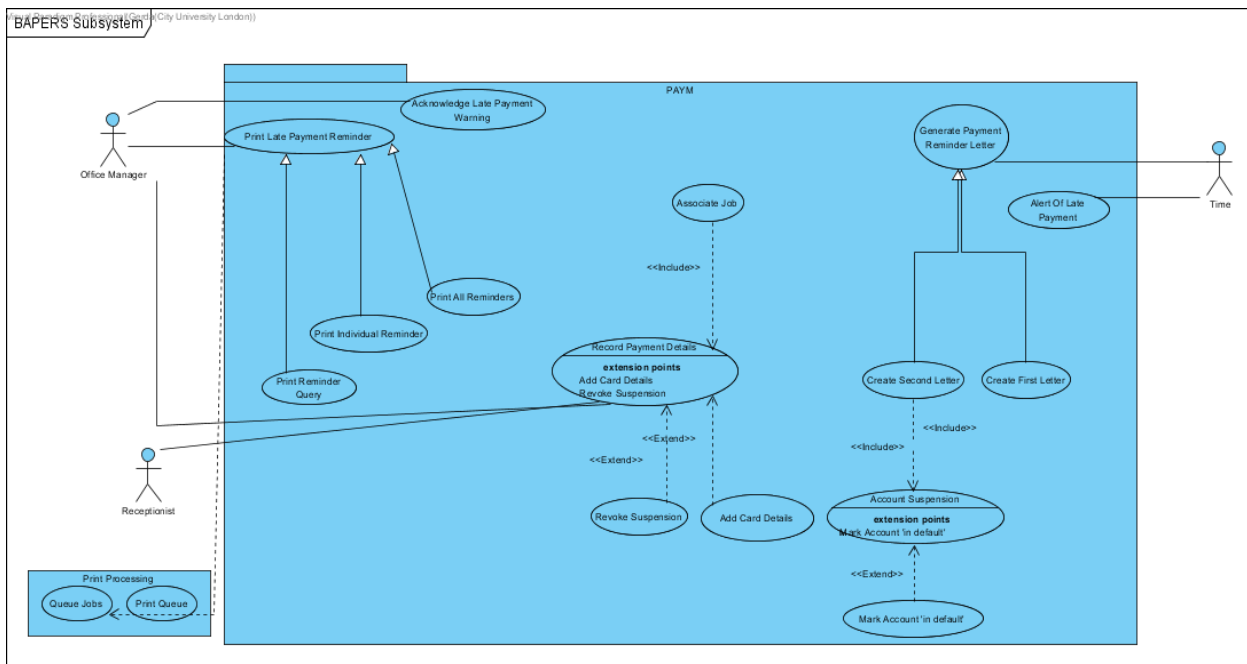
Record Deadline: Record the required job end time.

Lookup Customer Account: Search the customer account in order to identify the identity of the customer.

Create Customer Account: **creating** a new instance of customer account due to the very first order from the new customer.

Print Label: Print the label for materials required for job.

Assign Job Number: Generate a number for the job which is unique compare to other jobs.



PAYMENT

Print Late Payment Reminder: when a late payment is made, a late payment reminder is created which can then be printed by the office managers discretion.

Print Reminder Query: this use case gives the option/functionality to the office manager to print whichever late payment reminder they want.

Print Individual Reminder: This prints the late payment reminder for a single job. It prints the latest job reminder.

Print All Reminders: Prints all the late reminders that have been stored.

Acknowledge Late Payment Warning: the office manager gets alerts of late payments to his/her computer, and the office manager acknowledges these late payment warnings.

Record Payment Details: once the customer pays for the completion of the job payment method and details are recorded.

Revoke Suspension: when a customer account is suspended, and a payment is made from that account. The suspension on that account is revoked and the account is reactivated automatically.

Add Card Details: if the customer is paying with card, card details are recorded such as expiry date, type and last 4 digits.

Associate Job: each payment needs to be associated with a job. This checks the payment with the job ensuring the payment is for the right job.

Generate Payment Reminder Letter: triggered automatically, when valued customers are unable to clear the outstanding balance, a late payment reminder letter is generated.

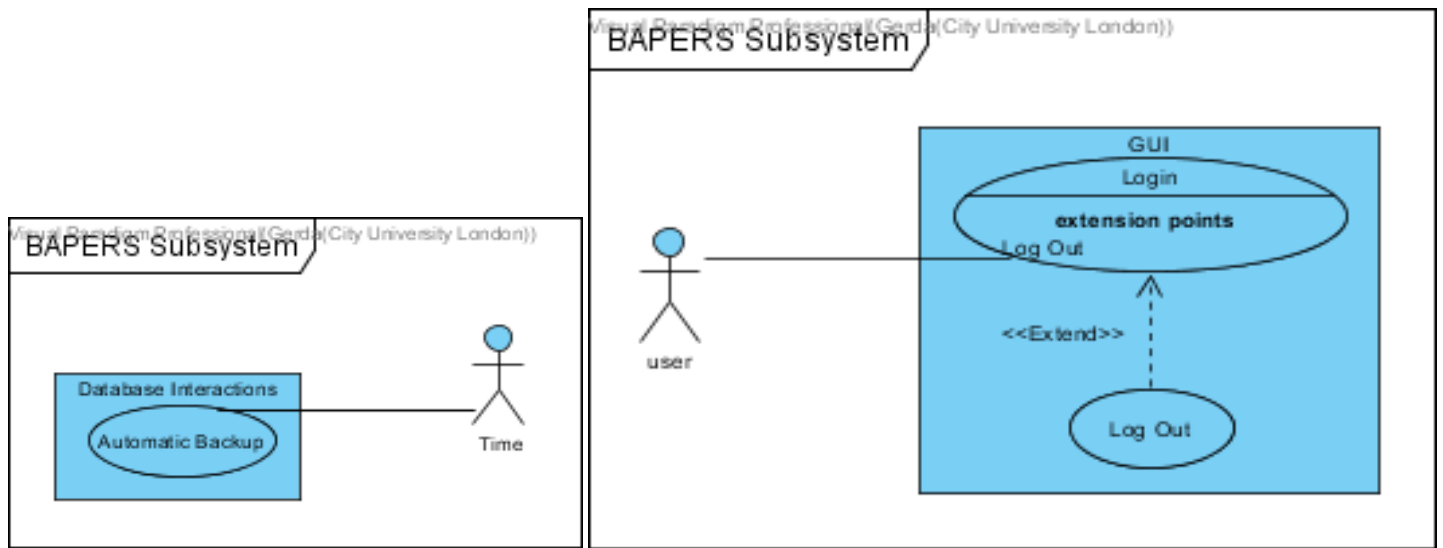
Create First Letter: generates the first late payment letter.

Create Second Letter: generates the second late payment letter.

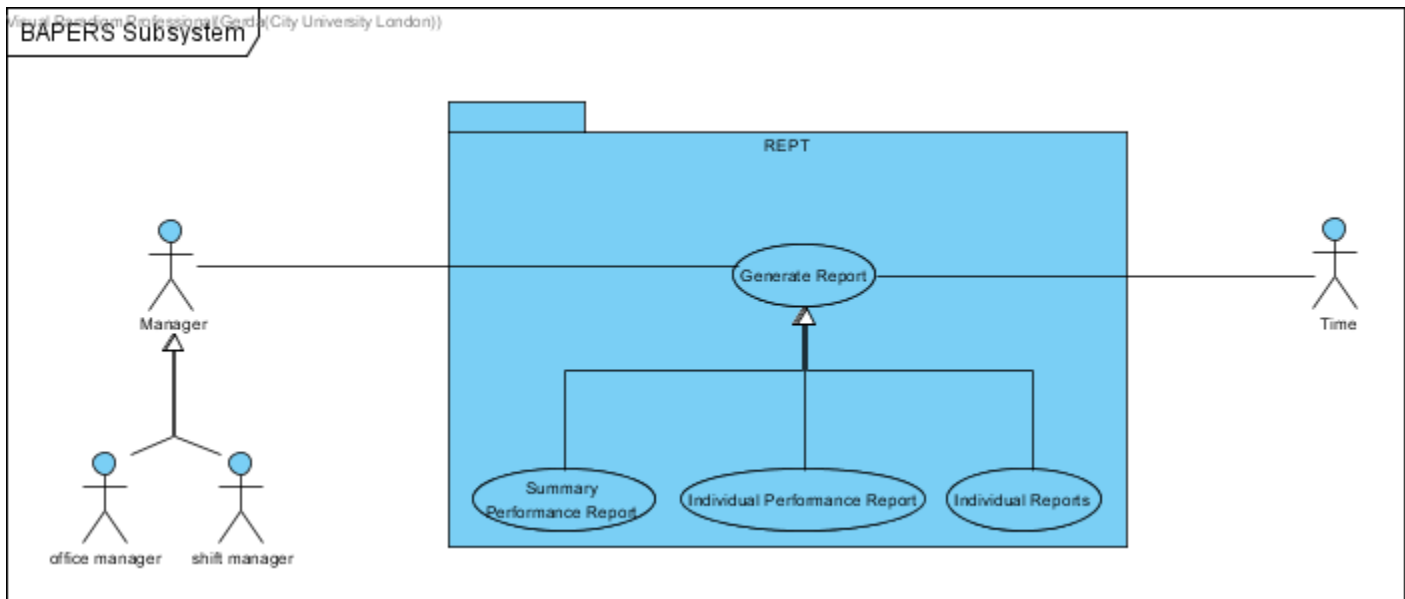
Account Suspension: after the second letter is created the account associated with it is suspended automatically.

Mark Account "in default": a month after the second letter has been sent and the outstanding balance has not been cleared the system marks that account as "in default" automatically.

Alert of Late Payment: automatic alerts of late payments are shown to the office manager as pop-up windows in 15 minutes intervals.



need to add description for these two



REPORT

Generate Report: Generate a report according to the type according to the user (Office Manager) chosen.

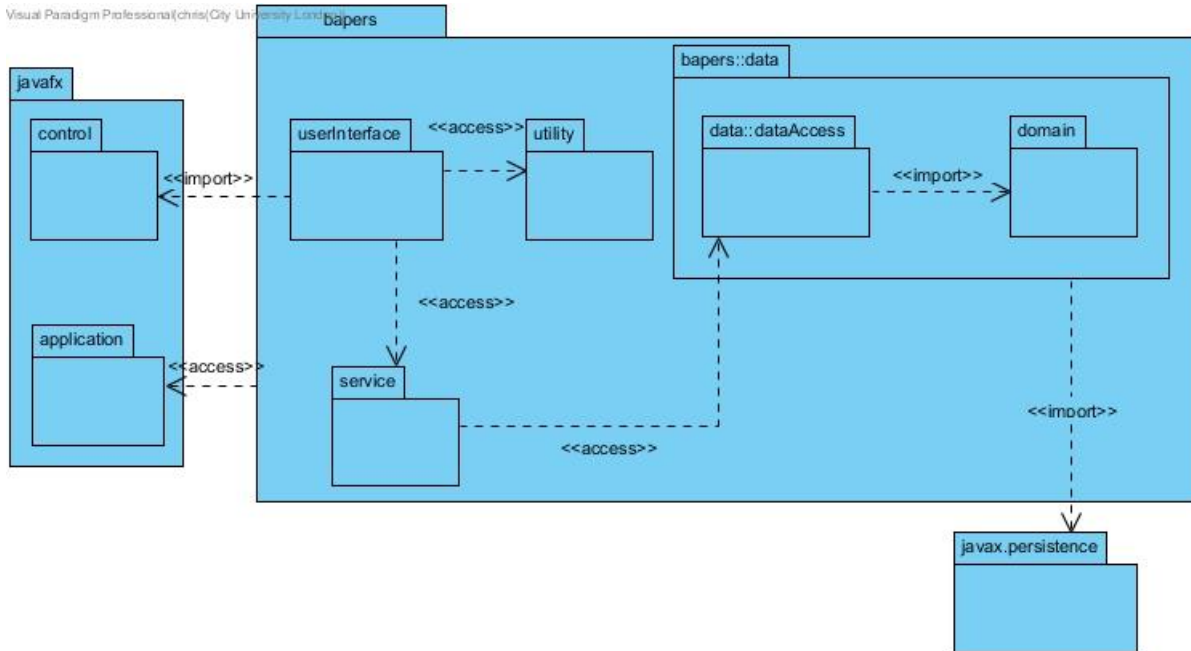
Summary Performance Report: Generate the report about summary performance with the data's stores in the database.

Individual Performance Report: Generate the report about individual performance with the data's stores in the database.

Individual Report: Generate the report about individual with the data's stores in the database.

6 Design Class

6.1 Package Diagram



DB connectivity is modelled with the expectation of using the eclipselink library, which is an implementation of JPA, hence the import to `javax.persistence`. I have simplified the diagram by not showing the persistence unit, as this will be autogenerated by the chosen IDE, making it redundant to show it in the class diagram.

