**Implementation Report:**

## **Team 17 (Digital Inspirations)**



# **Team members:**

Anthony Gibson, Sharmistha Dutt, Hamzah Malik, Mariia Onyshchenko, Devina Thakker, Martin Ivanov

**Contents**

[**Team 17 (Digital Inspirations)** 1](#_Toc512259699)

[**Team members:** 1](#_Toc512259700)

[**1.** **Source code compilation** 3](#_Toc512259701)

[1.1 Program structure and what gets compiled 4](#_Toc512259702)

[**2.** **Run-time components** 5](#_Toc512259703)

[2.1 Component Diagram 5](#_Toc512259704)

[**3.** **Deployment diagram** 5](#_Toc512259705)

[**4.** **System Testing** 6](#_Toc512259706)

[4.1 Test-cases 6](#_Toc512259707)

[4.1 Non-Functional requirements using Volere template 15](#_Toc512259708)

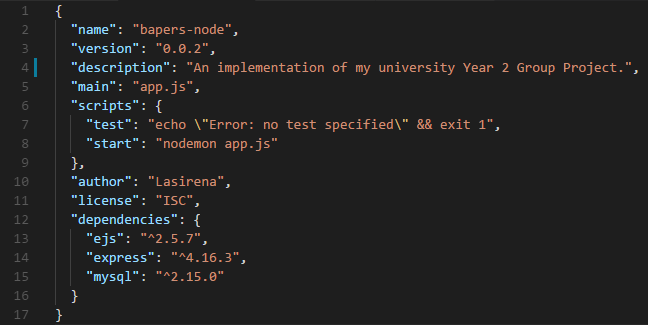
[4.2 Security testing 15](#_Toc512259709)

[4.2 Performance Testing 16](#_Toc512259710)

# **Source code compilation**

Our project is a web application with the back-end server written in Node.js. Node.js is essentially an implementation of the JavaScript language as an engine. It is fundamentally different from the JavaScript that web browsers use to interpret front-end code because unlike JavaScript, which is an interpreted language (meaning it never gets compiled), Node.js code is continuously compiled at run-time for further optimization. It is, however, different from other compiled languages, because despite being compiled it does not produce an executable file.

Instead, Node.js relies on a configuration file which specifies all information about a project, its dependencies and entry point (the main JavaScript file from which the app is to execute). The file is written in JavaScript Object Notation (JSON) and is called package.json, located in the app’s root folder. Here is an example of what package.json can look like:



The ‘name’, ‘version’, ‘description’, ‘author’ and ‘license’ keys in this file are self-explanatory – this is just information used to describe the project.

The ‘main’ key specifies the filename and file path to the application’s main entry point – in this case it’s app.js, located in the root folder.

The ‘scripts’ key contains an array of script commands that can be used with the Node Package Manager (npm) from the command line interface. Node.js and NPM are tied very closely since NPM is used for installing all dependencies, starting the server, and generating/updating the package.json file. The script’s value specifies the code that is to be executed when the command ‘npm <script>’ is run from the command line. In this case, running ‘npm start’ will execute the command ‘nodemon app.js’ (which corresponds to starting the server using the Node Monitor package).

The ‘dependencies’ array lists all the node package dependencies of the project and their corresponding versions. If the project is installed on a new machine with no dependencies available, running the ‘npm install’ command from the project folder will install all of the dependencies required in the package.json file, with the version number that was specified.

## 1.1 Program structure and what gets compiled

The structure of our Node.js application can be roughly split into four major parts.

The first, app.js, is the entry point of our application. This is where all the primary configuration of the project, such as the HTTP server and Database configurations, occurs.

Aside from that, we have three major sub-folders: public, routes, and views.

The files in the ‘public’ folder are static files which do not get compiled but only served as-is as part of the webpages. The files that are usually stored in the public folder include front-end dependencies (such as Bootstrap, custom CSS, and the JavaScript code that is to be executed by the browser rather than the server) and various data files such as images or logos.

The ‘routes’ and ‘views’ are closely tied together and are usually the largest part of the project. These are compiled by Node at run-time. The routes folder contains JavaScript files which specify how the HTTP server is to behave in response to various type of page requests (such as GET and POST requests from the various parts in the site). Each route also has one or more corresponding views. The views folder contains the front-end content of web pages, however rather than it being static mark-up, the pages are written in what is called a templating engine. A templating engine allows the developer to inject server-side JavaScript into the web pages, which is essential for displaying data received from the database to generate the web pages according to the results received from an SQL response.

There are some other sub-folders and files in our project which have been split up from the main code for the sake of keeping the project a little more organized. The database configuration has been carried over to a separate module (db.js) in the root folder. Within the views folder, there is also a ‘partials’ subfolder. It refers to EJS partials which are snippets of code for the templating engine, separated into individual files so that they can later be included in more places than one, or included for the sake of reducing code clutter in the view files. In the root folder, there is a ‘queries’ folder within which the major database query functions have been separated into based on the part of the system that they interact with. This folder basically contains query sets as individual modules which can then be connected and used by the main program.

# **Run-time components**

Our project has several major node module dependencies, all of which are installed with the aforementioned Node Package Manager.

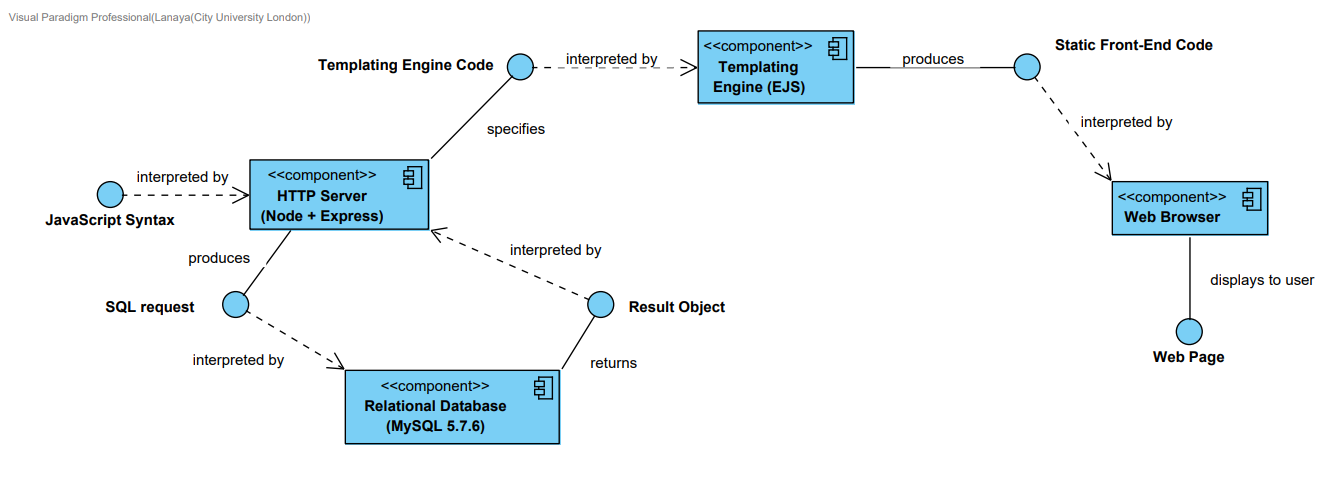
First of all, we are using Express.js as our HTTP server framework. Express is a standard in the Node developer community and it makes it very simple to set up a HTTP server with routes. Writing our own implementation of an HTTP server would be very hard and needless. We are also using express-validator to validate various form inputs.

We are also using EJS as our views templating engine. It doesn’t particularly matter what templating engine is used, but EJS is one of the more common choices, because it is the most likely to avoid syntax conflicts with front-end frameworks such as Angular2+. Other engines such as Jade are also a good choice, but stray from the default HTML syntax too much in some ways, and would be counter-intuitive to learn for someone who has been using HTML for a long time.

Since Node.js has no built-in support for non-relational databases, we are also using a node package called mysql as our database driver.

We also used bower as our tool for managing front-end dependencies and nodemon to manage auto-restarting of the server during the development stage every time changes to the code are made.