The persistence unit will need to include the following property: `<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/bapers?zeroDateTimeBehavior=convertToNull&autoReconnect=true&useSSL=true"/>`

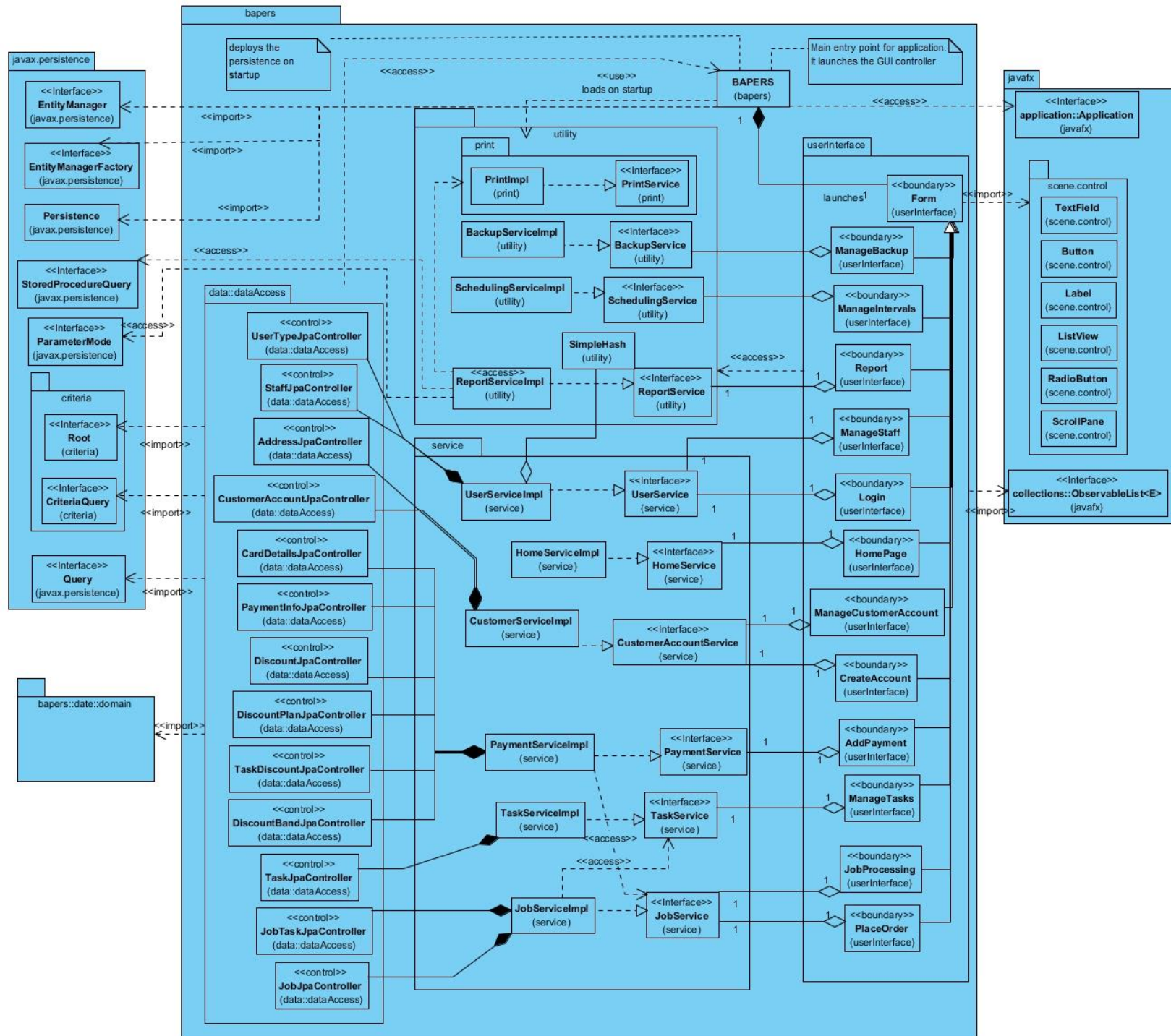
The `userInterface` will utilise `javaFX`; each form will have three separate files, a `bapers::fxml::form.fxml`, `bapers::styles::form.css` and `bapers::FormController.java`. The `fxml` file details objects within the form, and their respective location, and the controller class will describe the behaviour of the forms objects. However, for the sake of brevity, I have simplified this for the class diagram by bundling all three files into a single boundary class, and making the `userInterface` package, where they will be stored

6.2 Simplified Design Class Diagram

The following diagram presents the full class diagram; however, all classes have been stripped of their members. This is to concisely show all associations. Full details will be provided in section 6.3.

Please note that the `javax.persistence` package appears more than once. This was to allow for neater formatting of the diagram, all repeated instances should be treated as part of the same package. This also applies to any other repeated packages/classes.

6.2.1



This diagram excludes all entities from the `bapers::data::domain` package, as the associations for that package will be shown in the detailed diagram in section 1.3.

7 Use Case Indexing

In this section, a list is created to rank the most important use cases we believe are necessary for the functionality of BAPERS.

UC priority number	Use case
1	Place Order
3	Create Customer Account
2	Lookup Customer Account
4	Log In
5	Record Payment Details
6	Update Job Status
7	Add User
8	Configure Database Backup
9	Upgrade Customer Value
10	Update existing Task
11	Print Late Payment Reminder
12	Generate Report
13	Log Out
14	Create Second Letter
15	Remove Task
16	Recording Commencement Of Next Task
17	Alert Incomplete Job
18	Assign Job Number
19	Record Payment Details
20	Alert Shift Manager
21	Print Label
22	Add Card Details
23	Set User Access Privilege
24	Activate Customer Account
25	Record Completion Of Current Task
26	Log Out
27	Create Backup
28	Restore From Backup
29	Account Suspension
30	Configure Reports
31	Associate Job
32	Automatic Backup
33	Set Customer Value
34	Downgrade
35	Upgrade
36	Alert Of Late Payment
37	Acknowledge Late Payment Warning
38	Generate Payment Reminder Letter
39	Print Receipt
40	Print Reminder Query

41	Print Individual Reminder
42	Print All Reminders
43	Print Queue
44	Create First Letter
45	Manage Task
46	Make Account 'in default'
47	Set Customer Value
48	Downgrade
49	Create New Task
50	Time Configuration
51	Set Discount Plan
52	Revoke Suspension
53	Record Special Instructions
54	Summary Performance Report
55	Individual Performance Report
56	Individual Reports

Insert here – Risk Analysis (missing reports +

ADMIN

- **Activate Customer Account:** Reactivates customer account marked as 'in default'.
 - The importance of this use case is high because this enables the customer account to be reactivated and useable. The risk of not having this use case means that we would have idle customer accounts in our database, which takes money for space and those customers with that account would not be able to demand any more to-do jobs until this is fixed. This is also a potential risk to push our customers away.
- **Time Configuration:** Configures the periodicity of automatic functions.
 - This is important for all automatic timely events and functions that take place within the system. The reason being that this use case sets the time for those functions to occur. Doing this it will ensure that all databases and reports will be backed up automatically and the latest versions are available in case of data loss or server downs. Also, this is helpful because it is very possible that the users forget to backup the data manually thus, this removes that human error. If this was not to be implemented correctly this could lead to loss of access to specific data when need in case of services down such as database server downs.
- **Configure Reports:** Changes the frequency of reports.
 - This is directly linked to the Time Configuration use case as initially it is setup from that use case, but this use case allows you to change the automatic backup of the reports. This is an appropriate use case that can be used to automatically backup the reports to meet your requirements. Moreover, this is very useful because it is possible that the users forget to create the report backup and they would be able to keep track of progress and report backups.
- **Configure Database Backup:** Changes the frequency of automatic backups.
 - This is directly linked to the Time Configuration use case as initially it is setup from that use case, but this use case allows you to configure the automatic backup of the database. This is important as you need regular updates of the database in case of data loss or inaccessibility of the database. That way you don't lose all the information that you had in your database. Moreover, it is good to have automatic backups as it is very likely for the users to forget to create regular backups and they would be able to monitor and keep track of their database and can be used for analysis.
- **Create Backup:** Manually creates/overwrites system backup.
 - Being able to create a backup when necessary is important. The risks of not being able to do this include the fact that if your servers/databases were to crash or fail, our system would be down until fixed. However, with a backed-up database we can continue to work with the latest information from the backup. Having the system down will result in a major financial loss as for that period of time the system was not being used. This would also be more time efficient because as we continue to work of the backup we can proceed to fixing the main server. Also, this allows the backing up any important data as soon as it is entered in the system, rather than having to wait for the timely backups.

- **Add User:** Creates a user account for BAPERS.
 - The risk of not implementing this would include the fact that you would have no users for the system thus no coordination and no access to the software functions. To be able to use certain functions/the system you are required to have an account and without it your access will technically be blocked and unusable. This would result to you having to continue to process jobs, tasks etc. the entire process as you have done prior to this software, which is time consuming and much costlier.

- **Set User Access Privilege:** Sets up user access privileges.
 - This is important to set what each user can do in the system. What access they have, what functions they can use, what their responsibilities are etc. The risk of not having this can be drastic, for example if all users have access to changing a job status/demands and everyone is changing those requirements and statuses this can cause confusion between the employees and can effect the completion of the job. Which will result in unsatisfied customers and we would have to amend the job to satisfaction, which will result in more cost and time. Therefore, it is important to set user privileges that allows access according to your user account type.

- **Restore from Backup:** Restores BAPERS from an existing backup.
 - This follows from the Create Backup use case. If we encounter a problem with the system/ system data and it is no longer useable we need to be able to restore the system to a working backup so that it does not halt the jobs that need to be done, because consequently if we do not have this backup functionality this will result in money loss, data loss, server space loss and dissatisfied customers. Making this use case very important.

- **Place Order:** The actor (Managers or Receptionist) placing the order on behalf of the customers.
 - The risks involved in not having this use case implemented properly is that no job will be entered into the system thus the system would be useless as it won't be used, as a job is required to be processed and go through all the other necessary functions to complete a job. This would result in a waste of data-base space and money as we paid for it, but it is not being used. Moreover, customers would be dissatisfied with uncompleted jobs and a waste of their time, which would push customers away.
- **Alert Shift Manager:** To alert the shift manager there is a new job arrive and should be process.
 - This is an important use case to be implemented as it relays information to one of the heads of the managers with high privileges in the system, the shift manager. If this use case was not to be implemented properly, this would mean that the shift manager would not be notified on the new jobs that come in and he would not redirect them, manage them (oversee them) and would not be able to process them. Moreover, this can cause confusion of unknown jobs being entered into the system which the manager may think the system needs checking which takes time and money, so wouldn't be efficient. Also, this would leave a gap in his knowledge of processing jobs which can cause problems for him/her when asked questions or talking to customers.
- **Print Receipt:** Print a receipt for the customer about the job he/she order.
 - The risk of not having this use case implemented means that the customer would not have solid proof of the job order that they have given BILP. Moreover, it's important for the business to have proof of the jobs they have taken in case of any legal issues or customer complaints, they can refer to the receipts to counter their arguments. If this use case was not to be implemented correctly this could have a negative impact on the income and potentially time.
- **Record Special Instruction:** Record additional requirement from the customer.
 - This is a necessary additional functionality to enable customers to have their input on certain tasks and must be implemented correctly. If this was not implemented there is the risk of the tasks not being completed to the customer satisfaction and having missed out key elements that is required to complete the job. This can cause customers wanting a refund as expectations were not met and to reprocess the job (finishing the job by adding the missed-out details), which will take more time and can halt taking in more jobs. This can have a negative impact on the income and time efficiency thus making it important that this use case is implemented correctly.
- **Record Deadline:** Record the required job end time.
 - Recording the deadline is important for several reasons that involve both the business and the customers. The most important being so that we can deliver the completed job on time to the customer and ensuring that they are satisfied. This in turn will bring them back to our business as they were pleased with the service and quality of the job. If there was no deadline the business could take a time to com-

plete it and delivery to customers would be inconsistent which would make them dissatisfied. Furthermore, it doesn't show an indication of what priority level the job has. This would cost us our time, database space and money negatively.

- **Lookup Customer Account:** Search the customer account in order to identify the identity of the customer.
 - This is an essential use case that needs to be implemented correctly otherwise this can risk creating duplicated customer accounts and unmatched/linked jobs, which will create confusion in the team and take up more database space and time as more accounts are being made than necessary and searched. Moreover, then one job could be linked to multiple customer accounts which cannot happen. Being able to search up customers gives us the ability to search valuable information that is associated with that customer. Thus, it is important to ensure this use case is implemented correctly.
- **Create Customer Account:** creating a new instance of customer account due to the very first order from the new customer.
 - This is closely linked to the Lookup Customer Account use case and is very important. The risks of this not being implemented correctly include that jobs given to BILP will have no customer associated with it, meaning that the team would not know who these jobs belong to. This can cause major confusion and can potentially lead to not doing the job as it belongs to no one. Moreover, there would be no data/information about the customer so there would be no way to contact them or get paid or get other specific details. Therefore, it is important that this is conducted correctly and linked to one customer and their jobs only.
- **Print Label:** Print the label for materials required for job.
 - The risk of this not being implemented include that the job cannot be completed as the teams updating/doing the job will not know what the customer wanted/what materials are required, thus don't know what to do to the picture or what is required to complete the process. This will affect team coordination and time as constant communication will be required to answer queries they will have about the job and the materials required.
- **Assign Job Number:** Generate a number for the job which is unique compare to other jobs.
 - This is important to help coordinate the job pool. This will also allow the team to access jobs easier by just calling the job number rather than the job name or customer associated with it. This is more time efficient. Alternatively, this risks job confusion with other jobs and can make it less time efficient if this is not implemented.

CUST

- **Set Customer value:** office manager chooses to set a customer to a valued customer
 - This functionality is important to be implemented because there are other functions that can be carried out after this has been done. The risk of this not being implemented correctly effects the types of discount that the customer can get, as only when a customer is valued can they be given a discount plan and other functions would not be used thus taking up more space and code that is not used.
- **Downgrade:** office manager can choose to downgrade a customer's valued status
 - This is directly linked to the Set Customer value use case. Its important that this is implemented correctly so that the manager can change a customer account from valued to non-valued. This is important because if this was not available customers that became valued customers would stay valued customers even if they don't reach the requirements for the valued customer status. Thus, they could still have the discounts which would result in the company/business making a loss which is a major risk therefore, this needs to be implemented correctly.
- **Upgrade:** office manager can choose to upgrade a customer's valued status
 - This is directly linked to the Downgrade and Set Customer value use cases. This is important and must be implemented because this is the use case that triggers the other use cases. Not implementing this will mean that we are unable to update the valued status of the customer which will cause risks and problems as explained in the other two related use cases.
- **Set discount plan:** when a customer is upgraded, the office manager has the option to set up a discount plan according to how many Jobs/tasks a customer has placed.
 - When a customer account is upgraded to a valued customer account they then get a discount plan. This use case sets the discount plan to that account therefore is a necessity. If this use case was not to be implemented correctly this would lead to the manager not being able to give the valued customer a discount plan which can then lead to unvalued customers, consequently leading to less customers and dissatisfaction. Moreover, it would make the previous three use cases useless (Set Customer value, Downgrade, Upgrade).
- **Flexible:** this use case defines the percentage of the discount that depends on the values of the jobs by the same customer accumulated within a calendar month.
- **Variable:** this use case defines the percentage of the discount that is set for each task and may vary between the tasks. The value of this discount is calculated and deducted from the value of the job at the time of accepting the job.

- **Fixed:** this use case defines the same percentage of discount given to the valued customer for each job. The value of this discount is calculated and deducted from the value of the job at the time of accepting the job.