# 2.1 Component Diagram



Featured above is a component diagram, highlighting the run time components of our system. It models the process of how our JavaScript syntax ends up as information displayed to the user through a Web Page.

# **Deployment diagram**

A deployment diagram is not required for this project, since all the key components, including the client (web browser), the server and the database are running on the same local machine.

# **System Testing**

After the completion of the first deliverable, our team established some changes that needed to be implemented that would affect the ‘Implementation testing’ stage of the system.

Our use case diagram had changed for the benefit of our system and we had made changes to the use case priority list. This means that some of our use cases which previously were not as important have risen in their significance to the system; leading to a new set of test cases that we consider as the most important to be tested.

# 4.1 Test-cases

|  |  |
| --- | --- |
| **Use case ID:** UC26 | **Use case name:** CreateUserAccount |
| **Test Number:** 1 | |
| **Objective**: Test the main flow to see whether the system will create a user account correctly, allocating the user a distinct user ID. | |
| **Set up:** Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Employees’ subsection within the BAPERS System. This can be found from the main home page or from the drop-down list. | |
| **Expected results:**   1. The user account will be created. 2. The user will be given a unique user ID. 3. The new user will be placed within the database with user details provided. | |
| **Test:**   1. ‘Create a new user’ button is pressed. 2. User account information is entered for the users ‘First Name’, ‘Surname’ and ‘E-Mail’ respectively. The name ‘Timothy’ is entered for ‘First Name’, ‘Weng’ for ‘Surname’, and an e-mail account is assigned, named “TimothyWeng1998@hotmail.com”. From a drop- down list of ‘Employee Roles’ the role ‘Technician’ is selected and the user is given a username ‘TimWeng1’ with a password of ‘Qw3rTy’. 3. The button ‘Create’ is pressed to generate the customer account. | |
| **Test record:** Test partially passed to begin with, but account was not showing up despite being ‘created’. This was fixed, and the account now shows up as expected with a full list of the user information, with a distinct ID number. | |
| **Date:** 13th April 2018 | **Tester:** Anthony Gibson |
| **Result:** Failure, the system confirms the account is created but the account does not show up on the system itself. The account also appears in the database. Although this test partially passed, it is missing key functionality so is labelled as a failure. | |
| **Date:** 15th April 2018 | **Tester:** Hamzah Malik |
| **Result:** Passed, the user accounts now appear on the system. | |

|  |  |
| --- | --- |
| **Use case ID:** UC24 | **Use case name:** RecordPaymentAmount |
| **Test Number:** 2 | |
| **Objective:**  Test the main flow, to see if payments are correctly monitored on the system by creating a payment record and storing the details in the database | |
| **Set up:** Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Jobs page’. Only completed jobs must be available to apply a payment towards, so the jobs must be filtered by ‘All Completed Jobs’, rather than ‘All Jobs In-Progress’ or ‘All-Jobs’. The payment status should also be presented as ‘Not Paid’. | |
| **Expected results:**   1. A record of the payment and future payments will be stored and displayed within the BAPERS database. 2. The user/customer with a valid BAPERS account will be able to access the record on request. 3. The card details will be stored within the payment record. | |
| **Test:**   1. Filter through the customer accounts to find the account that needs to pay for the job. 2. Clicking ‘View’ on the job icon for the relevant customer props up their current jobs, indicating the urgency, duration, and final/base prices. For Customer ID ‘3’ a Job ID has been set of ‘18’ with an urgency of ‘Normal’, so there is a 0% surcharge. The base price and final price is set to £1910.00 as the customer is not given a discount. The Job Status is ‘Complete’ and the Payment Status is ‘Not Paid’. 3. Click ‘Record Payment’ and select payment type to ‘Card’, set the card type to ‘Visa’ and the expiry date to ‘03/2020’, the 16-digit card number is set to ‘5667854256102486’ and the security code is 755. Submit the card details and payment details. 4. Click the ‘Submit’ button. | |
| **Test record: All features of this use case worked correctly, the job is labelled as paid and the information can be viewed at any time of the transaction.** | |
| **Date:** 16 April 2018 | **Tester:** Anthony Gibson |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC23.1 | **Use case name:** PaymentDeclined **(Alt flow)** |
| **Test Number:** 2 | |
| **Objective:**  Testing alternative flow to see if a payment by card can be declined based on user input, such as invalid details or an invalid balance amount for a transaction. | |
| **Set up:**  Customer details for customer number ‘3’ with ID ‘18’ must be allocated to a payment type for the requested transaction, this is set as ‘Card’ respectively to test for any failures and CCV expiry date must not be surpassed (01/2020). | |
| **Expected results:**   1. Incorrect card details will display an error message “Transaction failed, please try again” 2. A card with insufficient funds will display an error message alternatively, stating “Transaction failed, please try again”. 3. User will be prompted to re-enter card details. | |
| **Test:**   1. For a pre-existing job with a job ID: 18 for Customer ID: 3, set the job to complete and record payment details. 2. Payments using card are intentionally entered incorrectly with a single digit value of ‘1’ instead of the 16-digit number for Customer 3. 3. User clicks the ‘Submit’ button and payment should be rejected. | |
| **Test record:** Error message does not appear, but functionality is working. Second test attempted to see if this error still occurred and after some changes within the implementation, the error message appeared successfully. | |
| **Date:** 17th April 2018 | **Tester:** Anthony Gibson |
| **Result:** Failed | |
| **Date:** 19th April 2018 | **Tester:** Hamzah Malik |
| **Result:** Passed | |