These are the various discount plans use cases that are available to the manager than he can set for the valued customers. These use cases are directly linked to the Set discount plan use case as that use case then refers to one of these for implementation. If these are not implemented correctly it would make the Set discount plan use case useless as there would be no discount plan that can be set for the valued customer. Moreover, this would also be a waste of memory/space as it is a use case that would not be able to complete its functionality. Therefore, to avoid this risk it is important that these use ca

TODO: FINISH^^

8 Use Case Specifications

This section includes the 10 key use case specifications that were made for BAPERS

Record Payment

Use Case ID:	Use Case: Record Payment
Brief Description: Once a customer makes a payment with BAPERS, a record of the payment is then stored in the local memory of the machine running the system and then the record is stored in the database.	
Primary Actors:	Office Manager, Receptionist
Secondary Actors:	Database Server
Preconditions: <ol style="list-style-type: none"> 1) The system is operational 2) Either the Office Manager, or the Receptionist is logged into the system and had chosen the GUI option to record a payment from a client. 	
Main Flow: <ol style="list-style-type: none"> 1) The use case starts when the Office Manager, or receptionist selects Record Payment functionality. 2) The GUI prompts the user to then enter the payment details including: amount paid, whether the payment is card or cash. 3) The GUI then sends the input information to the system, and the system creates a payment object. 4) The system then finds the outstanding (unpaid) jobs for the customer, and adds them to the payment object. 5) The system contacts the Database Server, and sends the payment object to be stored on disk. 6) The system deletes the local payment object. <p>The system alerts the user that payment has been added.</p>	
Postconditions: The database contains a new payment record	
Alternative Flows: CardPayment NoConnectionToServer UnSuspendAccount	

Use Case ID:	Use Case: Record Payment : CardPayment
Brief Description: The customer has made a payment with a card, so the card details will be added to the system.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) The system is operational and Either the Office Manager, or the Receptionist is logged into the system and had chosen the GUI option to record a payment from a client. 	
Alternate Flow: <ol style="list-style-type: none"> 1) The flow may start at step 2 of the main flow. 2) The GUI prompts the user to enter card details. 3) While (more cards) <ol style="list-style-type: none"> a. The user enters card details. 4) The GUI sends the information to the system, and the system then creates a card object, and a payment object. Giving the payment object a reference to the card object. 	

The flow should resume at step 2 of the main flow.
Postconditions: Local storage contains a Card Details Object, which is referenced by a Payment object.

Use Case ID:	Use Case: Record Payment : NoConnectionToServer
Brief Description: A connection cannot be established with the database server, so the operation is aborted.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: 1) The system is operational and either the Office Manager, or the Receptionist is logged into the system and had chosen the GUI option to record a payment from a client.	
Alternate Flow: 1) The flow may start at step 5 of the main flow. 2) The system informs the user that no communication channel can be established to the database.	
Postconditions: The contents of the database are unchanged	

Generate Letter

Use Case ID:	Use Case: Generate 2 nd Letter
Brief Description: BAPERS automatically generate the second letter to the Office Manager which going to be send to the customers.	
Primary Actors:	Time
Secondary Actors:	Office Manager
Preconditions: 1) First letter has been generated and sent to customer. 2) One month after the first letter is sent and payment yet to be receive. 3) Office Manager logs into his/her account.	
Main Flow: 1) The Use case starts when it is a month after the first letter sent out. 2) BAPERS suspend the corresponding customer's account. 3) BAPERS awaits User with user type Office Manager to Log-in. 4) BAPERS alert the Office Manager and second letter is prepare for him/her to print. 5) BAPERS connects to a printer with existing communication channel. 6) BAPERS inform the Office Manager the print has been completed.	
Postconditions: The letter successfully printed out by the Office Manager with no error occur.	
Alternative Flows: NoPrinterConnection	

Use Case ID:	Use Case: Generate 2 nd Letter:NoPrinterConnection
Brief Description: Printing error due to lack of communication with the hardware printer.	
Primary Actors:	N/A
Secondary Actors:	N/A

Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) Office Manager logs into BAPERS. 3) Office Manager receive the alert of the second letter. 4) Office Manager chosen to print the second letter.
Alternative Flows: <ol style="list-style-type: none"> 1) The flow may start at step 5 of the main flow. 2) BAPERS informs the Office Manager that no communication between the printer and the computer can be located.
Postconditions: The letter is not print on demand by the Office Manager.

Create User

Use Case ID:	Use Case: Create User
Brief Description: Office Manager creates an user account for BAPERS.	
Primary Actors:	Office Manager
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) A user with user type Office Manager has logged into BAPERS. 	
Main Flow: <ol style="list-style-type: none"> 1) The use case starts when the Office Manager selects “Create User” functionality. 2) Office Manager input details for the user account. 3) Office Manager setting up privileges for the account. 4) BAPERS connect with the database with existing communication channel and send details of the new creation account to it. 5) BAPERS informs Office Manager that a user account has been successfully created. 	
Postconditions: The database contains the new account details.	
Alternative Flows: NoCommunicationChannel	

Use Case ID:	Use Case: Create User:NoCommunicationChannel
Brief Description: Creation error due to lack of communication channel between BAPERS and the database.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) User with user type Office Manager logged into BAPERS. 3) Office Manager chosen to create user. 	
Alternative Flows: <ol style="list-style-type: none"> 1) The flow may start at step 4 of the main flow. 2) BAPERS informs the Office Manager that no communication channel can be located. 	
Postconditions: User account details in the database remain unchanged.	

Automatic Backup

Use Case ID:	Use Case: Automatic backup
Brief Description: BAPERS backs up the database server each time period specified.	
Primary Actors:	Time
Secondary Actors:	Database server
Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) Automatic backup period is specified. 	
Main Flow: <ol style="list-style-type: none"> 1) Use case starts when specified time occurs. 2) BAPERS backs up the Database server. 	
Postconditions: The database server is backed up/replaces the old backup.	
Alternative Flows: NoCommunicationChannel	

Use Case ID:	Use Case: Automatic backup:NoCommunicationChannel
Brief Description: Automatic backup unavailable due to lack of communication channel.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) Automatic backup period is specified. 	
Alternative Flows: <ol style="list-style-type: none"> 1) The flow may start at step 2 of the main flow. 2) BAPERS informs the User that no communication channel can be located 	
Postconditions: The database server is not backed up.	

Update Existing Task

Use Case ID:	Use Case: Update existing task
Brief Description: Edit existing tasks from the task list.	
Primary Actors:	Office manager
Secondary Actors:	Database server
Preconditions: <ol style="list-style-type: none"> 1) BAPERS is operational. 2) Primary actor is logged in to BAPERS. 	
Main Flow: <ol style="list-style-type: none"> 1) Use case starts when the User selects "Update existing task" functionality. 2) Tasks are retrieved from the database server 3) User selects a task. 4) User updates the details of the task 5) BAPERS updates the task. 6) BAPERS informs user that the task has been updated. 	
Postconditions: The database server contains updated task; all previously stored tasks are unchanged.	
Alternative Flows: NoCommunicationChannel	

Use Case ID:	Use Case: Update existing task:NoCommunicationChannel
Brief Description: Task updating unavailable due to lack of communication channel.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: 1) BAPERS is operational. 2) Primary actor is logged in to BAPERS	
Alternative Flows: 1) The flow may start at step 2 of the main flow. 2) BAPERS informs the User that no communication channel can be located	
Postconditions: The content of tasks in database server remains unchanged.	

Upgrade Customer Account

Use Case ID:	Use Case: UpgradeCustomerAccount
Brief Description: Upgrade important customers to the “Valued” status, enabling them to benefit from different discounts.	
Primary Actors:	Office Manager
Secondary Actors:	Database
Preconditions: 1) The system is functional and the Office Manager logs in to the system	
Main Flow: 1) The Use Case starts when the Office Manager logs into the system 2) The Office Manager accesses the customer list 3) The Office Manager selects the customer 4) The Office Manager clicks the GUI button for upgrading the customer to the “Valued” status 5) When the button is clicked, the system fetches the value from the database 6) The system creates an object into the system where it stores the value 7) The system changes the old value with the new value 8) The data is sent back by the system to the database 9) The database overwrites the old status with the new status.	
Postconditions: The database is updated with the new customer status and the customer is eligible for discounts	
Alternative Flows: InvalidAccountUpgraded	

Use Case ID:	Use Case: UpgradeCustomer Account:InvalidAccountUpgraded
Brief Description: The system informs the Office Manager that the account he is trying to upgrade is already upgraded	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: 1) The Office Manager has tried upgrading an account that is already upgraded	
Alternative Flows: 1) Alternative flow begins at step 4	

<ol style="list-style-type: none"> 2) The Office Manager clicks the GUI button for updating the user status 3) The system fetches the data from the database 4) The system compares detects that the change already took place 5) The system displays the error message 6) The system does not send any information back to the database and deletes the object created
Postconditions: The database is not modified

Print Late Reminder

Use Case ID:	Use Case: PrintLateReminder
Brief Description: Print the late payment reminder for the customer who did not pay in time for the service.	
Primary Actors:	Office Manager
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) The system is functional; the Office Manager logs in to the system and the customer exceeded the payment deadline 	
Main Flow: <ol style="list-style-type: none"> 1) The screen displays the notification that a customer has exceeded the payment deadline 2) The Office Manager Clicks the GUI button for printing the letter 3) The letter is printed 	
Postconditions: The Database is updated with the information of a user having a letter printed for	
Alternative Flows: PrinterError	

Use Case ID:	Use Case: PrintLateReminder: PrintLateReminder
Brief Description: Office manager is unable to print the letters required as a print error occurs	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1) The system is functional; the Office Manager logs in to the system and the customer exceeded the payment deadline 	
Alternative Flows: <ol style="list-style-type: none"> 1) Flow may start at step 2 of the main flow 2) There is a printer error when trying to print the letter 	
Postconditions: The late reminder message is not printed	

Login

Use Case ID:	Use Case: Login
Brief Description: The procedure where the users/staff can login to the BAPERS system so that they can use the system and its functions.	
Primary Actors:	Receptionist, Shift Manager, Office Manager, Technician
Secondary Actors:	Database server

Preconditions: <ol style="list-style-type: none"> 1. System is operational, the system has a secure connection to the database server. 2. The primary actors must be authorised.
Main Flow: <ol style="list-style-type: none"> 1. The use case starts when the user selects the “Login” button. 2. The system opens another page where the user is told to enter their login details in the space provided. 3. The user then presses the login button. 4. The system then searches the database for corresponding login details. <ol style="list-style-type: none"> 4.1 Corresponding login details are not found. While not found: <ol style="list-style-type: none"> 4.1.1 System tells user: invalid username or password. 4.1.2 System tells user to check and re-enter their username and password. 4.2 The user then clicks the login button again. 5. User then login 6. Use case completed.
Postconditions: The user login and the software and its functions are accessible to the user.
Alternative Flows: LoginFailed

Use Case ID:	Use Case: Login: LoginFailed
Brief Description: The system informs the user that login failed due to an incorrect username or password or both.	
Primary Actors:	N/A
Secondary Actors:	N/A
Preconditions: <ol style="list-style-type: none"> 1. The user entered an incorrect username and/or password, which was not found in the database. 	
Alternative Flows: <ol style="list-style-type: none"> 1. The alternative flow begins at step 4 of the main step. 2. The system informs the user that invalid details have been provided. 	
Postconditions: The user is denied access to the system	

Generate Reports

Use Case ID:	Use Case: Generate Report
Brief Description: Various reports are generated automatically by BAPERS	
Primary Actors:	Time
Secondary Actors:	Database server
Preconditions: <ol style="list-style-type: none"> 1. BAPERS is operational 2. There is an established connection to the database server. 	
Main Flow: <ol style="list-style-type: none"> 1. The use case starts when the time triggers the functionality of generating reports. 2. Individual report for the jobs brought in by a particular customer for an arbitrary period is taken. 3. Individual performance report on work undertaken by a member of BIPL staff is generated 4. Summary Performance report for work undertaken by BIPL during day and night shifts is generated. 	
Postconditions:	

The generated reports get stored in the database
Alternative Flows: None

Update Job Status

Use Case ID:	Use Case: Update Job Status
Brief Description: primary actor updates the status of an ongoing job	
Primary Actors: Office Manager, Shift Manager, Technician	
Secondary Actors: N/A	
Preconditions: <ol style="list-style-type: none"> 1. The system is operational. 2. The primary actor is already logged in. 	
Main Flow: <ol style="list-style-type: none"> 1. The use case starts when the Manager or Technician chooses to update the status of the job 2. They can record the completion of the current task which is shown as an include 3. They can also update the system with the commencement of the next task 4. Once updated, the status is recorded and sent back to the database. 	
Postconditions: Completion of tasks are recorded	
Alternative Flows:	

9. Systems Evolution

We expect to build a similar product based on BAPERS in the near future.

May add it to Cloud storage

Adding new tasks, system may be available on other platforms

10. Appendices

Initial statement of requirements set by client

After an interview with Mr Lancaster, the knowledge about the system is summarized as follows:

Insert our notes about what he said in the interview

11. Hardware and Software configuration

Hardware minimal requirements

Desktop PC with dual core Intel processor, at 3GHz, or quad core AMD processor at 3.5GHz,
2GB ddr3 RAM,
7200 RPM HDD with 2GB of free space,
Video card capable of 1024x768 output,
1024x768 monitor resolution.

Operating Systems:

All operating systems must support 64-bit operations.

Windows:

Windows 10 (version 8u51 and above)

Windows 8.x (Desktop version)

Mac:

Mac OS X 10.8.3+, 10.9+

Linux:

Linux kernel release 4.14.0 or above.

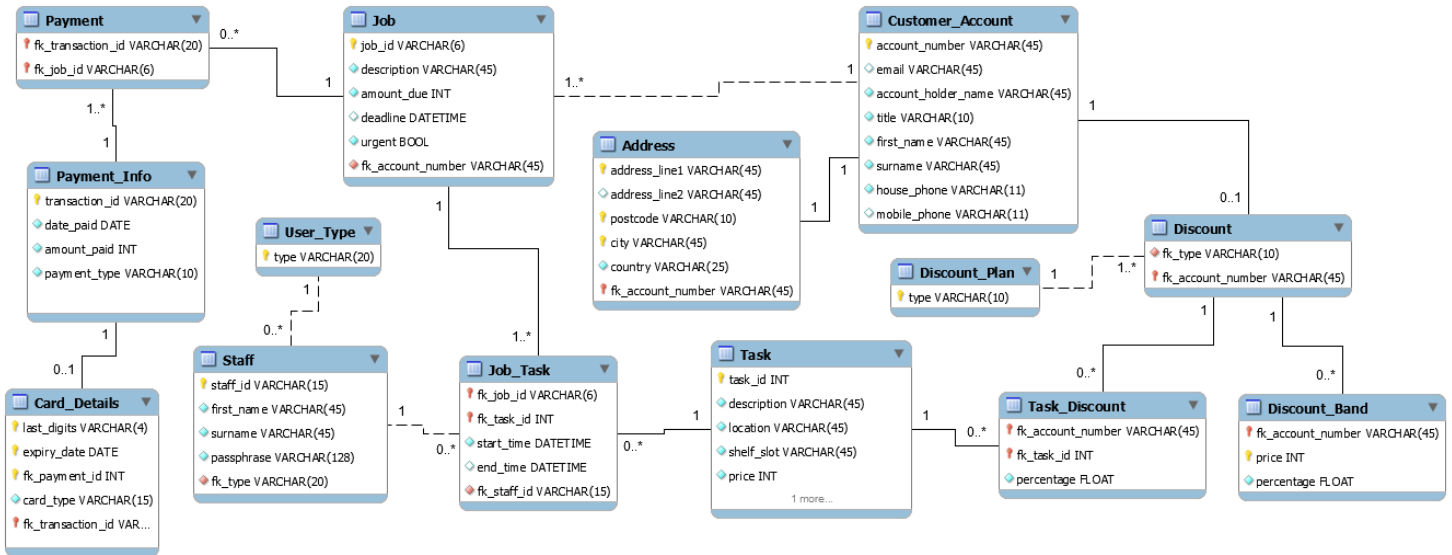
Support for open jdk 8 jre, and a MySQL implementation (recommended is MariaDB).

Additional Software

BAPERS requires the java 8/9 runtime to be installed.

MySQL Community Server 5.7.0 or above.

12. Database



12.1 SQL schema

Below is the schema, including all stored procedures with dummy data necessary for executing reports.

```
-- MySQL Workbench Forward Engineering
```

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
```

```
-- Schema BAPERS
```

```
DROP SCHEMA IF EXISTS `BAPERS` ;
```

```
-- Schema BAPERS
```

```
CREATE SCHEMA IF NOT EXISTS `BAPERS` DEFAULT CHARACTER SET utf8 ;
USE `BAPERS` ;
```

```
-- Table `BAPERS`.`User_Type`
```

```
DROP TABLE IF EXISTS `BAPERS`.`User_Type` ;
```

```
CREATE TABLE IF NOT EXISTS `BAPERS`.`User_Type` (
  `type` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`type`))
ENGINE = InnoDB;
```

```
-- Table `BAPERS`.`Staff`
```

```
DROP TABLE IF EXISTS `BAPERS`.`Staff` ;
```

```
CREATE TABLE IF NOT EXISTS `BAPERS`.`Staff` (
  `staff_id` VARCHAR(15) NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
```

```

`surname` VARCHAR(45) NOT NULL,
`passphrase` VARCHAR(128) NOT NULL,
`fk_type` VARCHAR(20) NOT NULL,
PRIMARY KEY (`staff_id`),
INDEX `fk_User_User_Type1_idx` (`fk_type` ASC),
UNIQUE INDEX `username_UNIQUE` (`staff_id` ASC),
CONSTRAINT `fk_User_User_Type1`
  FOREIGN KEY (`fk_type`)
    REFERENCES `BAPERS`.`User_Type` (`type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `BAPERS`.`Customer_Account`
-----
DROP TABLE IF EXISTS `BAPERS`.`Customer_Account` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Customer_Account` (
  `account_number` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NULL,
  `account_holder_name` VARCHAR(45) NOT NULL,
  `title` VARCHAR(10) NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
  `surname` VARCHAR(45) NOT NULL,
  `house_phone` VARCHAR(11) NOT NULL,
  `mobile_phone` VARCHAR(11) NULL,
  PRIMARY KEY (`account_number`))
ENGINE = InnoDB;

-----
-- Table `BAPERS`.`Job`
-----
DROP TABLE IF EXISTS `BAPERS`.`Job` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Job` (
  `job_id` VARCHAR(6) NOT NULL,
  `description` VARCHAR(45) NOT NULL,
  `amount_due` INT NOT NULL,
  `deadline` DATETIME NULL,
  `urgent` TINYINT(1) NOT NULL,
  `fk_account_number` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`job_id`),
  INDEX `fk_Job_Customer_Account1_idx` (`fk_account_number` ASC),
  CONSTRAINT `fk_Job_Customer_Account1`
    FOREIGN KEY (`fk_account_number`)
      REFERENCES `BAPERS`.`Customer_Account` (`account_number`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `BAPERS`.`Payment_Info`
-----
DROP TABLE IF EXISTS `BAPERS`.`Payment_Info` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Payment_Info` (
  `transaction_id` VARCHAR(20) NOT NULL,
  `date_paid` DATE NOT NULL,

```

```

`amount_paid` INT NOT NULL,
`payment_type` VARCHAR(10) NOT NULL,
PRIMARY KEY (`transaction_id`))
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Card_Details`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Card_Details` ;

```

```

CREATE TABLE IF NOT EXISTS `BAPERS`.`Card_Details` (
  `last_digits` VARCHAR(4) NOT NULL,
  `expiry_date` DATE NOT NULL,
  `fk_payment_id` INT NOT NULL,
  `card_type` VARCHAR(15) NOT NULL,
  `fk_transaction_id` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`last_digits`, `expiry_date`, `fk_payment_id`, `fk_transaction_id`),
  INDEX `fk_Card_Details_Payment_Info1_idx` (`fk_transaction_id` ASC),
  CONSTRAINT `fk_Card_Details_Payment_Info1`
    FOREIGN KEY (`fk_transaction_id`)
      REFERENCES `BAPERS`.`Payment_Info` (`transaction_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Address`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Address` ;

```

```

CREATE TABLE IF NOT EXISTS `BAPERS`.`Address` (
  `address_line1` VARCHAR(45) NOT NULL,
  `address_line2` VARCHAR(45) NULL,
  `postcode` VARCHAR(10) NOT NULL,
  `city` VARCHAR(45) NOT NULL,
  `country` VARCHAR(25) NOT NULL,
  `fk_account_number` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`address_line1`, `city`, `postcode`, `fk_account_number`),
  INDEX `fk_Address_Customer_Account1_idx` (`fk_account_number` ASC),
  CONSTRAINT `fk_Address_Customer_Account1`
    FOREIGN KEY (`fk_account_number`)
      REFERENCES `BAPERS`.`Customer_Account` (`account_number`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Task`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Task` ;

```

```

CREATE TABLE IF NOT EXISTS `BAPERS`.`Task` (
  `task_id` INT NOT NULL,
  `description` VARCHAR(45) NOT NULL,
  `location` VARCHAR(45) NOT NULL,
  `shelf_slot` VARCHAR(45) NOT NULL,
  `price` INT NOT NULL,
  `duration` TIME NOT NULL,
  PRIMARY KEY (`task_id`))

```

```
ENGINE = InnoDB;
```

```
-- Table `BAPERS`.`Discount_Plan`
```

```
DROP TABLE IF EXISTS `BAPERS`.`Discount_Plan` ;
```

```
CREATE TABLE IF NOT EXISTS `BAPERS`.`Discount_Plan` (  
  `type` VARCHAR(10) NOT NULL,  
  PRIMARY KEY (`type`))  
ENGINE = InnoDB;
```

```
-- Table `BAPERS`.`Job_Task`
```

```
DROP TABLE IF EXISTS `BAPERS`.`Job_Task` ;
```

```
CREATE TABLE IF NOT EXISTS `BAPERS`.`Job_Task` (  
  `fk_job_id` VARCHAR(6) NOT NULL,  
  `fk_task_id` INT NOT NULL,  
  `start_time` DATETIME NOT NULL,  
  `end_time` DATETIME NULL,  
  `fk_staff_id` VARCHAR(15) NOT NULL,  
  PRIMARY KEY (`fk_job_id`, `fk_task_id`),  
  INDEX `fk_Job_has_Task_Task1_idx` (`fk_task_id` ASC),  
  INDEX `fk_Tasks_Staff1_idx` (`fk_staff_id` ASC),  
  INDEX `fk_Job_Task_Job1_idx` (`fk_job_id` ASC),  
  CONSTRAINT `fk_Job_has_Task_Task1`  
    FOREIGN KEY (`fk_task_id`)  
    REFERENCES `BAPERS`.`Task` (`task_id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION,  
  CONSTRAINT `fk_Tasks_Staff1`  
    FOREIGN KEY (`fk_staff_id`)  
    REFERENCES `BAPERS`.`Staff` (`staff_id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION,  
  CONSTRAINT `fk_Job_Task_Job1`  
    FOREIGN KEY (`fk_job_id`)  
    REFERENCES `BAPERS`.`Job` (`job_id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

```
-- Table `BAPERS`.`Discount`
```

```
DROP TABLE IF EXISTS `BAPERS`.`Discount` ;
```

```
CREATE TABLE IF NOT EXISTS `BAPERS`.`Discount` (  
  `fk_type` VARCHAR(10) NOT NULL,  
  `fk_account_number` VARCHAR(45) NOT NULL,  
  INDEX `fk_Discount_Discount_Plan1_idx` (`fk_type` ASC),  
  PRIMARY KEY (`fk_account_number`),  
  CONSTRAINT `fk_Discount_Discount_Plan1`  
    FOREIGN KEY (`fk_type`)  
    REFERENCES `BAPERS`.`Discount_Plan` (`type`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION,
```

```

CONSTRAINT `fk_Discount_Customer_Account1`
  FOREIGN KEY (`fk_account_number`)
  REFERENCES `BAPERS`.`Customer_Account` (`account_number`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Task_Discount`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Task_Discount` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Task_Discount` (
  `fk_account_number` VARCHAR(45) NOT NULL,
  `fk_task_id` INT NOT NULL,
  `percentage` FLOAT NOT NULL,
  PRIMARY KEY (`fk_account_number`, `fk_task_id`),
  INDEX `fk_Customer_Account_has_Task_Task1_idx` (`fk_task_id` ASC),
  INDEX `fk_Discounts_Discount1_idx` (`fk_account_number` ASC),
  CONSTRAINT `fk_Customer_Account_has_Task_Task1`
    FOREIGN KEY (`fk_task_id`)
    REFERENCES `BAPERS`.`Task` (`task_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Discounts_Discount1`
    FOREIGN KEY (`fk_account_number`)
    REFERENCES `BAPERS`.`Discount` (`fk_account_number`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Discount_Band`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Discount_Band` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Discount_Band` (
  `fk_account_number` VARCHAR(45) NOT NULL,
  `price` INT NOT NULL,
  `percentage` FLOAT NOT NULL,
  PRIMARY KEY (`fk_account_number`, `price`),
  CONSTRAINT `fk_table1_Discount1`
    FOREIGN KEY (`fk_account_number`)
    REFERENCES `BAPERS`.`Discount` (`fk_account_number`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `BAPERS`.`Payment`
-----

```

```

DROP TABLE IF EXISTS `BAPERS`.`Payment` ;

CREATE TABLE IF NOT EXISTS `BAPERS`.`Payment` (
  `fk_transaction_id` VARCHAR(20) NOT NULL,
  `fk_job_id` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`fk_transaction_id`, `fk_job_id`),
  INDEX `fk_Payment_Info_has_Job_Job1_idx` (`fk_job_id` ASC),
  INDEX `fk_Payment_Info_has_Job_Payment_Info1_idx` (`fk_transaction_id` ASC),

```

```

CONSTRAINT `fk_Payment_Info_has_Job_Payment_Inf1`
  FOREIGN KEY (`fk_transaction_id`)
  REFERENCES `BAPERS`.`Payment_Info` (`transaction_id`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT `fk_Payment_Info_has_Job_Job1`
  FOREIGN KEY (`fk_job_id`)
  REFERENCES `BAPERS`.`Job` (`job_id`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;

USE `BAPERS` ;

-- -----
-- procedure night_shift
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`night_shift`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE night_shift
(IN date_start date,IN date_end date)
BEGIN
call shift(date_start,date_end,'22:00:00','05:00:00');
END$$

DELIMITER ;

-- -----
-- procedure shift
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`shift`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE shift
(in date_start char(20),in date_end char(20),in shift_start char(20), in shift_end
char(20))
BEGIN
drop temporary table if exists tmp;
BEGIN
create temporary table tmp
Select date(job_task.start_time)as 'Date',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Copy Room' and job_task.start_time>=concat(date,' ',shift_start)
and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT concat(DATE_ADD(date,
INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and if(shift_end
='05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)= DATE_ADD(date,
INTERVAL 1 DAY),date(job_task.start_time) = date),date(job_task.start_time) = date)
)
as 'Copy_Room',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )

```



```

from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Development Area' and job_task.start_time>=concat(date,
',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end ='05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Development',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Finishing Room' and job_task.start_time>=concat(date,
',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end ='05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Finishing',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Packing Departments' and job_task.start_time>=concat(date,
',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end ='05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Packing'
from Job_Task
inner join task
on Job_task.fk_task_id = task.task_id
where date(job_task.start_time) between date_start and date_end
group by date(job_task.start_time);
END;
select * from tmp;
select sec_to_time(sum(time_to_sec(Copy_room))) as Copy_room,
sec_to_time(sum(time_to_sec(Development))) as Development,
sec_to_time(sum(time_to_sec(Finishing))) as Finishing,
sec_to_time(sum(time_to_sec(Packing))) as Packing
from tmp;
drop temporary table if exists tmp;
END$$

DELIMITER ;

-- -----
-- procedure day_shift1
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`day_shift1`;

DELIMITER $$

```

```

USE `BAPERS`$$
CREATE PROCEDURE day_shift1
(IN date_start date,IN date_end date)
BEGIN
call shift(date_start,date_end,'05:00:00','14:30:00');
END$$

DELIMITER ;

-- -----
-- procedure day_shift2
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`day_shift2`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE day_shift2
(IN date_start date,IN date_end date)
BEGIN
call shift(date_start,date_end,'14:30:00','22:00:00');
END$$

DELIMITER ;

-- -----
-- procedure summary_shift
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`summary_shift`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE summary_shift
(in date_start char(20),in date_end char(20),in shift_start char(20), in shift_end
char(20),in shift char(20))
BEGIN
drop temporary table if exists tmp;
drop table if exists Sum_shift;
BEGIN
create temporary table tmp
Select date(job_task.start_time)as 'Date',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Copy Room' and job_task.start_time>=concat(date,' ',shift_start)
and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT concat(DATE_ADD(date,
INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and if(shift_end
='05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)= DATE_ADD(date,
INTERVAL 1 DAY),date(job_task.start_time) = date),date(job_task.start_time) = date)
)
as 'Copy_Room',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Development Area' and job_task.start_time>=concat(date,'