|  |  |
| --- | --- |
| **Use case ID:** UC1 | **Use case name:** CreateJob |
| **Test Number: 3** | |
| **Objective:** Test the main flow to see if front staff can create jobs using the BAPERS system. | |
| **Set up:** Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Jobs’ page where the ‘New Job’ button exists. A valid customer account (Customer ID 1) must be set up to have a job set under a customer. ‘AcceptUrgentJob’ and ‘PrintReceipt’ use cases must also be available, and unit tested. | |
| **Expected results:**   1. The job is created and is then recorded under the ‘Jobs’ database table, based on the jobs urgency. 2. The assigned date for completion is shown, and the surcharge for the cost of the job is shown. 3. The option to print an invoice appears for the customer. | |
| **Test:**   1. Create a job for the Customer ID:1, with an urgency set to ‘Custom Deadline’ (under 6 hours). The ‘Custom Duration(minutes)’ should be set to 180 and the surcharge set to 10%. 2. Tasks are set of ‘Use of large copy camera’ and ‘Colour film processing’ both set at quantity amount of: 1. Set the Custom job instructions to ‘N/A’. 3. Press the ‘Create’ button. | |
| **Test record:** All parts worked correctly, the job is created and the deadline, job status, payment status and task completion status is set. | |
| **Date:** 18 April 2018 | **Tester:** Hamzah Malik |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC9 | **Use case name:** UpdateTaskCompletion |
| **Test Number:** 4 | |
| **Objective:** Test the main flow to see if the system allows tasks progression to be updated. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Jobs’ page to view the relevant job which contains tasks for a specific customer.  The use cases ‘SetComplete’ and ‘SetInProgress’ should also be available and must be unit tested. | |
| **Expected results:**   1. The updated tasks will display within the “Jobs” database table and will be able to identify the progress/completion of a task. 2. The day the task is completed will be displayed as a date within the table and the day the job was started will be displayed. 3. The task will be completely updated and show the progression of the overall job for the customer. | |
| **Test:**   1. With the new job created in the test case before this, with the “Large Copy Camera” assigned to Customer ID: 1, click onto the jobs tab and view customer ID: 1. 2. Click the ‘Start’ button under ‘Start Task’ to set the job under ‘InProgress’ and not ‘Must Start First’. 3. Once the job has started, set the job to “Complete” to have the job’s “Mark Completion” as ‘Completed’. The date SHOULD also state when the job has been completed. | |
| **Test record:** All parts worked for this and the correct information was displayed, with relevant job update times and completion times, alongside the Mark Completion of the test set to ‘Completed’. | |
| **Date:** 16 April 2018 | **Tester:** Anthony Gibson |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC44 | **Use case name:** DeleteTask |
| **Test Number: 5** | |
| **Objective:** Test the main flow to see if the system allows tasks to be deleted. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Jobs’ page to view the relevant job which contains tasks for a specific customer. | |
| **Expected results:**   1. Staff members can delete the task under a job. 2. The task is removed, and the database is updated to show it no longer is in the system. | |
| **Test:**   1. With the new job created in the ‘CreateJob’ test case, the job will contain the task “Large Copy Camera”, which has been assigned to Customer ID: 1, click onto the jobs tab and view customer ID: 1. 2. Click” Edit Task” button and then “Delete Task”. 3. The system should delete the task from the list. | |
| **Test record:** All parts worked for this and the task was deleted successfully from the system. | |
| **Date:** 18th April 2018 | **Tester:** Anthony Gibson |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC17 | **Use case name:** SetDiscountPlan |
| **Test Number: 6** | |
| **Objective:** Test the main flow to see if the system allows the valid BAPERS account to set a discount plan “Fixed, variable or flexible” for a customer. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Customers’ page on the home menu. | |
| **Expected results:**   1. The default office manager account will set a discount plan to a valued customer. 2. The customer ‘Anthony Gibson’ will have their discount plan set to ‘variable discount’ which will be recorded and displayed within the Customer database table 3. The system will update and show Anthony Gibson as a valued customer. | |
| **Test:**   1. Create a new customer named “Anthony Gibson” with relevant customer details such as an e-mail set to ‘AGib123@hotmail.com’, an address set to ‘16B Holloway Road’, their city as ‘London’, their postcode as ‘N16 6BH’ and a phone number as ‘0795185436’. 2. After creating the customer, the system will lead back to the customers page and show Anthony is not a ‘Valued Customer’. Click ‘view’ and then ‘edit’ and set Anthony as a valued customer. 3. Select the button ‘Add discount plan’ and choose the variable discount to be 15%. 4. Click submit. | |
| **Test record: A**ll features of this worked correctly and a discount is set to the customers payment of all collective tasks. | |
| **Date:** 17 April 2018 | **Tester:** Hamzah Malik |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC17.1 | **Use case name:** IncorrectCustomerType**(Alt Flow)** |
| **Test Number: 6** | |
| **Objective:** Test the alternative flow to see if the system rejects the office manager from setting a discount plan to a non-valued customer account. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Customers’ page. Customer account must already be set up with name ‘Eva Bauyer’ and Account Status as ‘Normal’ instead of ‘Valued’. | |
| **Expected results:**   1. The system will not allow the office manager to set the non-valued customer type a discount as the option will not appear as they are not a ‘valued’ customer. 2. The customers details will not be updated with a discount plan as the system rejected it and the customer details will remain the same. | |
| **Test:**   1. Click the ‘View’ button to visit Boris Bridge, once on the Manage Customer page, Boris Bridge’s account is set as ‘Normal’ and their account is labelled as ‘No’ under the sub-category of ‘Valued Customer’. 2. When viewing Boris Bridge’s account, check whether there you can select a ‘discount plan’ or whether the ‘set discount plan’ button is even there. The discount plan option should not appear for this non-valued customer. | |
| **Test record:** All features work correctly for this use case, and Boris Bridge does not have the access to a discount plan, as he is not set as a valued customer. | |
| **Date:** 17 April 2018 | **Tester:** Hamzah Malik |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC15 | **Use case name:** DowngradeCustomerAccount |
| **Test Number: 7** | |
| **Objective:** Test the main flow to see if the system allows user accounts to be downgraded from valued to normal. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Customer’ page to view a list of all the current customers. Account must be already set up with name ‘Anthony Gibson’ and account set to ‘Valued Customer’ | |
| **Expected results:**   1. Office Manager will downgrade customer from ‘Valued’ to ‘Normal’. 2. The privileges of the valued customer’s account are revoked, and they can no longer get discounts for purchases from BAPERS. 3. The database will update and show ‘Anthony Gibson’s’ account as ‘normal’. | |
| **Test:**   1. User navigates to ‘Manage Customer’ page and clicks ‘View’ on ‘Anthony Gibson’s’ account. 2. User has the option to edit, delete, view customers jobs or view their discount plan. Click the ‘Edit’ button. 3. User can change the customer account from ‘Valued Customer’ to ‘Normal customer’. | |
| **Test record:** All parts worked, and user account status is set to ‘Normal’, their ‘Valued Customer’ column is set to ‘No’ and they cannot be given a discount plan. | |
| **Date:** 16th April 2018 | **Tester:** Anthony Gibson |
| **Result:** Passed. | |