```

```

',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end = '05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Development',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Finishing Room' and job_task.start_time>=concat(date,'
',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end = '05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Finishing',
(select SEC_TO_TIME( SUM( TIME_TO_SEC( TIMEDIFF(Job_Task.end_time,Job_task.start_time) )
) )
from Job_Task
inner join Task
on Job_Task.fk_task_id = task.task_id
where task.location = 'Packing Departments' and job_task.start_time>=concat(date,'
',shift_start) and job_task.start_time < (IF(shift_end = '05:00:00',(SELECT
concat(DATE_ADD(date, INTERVAL 1 DAY),' ',shift_end)),concat(date,' ',shift_end))) and
if(shift_end = '05:00:00',if(date(job_task.start_time)>date,date(job_task.start_time)=
DATE_ADD(date, INTERVAL 1 DAY),date(job_task.start_time) =
date),date(job_task.start_time) = date)
)
as 'Packing'
from Job_Task
inner join task
on Job_task.fk_task_id = task.task_id
where date(job_task.start_time) between date_start and date_end
group by date(job_task.start_time);
END;

create table Sum_shift as(
select shift as Title,
sec_to_time(sum(time_to_sec(Copy_room))) as Copy_room,
sec_to_time(sum(time_to_sec(Development))) as Development,
sec_to_time(sum(time_to_sec(Finishing))) as Finishing,
sec_to_time(sum(time_to_sec(Packing))) as Packing
from tmp);
drop temporary table if exists tmp;
END$$

DELIMITER ;

-- -----
-- procedure sum_night_shift
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`sum_night_shift`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE sum_night_shift
(IN date_start date,IN date_end date)

```

```

BEGIN
call Summary_Shift(date_start,date_end,'22:00:00','05:00:00','Night Shift');
END$$

DELIMITER ;

-- -----
-- procedure sum_day_shift2
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`sum_day_shift2`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE sum_day_shift2
(IN date_start date,IN date_end date)
BEGIN
call Summary_shift(date_start,date_end,'14:30:00','22:00:00','Day Shift2');
END$$

DELIMITER ;

-- -----
-- procedure sum_day_shift1
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`sum_day_shift1`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE sum_day_shift1
(IN date_start date,IN date_end date)
BEGIN
call Summary_Shift(date_start,date_end,'05:00:00','14:30:00','Day Shift1');
END$$

DELIMITER ;

-- -----
-- procedure summary
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`summary`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE summary
(IN date_start date,IN date_end date)
BEGIN
DROP temporary table IF EXISTS Sum_table;
create temporary table Sum_table(Title char(20),Copy_Room time,Development time,Finishing
time,Packing time);
call Sum_Day_shift1(date_start,date_end);
INSERT INTO Sum_table SELECT * FROM Sum_shift order by Sum_shift.Title;
DROP table IF EXISTS Sum_shift;
call Sum_Day_shift2(date_start,date_end);
INSERT INTO Sum_table SELECT * FROM Sum_shift order by Sum_shift.Title;
DROP table IF EXISTS Sum_shift;
call Sum_night_shift(date_start,date_end);

```

```

INSERT INTO Sum_table SELECT * FROM Sum_shift order by Sum_shift.Title;
select * from Sum_table;
select
sec_to_time(sum(time_to_sec(Copy_room))) as Copy_room,
sec_to_time(sum(time_to_sec(Development))) as Development,
sec_to_time(sum(time_to_sec(Finishing))) as Finishing,
sec_to_time(sum(time_to_sec(Packing))) as Packing
from Sum_table;
DROP table IF EXISTS Sum_shift;
DROP temporary table IF EXISTS Sum_table;
END$$

DELIMITER ;

-- -----
-- procedure ipr
-- -----

USE `BAPERS`;
DROP procedure IF EXISTS `BAPERS`.`ipr`;

DELIMITER $$
USE `BAPERS`$$
CREATE PROCEDURE ipr
(IN ID char(20))
BEGIN
DROP TEMPORARY TABLE IF exists IPR1;
Create temporary table IPR1 as
select concat(first_name,' ',surname) as Name,
Job.job_id as 'Code',
Job_Task.fk_task_id as 'Task IDs',
Task.location as 'Department',
DATE(Job_Task.start_time) as 'Date',
TIME(Job_Task.start_time) as 'Start Time',
TIMEDIFF(Job_Task.end_time,Job_task.start_time) as 'Time Taken'
from staff
inner join job_task
on staff.staff_id = job_task.fk_staff_id
inner join job
on job_task.fk_job_id = job.job_id
inner join task
on job_task.fk_task_id = task.task_id
where if(ID = 'all',staff.fk_type = 'technician',staff_id = ID)
order by Name,SUBSTRING_INDEX('Start time', " ", -1), SUBSTRING_INDEX('Start time', " ",
1);
select * from IPR1;
select Name,sec_to_time(sum(time_to_sec(`Time Taken`))) as 'Individual Effort' from IPR1
group by Name;
select sec_to_time(sum(time_to_sec(`Time Taken`))) as 'Total Effort' from IPR1;
DROP TEMPORARY TABLE IF exists IPR1;
END$$

DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- -----
-- Data for table `BAPERS`.`User_Type`
-- -----

START TRANSACTION;

```

```

USE `BAPERS`;
INSERT INTO `BAPERS`.`User_Type` (`type`) VALUES ('office manager');
INSERT INTO `BAPERS`.`User_Type` (`type`) VALUES ('shift manager');
INSERT INTO `BAPERS`.`User_Type` (`type`) VALUES ('receptionist');
INSERT INTO `BAPERS`.`User_Type` (`type`) VALUES ('technician');

COMMIT;

-----
-- Data for table `BAPERS`.`Staff`
-----

START TRANSACTION;
USE `BAPERS`;
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0000', 'chris', 'stokes',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'office manager');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0001', 'john', 'nash',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'technician');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0002', 'lee', 'hong',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'technician');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0003', 'julie', 'abbot',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'technician');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0004', 'marina', 'scott',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'technician');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0005', 'stewart', 'pask',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'technician');
INSERT INTO `BAPERS`.`Staff` (`staff_id`, `first_name`, `surname`, `passphrase`,
`fk_type`) VALUES ('0006', 'rich', 'evans',
'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df
5f1326af5a2ea6d103fd07c95385ffab0cacbc86', 'office manager');

COMMIT;

-----
-- Data for table `BAPERS`.`Customer_Account`
-----

START TRANSACTION;
USE `BAPERS`;
INSERT INTO `BAPERS`.`Customer_Account` (`account_number`, `email`,
`account_holder_name`, `title`, `first_name`, `surname`, `house_phone`, `mobile_phone`)
VALUES ('acc0001', NULL, 'City University', 'prof', 'david', 'rhind', '02070408000',
NULL);
INSERT INTO `BAPERS`.`Customer_Account` (`account_number`, `email`,
`account_holder_name`, `title`, `first_name`, `surname`, `house_phone`, `mobile_phone`)
VALUES ('acc0002', NULL, 'AirVia Ltd', 'mr', 'boris', 'berezovsky', '02073218523', NULL);
INSERT INTO `BAPERS`.`Customer_Account` (`account_number`, `email`,
`account_holder_name`, `title`, `first_name`, `surname`, `house_phone`, `mobile_phone`)
VALUES ('acc0003', NULL, 'InfoPharma Ltd', 'mr', 'alex', 'wright', '02073218001', NULL);

```

COMMIT;

-- Data for table `BAPERS`.`Job`

START TRANSACTION;

USE `BAPERS`;

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('abn54', '5 x 4 B& W copy negatives', 1900, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('acn54', '5 x 4 Colour copy negatives ', 1900, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('act108', '10 x 8 Colour copy transparency', 9600, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('act35', '35 mm Colour copy transparency', 2000, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('b108', '10 x 8 processing', 830, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('c108', '10 x 8 C41 processing', 830, NULL, false, 'acc0001');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('abc123', 'Test1', 1010, NULL, false, 'acc0002');

INSERT INTO `BAPERS`.`Job` (`job_id`, `description`, `amount_due`, `deadline`, `urgent`, `fk_account_number`) VALUES ('def123', 'Test2', 2020, NULL, false, 'acc0002');

COMMIT;

-- Data for table `BAPERS`.`Address`

START TRANSACTION;

USE `BAPERS`;

INSERT INTO `BAPERS`.`Address` (`address_line1`, `address_line2`, `postcode`, `city`, `country`, `fk_account_number`) VALUES ('northhampton square', NULL, 'ec1v 0hb', 'london', 'england', 'acc0001');

INSERT INTO `BAPERS`.`Address` (`address_line1`, `address_line2`, `postcode`, `city`, `country`, `fk_account_number`) VALUES ('12 bond street', NULL, 'wclv 8hu', 'london', 'england', 'acc0002');

INSERT INTO `BAPERS`.`Address` (`address_line1`, `address_line2`, `postcode`, `city`, `country`, `fk_account_number`) VALUES ('25 bond street', '', 'wclv 8ls', 'london', 'england', 'acc0003');

COMMIT;

-- Data for table `BAPERS`.`Task`

START TRANSACTION;

USE `BAPERS`;

INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`, `duration`) VALUES (1, 'use of large copy camera', 'copy room', 'cr25', 1900, '02:00:00');

INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`, `duration`) VALUES (2, 'black and white film processing', 'development area', 'dr12', 4950, '01:00:00');

```

INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`,
`duration`) VALUES (3, 'bag up', 'packing departments', 'pr10', 600, '30:00');
INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`,
`duration`) VALUES (4, 'colour film processing', 'development area', 'dr25', 8000,
'01:30:00');
INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`,
`duration`) VALUES (5, 'colour transparency processing', 'development area', 'dr100',
11030, '03:00:00');
INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`,
`duration`) VALUES (6, 'use of small copy camera', 'copy room', 'cr16', 830, '01:15:00');
INSERT INTO `BAPERS`.`Task` (`task_id`, `description`, `location`, `shelf_slot`, `price`,
`duration`) VALUES (7, 'mount transparencies', 'finishing room', 'fr5', 5550, '45:00');

```

```
COMMIT;
```

```

-----
-- Data for table `BAPERS`.`Discount_Plan`
-----

```

```

START TRANSACTION;
USE `BAPERS`;
INSERT INTO `BAPERS`.`Discount_Plan` (`type`) VALUES ('fixed');
INSERT INTO `BAPERS`.`Discount_Plan` (`type`) VALUES ('variable');
INSERT INTO `BAPERS`.`Discount_Plan` (`type`) VALUES ('flexible');

```

```
COMMIT;
```

```

-----
-- Data for table `BAPERS`.`Job_Task`
-----

```

```

START TRANSACTION;
USE `BAPERS`;
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abn54', 1, '2018-01-13 12:00:00', '2018-01-13 12:20:00', '0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abn54', 2, '2018-01-13 12:30:00', '2018-01-13 13:10:00', '0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abn54', 3, '2018-01-13 13:20:00', '2018-01-13 13:30:00', '0004');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('acn54', 1, '2018-01-13 12:20:00', '2018-01-13 12:55:00', '0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('acn54', 4, '2018-01-13 13:10:00', '2018-01-13 13:50:00', '0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('acn54', 3, '2018-01-13 14:00:00', '2018-01-13 14:10:00', '0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act108', 1, '2018-01-13 12:55:00', '2018-01-13 14:35:00',
'0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act108', 5, '2018-01-13 14:40:00', '2018-01-13 15:10:00',
'0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act108', 3, '2018-01-13 15:20:00', '2018-01-13 15:30:00',
'0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act35', 1, '2018-01-13 14:40:00', '2018-01-13 15:00:00', '0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act35', 5, '2018-01-13 15:10:00', '2018-01-13 15:30:00', '0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('act35', 7, '2018-01-13 15:35:00', '2018-01-13 16:00:00', '0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('b108', 2, '2018-01-13 12:20:00', '2018-01-13 12:50:00', '0005');

```



```

INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('b108', 3, '2018-01-13 12:50:00', '2018-01-13 13:00:00', '0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('c108', 4, '2018-01-13 13:10:00', '2018-01-13 13:30:00', '0005');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('c108', 3, '2018-01-13 13:35:00', '2018-01-13 13:45:00', '0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 1, '2018-01-14 13:35:00', '2018-01-14 13:45:00',
'0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 2, '2018-01-14 15:35:00', '2018-01-14 16:35:00',
'0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 3, '2018-01-14 17:35:00', '2018-01-14 18:35:00',
'0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 4, '2018-01-14 19:35:00', '2018-01-14 20:35:00',
'0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 5, '2018-01-14 21:35:00', '2018-01-14 22:35:00',
'0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 6, '2018-01-14 23:35:00', '2018-01-15 00:35:00',
'0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('abc123', 7, '2018-01-15 00:45:00', '2018-01-15 02:00:00',
'0004');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 1, '2018-01-15 05:45:00', '2018-01-15 06:45:00',
'0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 2, '2018-01-15 07:45:00', '2018-01-15 08:45:00',
'0001');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 3, '2018-01-15 14:45:00', '2018-01-15 16:45:00',
'0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 4, '2018-01-15 16:45:00', '2018-01-15 18:45:00',
'0002');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 5, '2018-01-15 22:45:00', '2018-01-15 23:00:00',
'0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 6, '2018-01-15 23:45:00', '2018-01-16 00:30:00',
'0003');
INSERT INTO `BAPERS`.`Job_Task` (`fk_job_id`, `fk_task_id`, `start_time`, `end_time`,
`fk_staff_id`) VALUES ('def123', 7, '2018-01-16 00:45:00', '2018-01-16 02:45:00',
'0004');

```

COMMIT;

```

-----
-- Data for table `BAPERS`.`Discount`
-----

```

START TRANSACTION;

USE `BAPERS`;

```

INSERT INTO `BAPERS`.`Discount` (`fk_type`, `fk_account_number`) VALUES ('fixed',
'acc0001');

```

```

INSERT INTO `BAPERS`.`Discount` (`fk_type`, `fk_account_number`) VALUES ('flexible',
'acc0002');

```

```

INSERT INTO `BAPERS`.`Discount` (`fk_type`, `fk_account_number`) VALUES ('variable',

```

```
'acc0003');
```

```
COMMIT;
```

```
-- Data for table `BAPERS`.`Task_Discount`
```

```
START TRANSACTION;
```

```
USE `BAPERS`;
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 1, 0.01);
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 2, 0.01);
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 3, 0);
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 4, 0.02);
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 5, 0);
```

```
INSERT INTO `BAPERS`.`Task_Discount` (`fk_account_number`, `fk_task_id`, `percentage`)
```

```
VALUES ('acc0002', 6, 0.02);
```

```
COMMIT;
```

```
-- Data for table `BAPERS`.`Discount_Band`
```

```
START TRANSACTION;
```

```
USE `BAPERS`;
```

```
INSERT INTO `BAPERS`.`Discount_Band` (`fk_account_number`, `price`, `percentage`) VALUES
```

```
('acc0003', 0, 0);
```

```
INSERT INTO `BAPERS`.`Discount_Band` (`fk_account_number`, `price`, `percentage`) VALUES
```

```
('acc0003', 100000, 0.01);
```

```
INSERT INTO `BAPERS`.`Discount_Band` (`fk_account_number`, `price`, `percentage`) VALUES
```

```
('acc0003', 200000, 0.02);
```

```
INSERT INTO `BAPERS`.`Discount_Band` (`fk_account_number`, `price`, `percentage`) VALUES
```

```
('acc0001', 0, 0.03);
```

```
COMMIT;
```

12.2 Reports

The reports will utilise the stored procedures that have been detailed above.

12.2.1 Summary Performance Report

This report requires the 'night_shift()', 'day_shift1()' and 'day_shift2()' procedures. Below details how to obtain the correct output.

```
call night_shift('2018-01-13','2018-01-17');
```

Create the Night Shift report according to input value with parameter(Start_date,End_date), If End_date values is later then datas date, it will return datas between start_date till the end of the data's date inclusively.

Outcome: Hours on each location according to date and Total hours on each location on period specify by the user in the parameter.

	Date	Copy_Room	Development	Finishing	Packing
►	2018-01-13	NULL	NULL	NULL	NULL
	2018-01-14	01:00:00	NULL	01:15:00	NULL
	2018-01-15	00:45:00	00:15:00	02:00:00	NULL
	2018-01-16	NULL	NULL	NULL	NULL

```
call day_shift1('2018-01-13','2018-01-17');
```

Create the Day Shift1 report according to input value with parameter(Start_date,End_date), If End_date values is later then datas date, it will return datas between start_date till the end of the data's date inclusively.

Outcome: Hours on each location according to date and Total hours on each location on period specify by the user in the parameter.

	Date	Copy_Room	Development	Finishing	Packing
►	2018-01-13	02:35:00	02:10:00	NULL	00:40:00
	2018-01-14	00:10:00	NULL	NULL	NULL
	2018-01-15	01:00:00	01:00:00	NULL	NULL
	2018-01-16	NULL	NULL	NULL	NULL

```
call day_shift2('2018-01-13','2018-01-17');
```

Create the Day Shift2 report according to input value with parameter(Start_date,End_date), If End_date values is later then datas date, it will return datas between start_date till the end of the data's date inclusively.

Outcome: Hours on each location according to date and Total hours on each location on period specify by the user in the parameter.

	Date	Copy_Room	Development	Finishing	Packing
►	2018-01-13	00:20:00	00:50:00	00:25:00	00:10:00
	2018-01-14	NULL	03:00:00	NULL	01:00:00
	2018-01-15	NULL	02:00:00	NULL	02:00:00
	2018-01-16	NULL	NULL	NULL	NULL

```
call summary('2018-01-13','2018-01-17');
```

Create Summary Report according to input value with parameter(Start_date,End_date), if End_date values is later than datas date, it will return dates between Start_date till the end of the data's date inclusively.

Outcome: Total Hours from each Shift from the period specify by the parameter and total hours for each location neglect from which shift.

	Title	Copy_Room	Development	Finishing	Packing
►	Day Shift1	03:45:00	03:10:00	HULL	00:40:00
	Day Shift2	00:20:00	05:50:00	00:25:00	03:10:00
	Night Shift	01:45:00	00:15:00	03:15:00	HULL

12.2.2 Individual performance report

```
call ipr('all');
```

Create Individual Performance Report with parameter(ID), which ID represents the staff_id in the database. User can also input 'all' inside the parameter to get every technician performance report with the total effort for each individual and overall effort which com-bine all individual's hours.

Outcome: Details of the selected individual's job_task, time spend on each task and total time spend on task by selected individual, and total effort(Total time spend on task) from the report generate.)

Name	Code	Task IDs	Department	Date	Start Time	Time Taken
john nash	abn54	1	copy room	2018-01-13	12:00:00	00:20:00
john nash	acn54	1	copy room	2018-01-13	12:20:00	00:35:00
john nash	act108	1	copy room	2018-01-13	12:55:00	01:40:00
john nash	act35	1	copy room	2018-01-13	14:40:00	00:20:00
john nash	abc123	1	copy room	2018-01-14	13:35:00	00:10:00
john nash	abc123	2	development area	2018-01-14	15:35:00	01:00:00
john nash	def123	1	copy room	2018-01-15	05:45:00	01:00:00
john nash	def123	2	development area	2018-01-15	07:45:00	01:00:00
julie abbot	acn54	3	packing departments	2018-01-13	14:00:00	00:10:00
julie abbot	act108	3	packing departments	2018-01-13	15:20:00	00:10:00
julie abbot	act35	7	finishing room	2018-01-13	15:35:00	00:25:00
julie abbot	b108	3	packing departments	2018-01-13	12:50:00	00:10:00
julie abbot	c108	3	packing departments	2018-01-13	13:35:00	00:10:00
julie abbot	abc123	3	packing departments	2018-01-14	17:35:00	01:00:00
julie abbot	abc123	4	development area	2018-01-14	19:35:00	01:00:00
julie abbot	def123	5	development area	2018-01-15	22:45:00	00:15:00
julie abbot	def123	6	copy room	2018-01-15	23:45:00	00:45:00
lee hong	abn54	2	development area	2018-01-13	12:30:00	00:40:00
lee hong	acn54	4	development area	2018-01-13	13:10:00	00:40:00
lee hong	act108	5	development area	2018-01-13	14:40:00	00:30:00
lee hong	act35	5	development area	2018-01-13	15:10:00	00:20:00
lee hong	abc123	5	development area	2018-01-14	21:35:00	01:00:00
lee hong	abc123	6	copy room	2018-01-14	23:35:00	01:00:00
lee hong	def123	3	packing departments	2018-01-15	14:45:00	02:00:00
lee hong	def123	4	development area	2018-01-15	16:45:00	02:00:00
marina scott	abn54	3	packing departments	2018-01-13	13:20:00	00:10:00
marina scott	abc123	7	finishing room	2018-01-15	00:45:00	01:15:00

Graphical User Interface Models

1. Pages

1.1. Page Tree

Home

Page 1

Page 2

Page 3

Page 4

Customer Account

Intervals

Backup

Users

Report

Place Order

Create Account

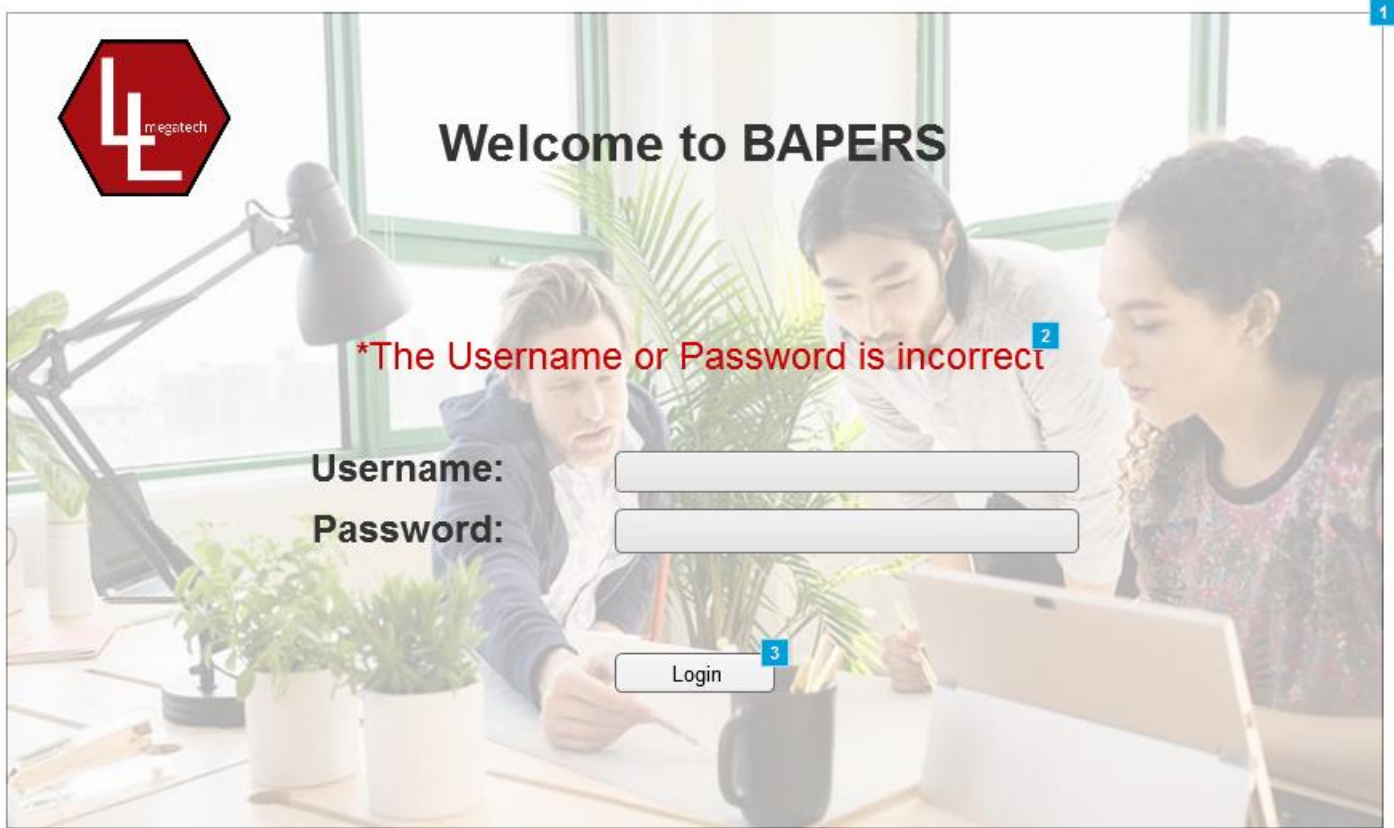
Job Process

Payment

Tasks

1.2. Home

1.2.1. User Interface

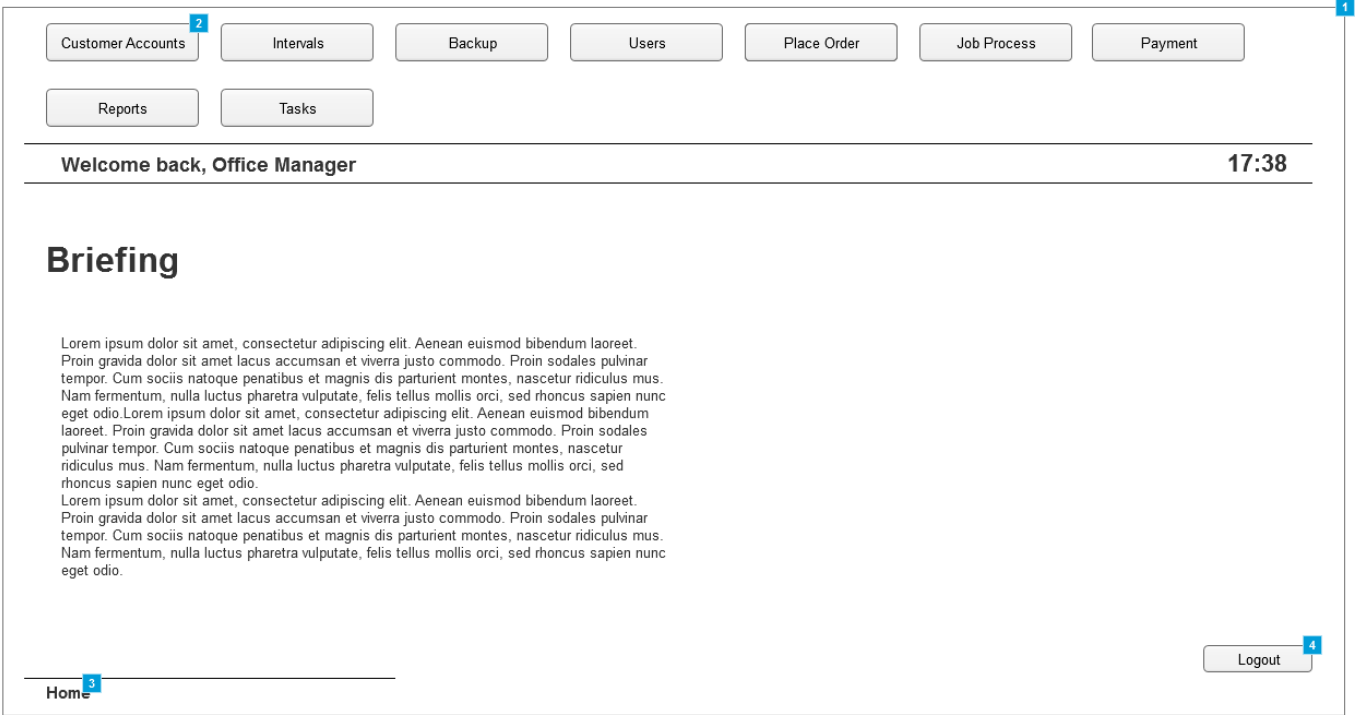


1.2.2. Widget Table

Footnote	Description	Risk
1	Main login screen	Low
2	Text that appears if the login details introduced are wrong	
3	When pressed, depending on the user and the details introduced, it either gives acces to the system or not.	