|  |  |
| --- | --- |
| **Use case ID:** UC16 | **Use case name:** UpgradeCustomerAccount |
| **Test Number: 8** | |
| **Objective:** Test the main flow to see if the system allows user accounts to be upgraded from normal to valued. | |
| **Set up:**  Logged in as the default Office Manager account ‘Jastor Gallywix’ and go to the ‘Manage Customer’ page to view a list of all the current customers. Account must be already set up with name ‘Anthony Gibson’ and account set to ‘Valued Customer’ | |
| **Expected results:**   1. Office Manager will upgrade customer from ‘Normal’ to ‘Valued’ 2. The privileges of the valued customer’s account are accessed by the newly set Valued Customer and the functionality of a discount plan can be set to their account. 3. The database will update and show ‘Anthony Gibson’s’ account as ‘valued’ | |
| **Test:**   1. User navigates to ‘Manage Customer’ page and clicks ‘View’ on ‘Anthony Gibson’s’ account. 2. User has the option to edit, delete, view customers jobs or view their discount plan. Click the ‘Edit’ button. 3. User can change the customer account from ‘Normal customer’ to a ‘Valued Customer’. 4. Check the customer account again to see whether a discount plan can be applied to Anthony’s account. | |
| **Test record:** All parts worked, and user account status is set to ‘Valued’, their ‘Valued Customer’ column is set to ‘Yes’ and they can view or be offered a discount plan. | |
| **Date:** 16th April 2018 | **Tester:** Anthony Gibson |
| **Result:** Passed. | |