1.3. Page 1

1.3.1. User Interface

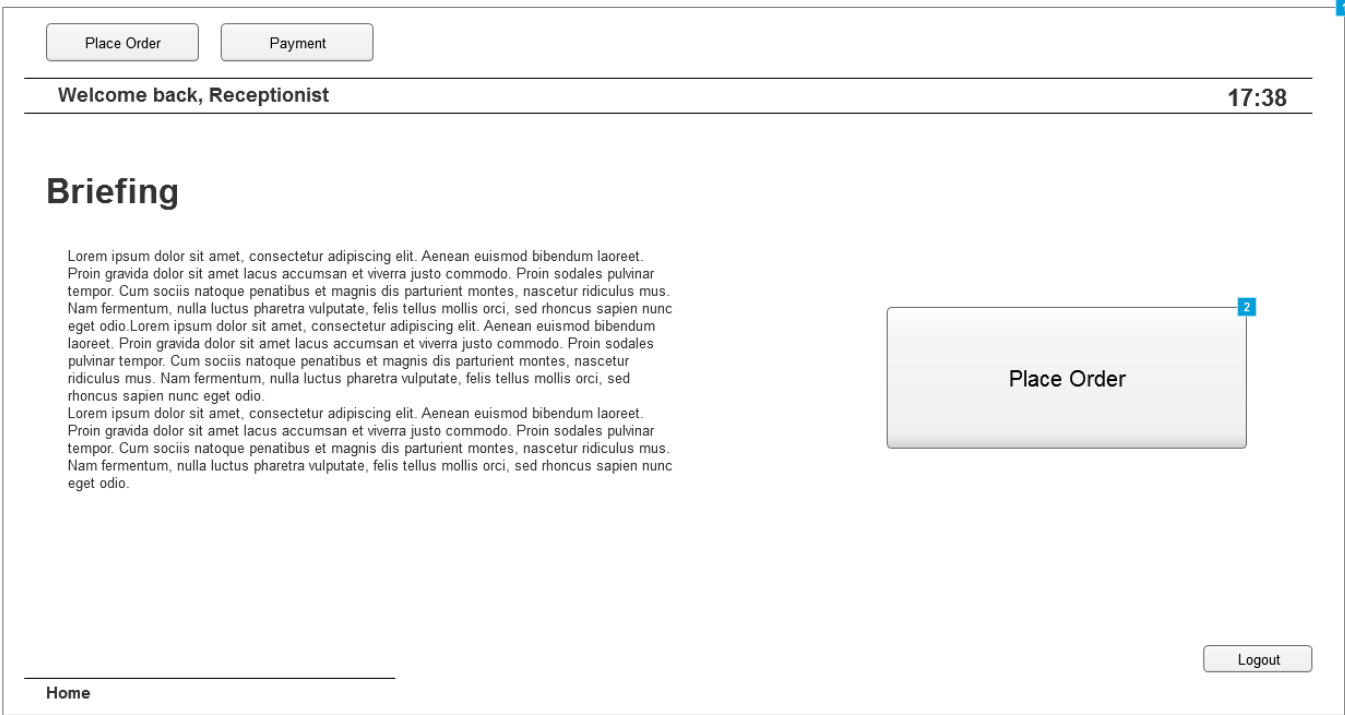


1.3.2. Widget Table

Foot-note	Description
1	Home screen panel for the Office Manager
2	Buttons that when clicked, take the user to the screen described in the tab. Buttons change depending on the users and their permissions
3	Displays the system path, for easier understanding where in the system the user is located. Can be clicked to go to a previous screen in that path,
4	Button that, when clicked, logs the user out of the system.

1.4. Page 2

1.4.1. User Interface

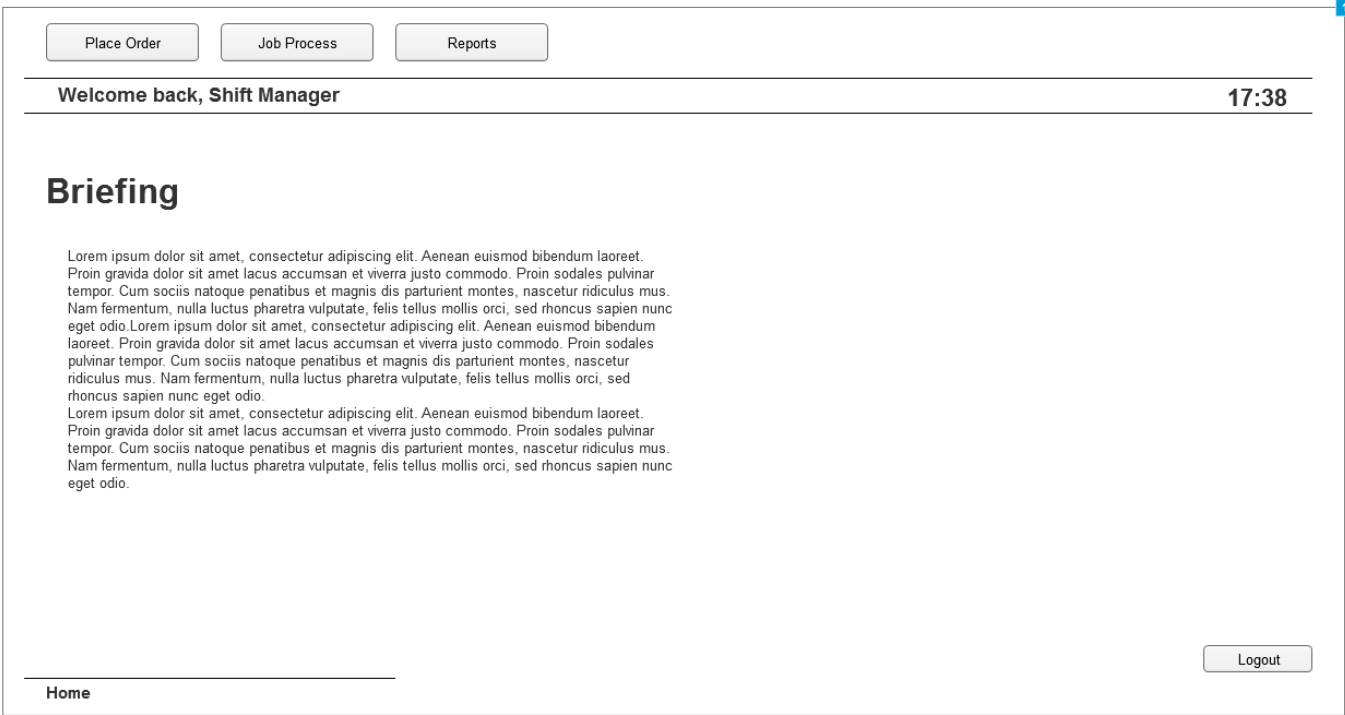


1.4.2. Widget Table

Footnote	Description
1	Home screen panel for the Receptionist
2	Quick access to "Place Order Tab" since is one of the main tasks.

1.5. Page 3

1.5.1. User Interface



1.5.2. Widget Table

Footnote	Description
1	Home screen panel for the Shift Manager

1.6. Page 4

1.6.1. User Interface

Job Process

Welcome back, Technician

17:38

Briefing

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean euismod bibendum laoreet. Proin gravida dolor sit amet lacus accumsan et viverra justo commodo. Proin sodales pulvinar tempor. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam fermentum, nulla luctus pharetra vulputate, felis tellus mollis orci, sed rhoncus sapien nunc eget odio. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean euismod bibendum laoreet. Proin gravida dolor sit amet lacus accumsan et viverra justo commodo. Proin sodales pulvinar tempor. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam fermentum, nulla luctus pharetra vulputate, felis tellus mollis orci, sed rhoncus sapien nunc eget odio.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean euismod bibendum laoreet. Proin gravida dolor sit amet lacus accumsan et viverra justo commodo. Proin sodales pulvinar tempor. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam fermentum, nulla luctus pharetra vulputate, felis tellus mollis orci, sed rhoncus sapien nunc eget odio.

Home

Logout

1.6.2. Widget Table

Footnote	Description
1	Home screen panel for the Technician

1.7. Customer Account

1.7.1. User Interface

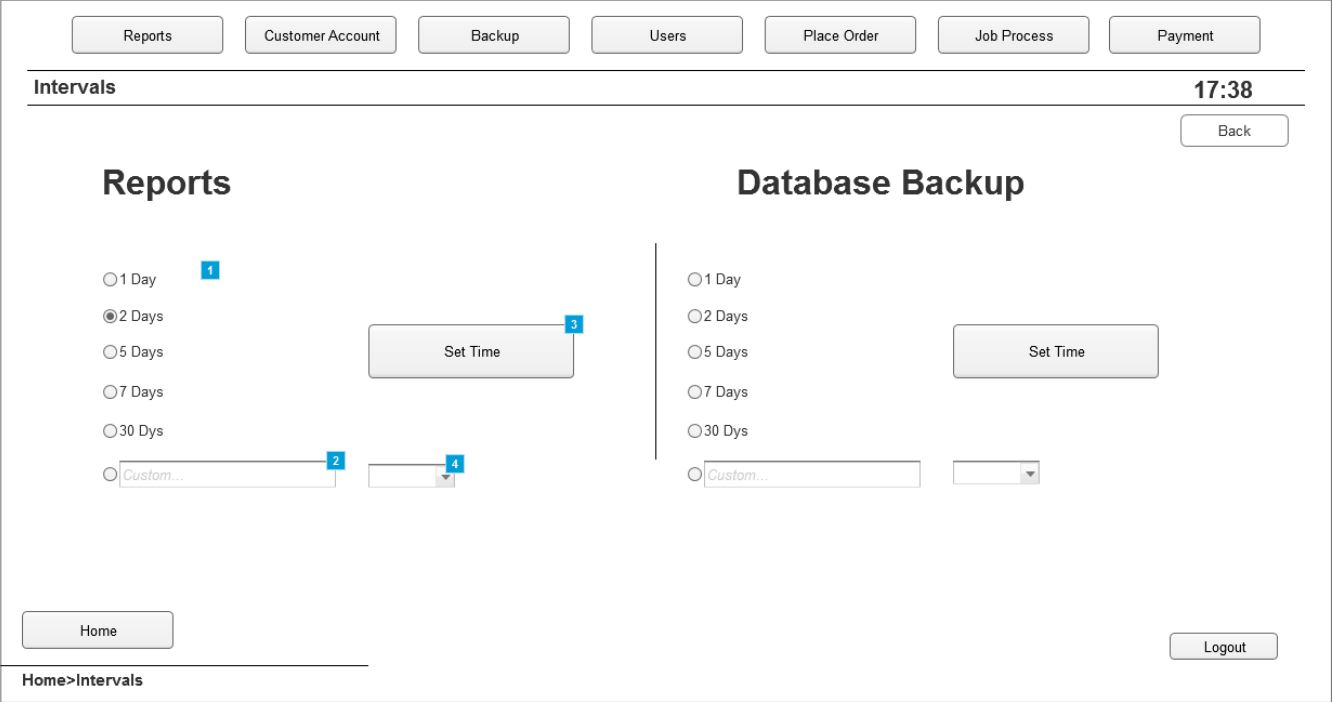
The screenshot shows a web application interface for managing customer accounts. At the top, there is a navigation bar with buttons: Reports, Intervals, Backup, Users, Place Order, Job Process, and Payment. Below this, the main header area displays 'Customer Accounts' on the left, a clock showing '17:38' on the right, and a 'Back' button. A search box is located below the header. The main content area is titled 'Leory Godwing - Valuable - Flexible Discount (3%)'. On the left, a list box contains several customer names, with 'Leroy Godwin' highlighted in bold. In the center, there are three buttons: 'Upgrade', 'Downgrade', and 'Activate Account'. To the right of these buttons, there is a panel for discount plans with options for 'Fixed Discount', 'Variable Discount', and 'Flexible Discount'. Below this, there is a section for tasks with checkboxes and percentage values. At the bottom left, there is a 'Home' button, and at the bottom right, there is a 'Logout' button. The breadcrumb 'Home>Customer Accounts' is visible at the bottom left.

1.7.2. Widget Table

Foot-note	Description
1	Panel where the existing customer accounts are managed
2	Search box, to look-up existing customer accounts, once chosen, upgrading/downgrading options are available as well as updating the discount plans associated with the account
3	List box where all the existing customer accounts are displayed. -The ones in Bold Display Valued Customers -The normal ones display normal customers -The ones in red display Suspended Customer Accounts
4	Upgrades the customer to the "Valued" status, only available if the customer is not already in this status
5	Downgrades the customer from the "Valued" status to the "Normal" one.
6	Reactivates a "Suspended" account
7	Panel displaying the available discount plans
8	Tab that allows to choose tasks and apply discounts to each one. Appears only if "Flexible Discount" is selected
9	If a discount for a task need to be deleted, unticking the box will do this. Selecting an existing task from the list and adding a different percentage will overwrite the previous one
10	The percentage of the discount will be introduced here
11	Button that, when clicked, confirms the discount options selected
12	Button that allows the user to go back to the previous screen
13	Button that takes the user to the home tab

1.8. Intervals

1.8.1. User Interface

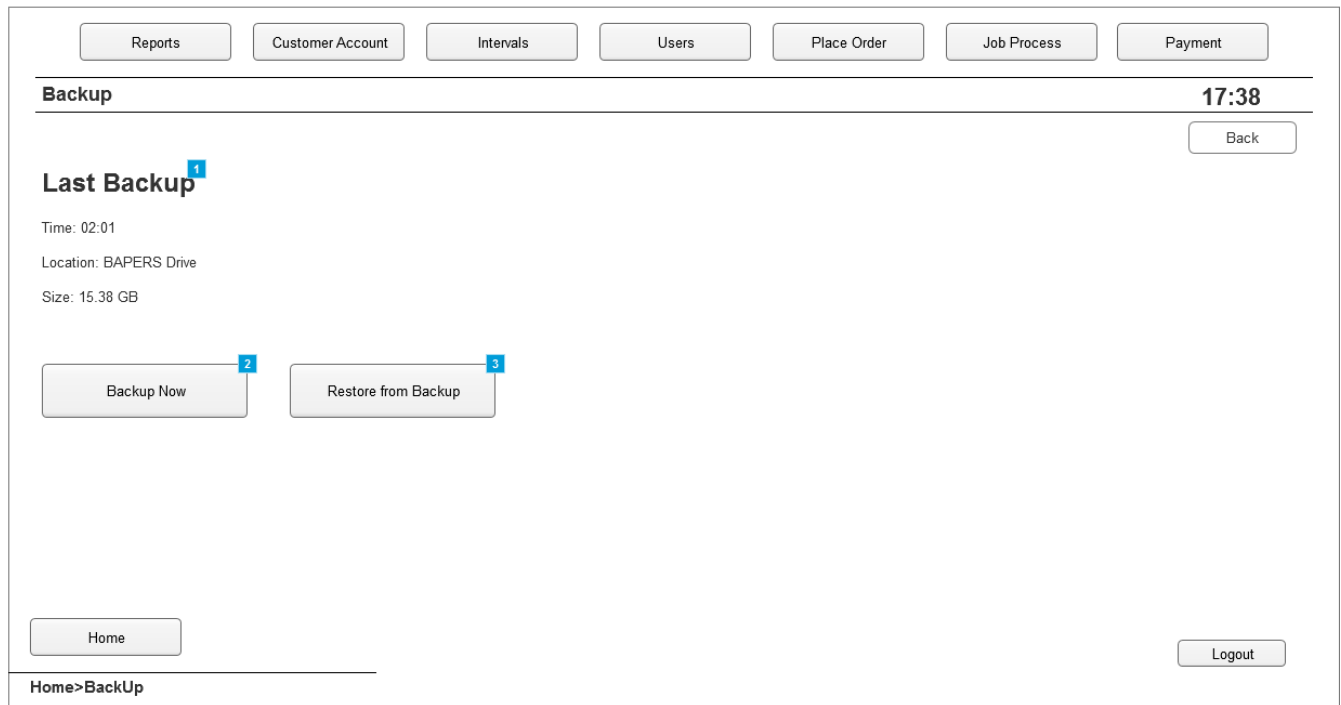


1.8.2. Widget Table

Footnote	Description
1	Premade options to choose the time intervals at which reports are created. Similar for Database Backup
2	Panel to introduce a custom time period for generating the repots. Same can be done to the Database Backup
3	Button that, when clicked, sets the time intervals at which reports are created.
4	Drop list allowing to set minutes, hours or days for the custom time intervals

1.9. Backup

1.9.1. User Interface



1.9.2. Widget Table

Footnote	Description
1	Information about the last backup created
2	Button that, when clicked, creates a system backup
3	Button that, when clicked, restores data from the last backup created

1.10. Users

1.10.1. User Interface

ReportsCustomer AccountIntervalsBackupPlace OrderJob ProcessPayment

Users17:38

Back

User ID...

First Name...

Surname...

Password...

User Type:

Create User Account

User Account Successfully Created

Home

Logout

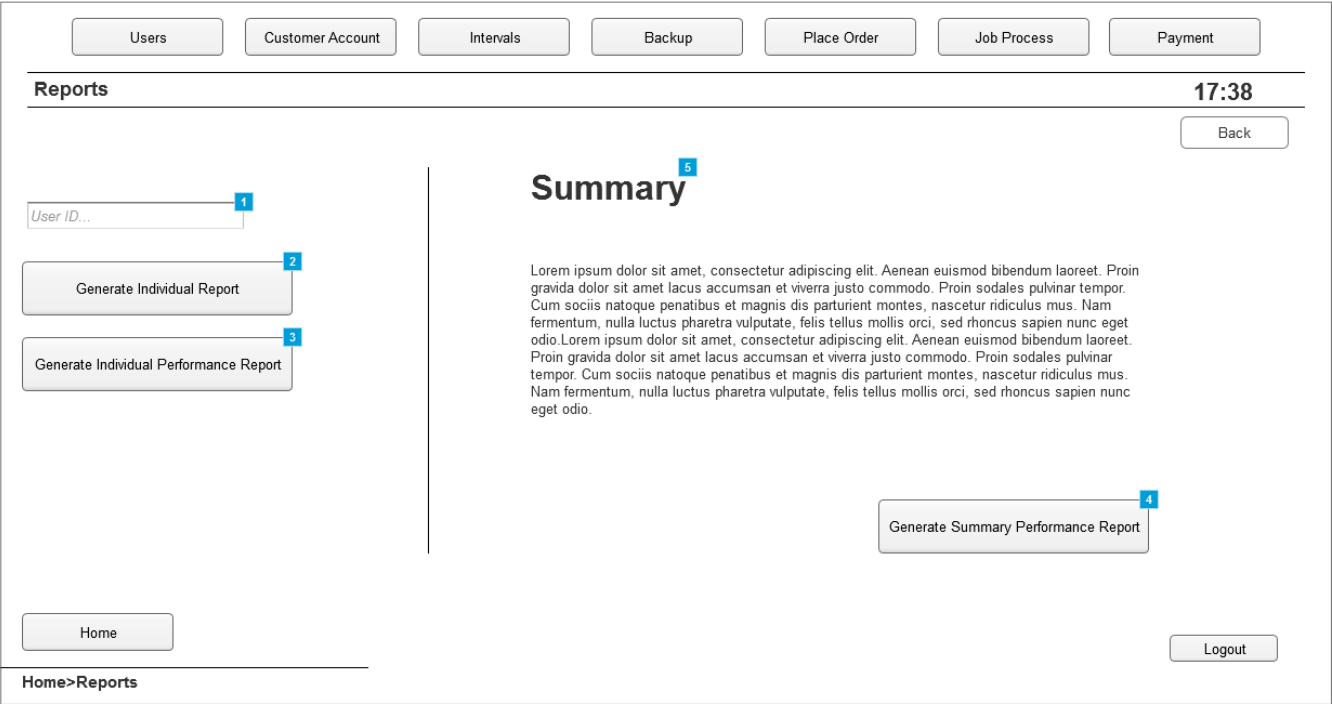
Home>Intervals

1.10.2. Widget Table

Footnote	Description
1	Button that, when clicked, creates the user account with the details introduced
2	Tab where new users are added to the system
3	Window that pops up when an account has been successfully created

1.11. Report

1.11.1. User Interface



1.11.2. Widget Table

Footnote	Description
1	Box where user ID is introduced. This refers to the system users. Once a user is selected, more options are available
2	Button that, when clicked, generates an individual report for the user selected
3	Button that, when clicked, generates an individual performance report for the user selected
4	Button that, when clicked, generates a summary performance report
5	Summary and details about it

1.12. Place Order

1.12.1. User Interface

The screenshot displays the 'Place Order' web application interface. At the top, a navigation bar contains buttons for Reports, Customer Account, Intervals, Users, Place Order, Job Process, and Payment. The main header shows 'Place Order' on the left and '17:38' on the right, with a 'Back' button. The interface is divided into several sections: 1. A search bar labeled 'Search Customer...' with a blue '1' annotation. 2. A 'Create Customer Account' button with a blue '2' annotation. 3. An 'Urgent Job' section with a radio button for 'Yes' (annotated with blue '3') and a 'Record Deadline...' input field (annotated with blue '4'). 4. A 'Special Requests' section with a radio button for 'Yes' (annotated with blue '5') and a 'Describe Requests...' text area (annotated with blue '6'). 5. A 'Job Requirements' section with a list of four checkboxes, the first and last of which are checked (annotated with blue '7'). 6. A 'Confirm' button (annotated with blue '8'). 7. A 'Print Label and Receipt' button (annotated with blue '9'). 8. A right sidebar displaying job details: Job Number: 439, Price: 255 \$, and Deadline: 04/03/2018. 9. A 'Home' button in the bottom left and a 'Logout' button in the bottom right. A breadcrumb trail at the bottom left reads 'Home>Place Order'.

1.12.2. Widget Table

Footnote	Description
1	Panel where customer accounts can be searched, if created, to allow for placing an order
2	Button that, when clicked, takes the user to the "Create Account" window. Used in case the customer does not have an account yet
3	Option to select if job placed is urgent.
4	Panel to introduce deadline if job is urgent
5	Option to select if special requests exist about the job
6	Panel to introduce details about the request
7	List of all tasks available. Checked only the ones which are needed for the job
8	Button that, when clicked, generates the job number and final price
9	Button that, when clicked, Assigns the job and prints the label and the receipt.

1.13. Create Account

1.13.1. User Interface

ReportsCustomer AccountIntervalsBackupPlace OrderJob ProcessPayment

Create Account17:38

Back

Customer ID...First Name...Address Line 1...

Email...Surname...Address Line 2...

Company Name.....House Phone...Postcode...

Title...Mobile Phone...City...

Country...

Create Customer Account2

Home

Logout

Home>Place Order>Create Account

1.13.2. Widget Table

Footnote	Description
1	Tab where new customer accounts are created
2	Button that, when clicked, creates the customer account with the details introduced in each field.

1.14. Job Process

1.14.1. User Interface

The screenshot displays the 'Job Process' user interface. At the top, a navigation bar contains buttons for 'Reports', 'Customer Account', 'Intervals', 'Backup', 'Place Order', 'Users', and 'Payment'. The main header area shows 'Job Process' on the left, the time '17:38' on the right, and a 'Back' button. Below the header, there is a 'Job Number...' input field (callout 2). A list of tasks with checkboxes is shown (callout 3), with the first two checked. A 'Refresh' button (callout 5) is positioned to the right of the list. At the bottom of the task list, there are two buttons: 'Update Job Status' (callout 4) and 'Alert Incomplete' (callout 6). A 'Home' button is located at the bottom left, and a 'Logout' button is at the bottom right. The breadcrumb 'Home>Job Process' is visible at the very bottom.

1.14.2. Widget Table

Footnote	Description
1	Tab where the ongoing jobs can be managed
2	Panel where the job number is introduced to get details about a job
3	List with the tasks available. Those ticked indicate that the particular task is completed
4	Button that, when clicked, updates the status of the jobs, A task must be ticked or unticked to be able to update
5	Button that, when clicked, refreshes the list to see if changes have been made
6	Button that is click to alert that the job is not going to be completed on time

1.15. Payment

1.15.1. User Interface

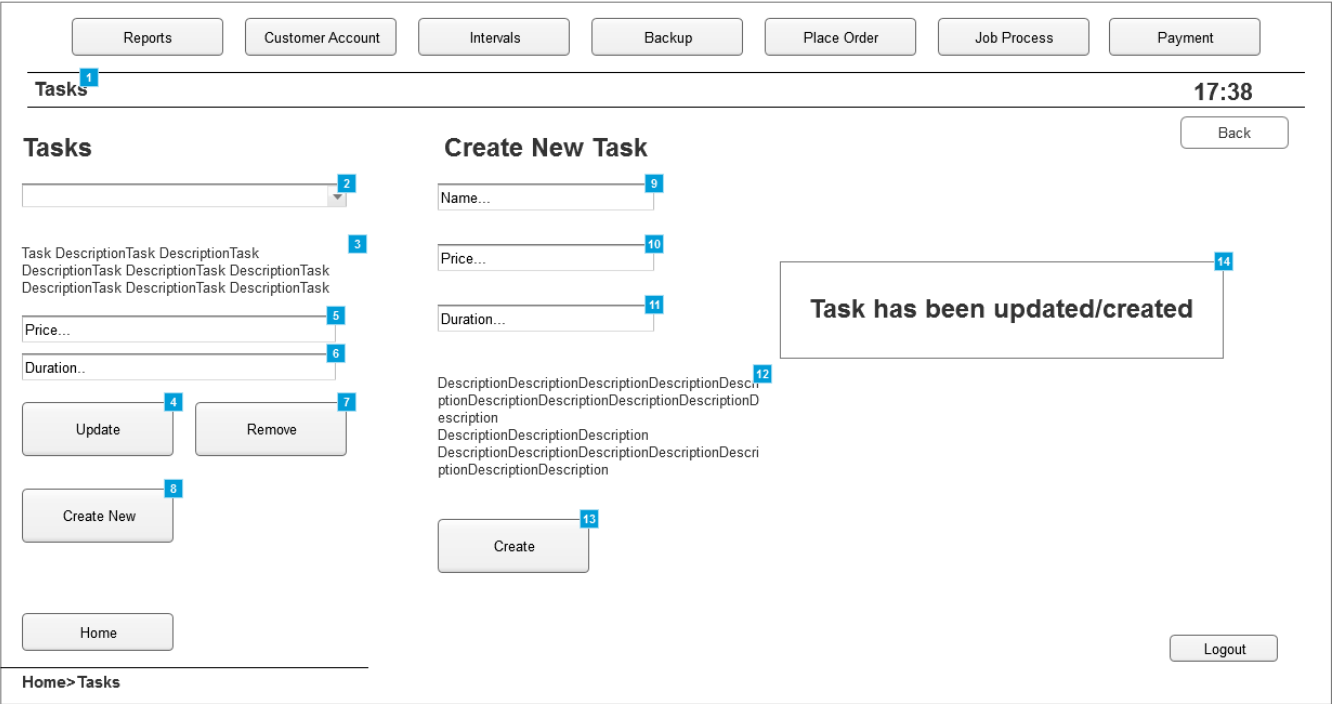
The screenshot shows a web application interface for the 'Payment' section. At the top, there is a navigation bar with buttons: Reports, Customer Account, Intervals, Backup, Place Order, Job Process, and Payment. The 'Payment' button is active. Below the navigation bar, the page title is 'Payment' and the time is '17:38'. A 'Back' button is located in the top right corner. The main content area is divided into two columns. The left column contains a search bar for 'Customer ID...' (callout 1), account status information (callout 2), a list of jobs (callout 3), and a 'Revoke and Pay' button (callout 7). The right column displays a pop-up window for 'Customer Gabriel Baeasu (90210)' (callout 4) with the message 'Has missed payment' and a 'Print Reminder Letter' button (callout 5). Below the main content area, there is a 'Home' button and a 'Logout' button. The breadcrumb 'Home>Payment' is shown at the bottom left.

1.15.2. Widget Table

Footnote	Description
1	Panel where a customer is searched to get details about the job placed
2	Details about the customer account,
3	List of all the jobs placed, only one will be displayed if the customer is not "Valued" If the job is ticked, it means it will be paid for in the next payment.
4	Pop-up window alerting the Office Manager that a customer has missed payment
5	Button that, when clicked will print the reminder letter for the user
6	Panels where information about card or cash payment is introduced
7	Button that, when clicked, completes payment and revokes suspension of an account if account is suspended.

1.16. Tasks

1.16.1. User Interface



1.16.2. Widget Table

Footnote	Description
1	Tab where tasks can be managed
2	List of all the tasks available in the system
3	Description of the task selected
4	Button that. when clicked, updates any details changed about the selected task
5	Price of the selected task
6	Time it takes for the task to be completed
7	Button that. when clicked, removes the selected task
8	Button that. when clicked, displays the "Create Task" options
9	Panel to introduce the name of the new task
10	Panel to introduce the price of the new task
11	Panel to introduce the time taken for the new task to be completed
12	Description of the new task
13	Button that, when clicked, creates and adds the task to the list of existing tasks.
14	Window that pops up when a task has been successfully created or updated