*Unfortunately, some of our test cases did not get implemented due to member contribution and therefore will be left as “testing not required” within the relevant fields, as we cannot test them. However, as they are considered important to the system, we decided to include them within this report anyway, to highlight their functionality and what these test cases would do theoretically.*

|  |  |
| --- | --- |
| **Use case ID:** UC41 | **Use case name:** AlertOfficeManager |
| **Test Number: 9** | |
| **Objective:**  Testing the main flow to see whether Alerts will be generated for unpaid and late payments, after a set amount of time, which will alert the Office Manager. | |
| **Set up:** Logged in as the default Office Manager account ‘Jastor Gallywix’. A valid customer account must also exist to which late payments are detected from. Account ID of customer account set to 12 and the customer name as ‘Anthony Gibson’. | |
| **Expected results:**   1. The office manager will be alerted every 15 minutes after logging into the system that there is a late payment. 2. The office manager will identify that the late payment is from a valued customer. 3. The system will begin generating reminders letters based on how long it takes for the customer to clear their outstanding balance. | |
| **Test:** Testing Not Required | |
| **Test record:** This test did not manage to get implemented so unfortunately, cannot be tested. In theory, this test should lead to an alert to appear on screen, exclusively to the Office Manager, alerting them of unpaid payments that need to be paid. Once acknowledged, the system itself would have generated a reminder for the relevant customer account in question. | |
| **Date: -** | **Tester: -** |
| **Result:** Testing Not Required | |

|  |  |
| --- | --- |
| **Use case ID:** UC21 | **Use case name:** Login |
| **Test Number:** 10 | |
| **Objective:** Test the main flow to see if the login works for a user with a valid BAPERS account. | |
| **Set up:** BAPERS must be operational and a prompt to enter the user’s login details must appear.  Autofill setting are optional and could save users time when entering their log in credentials. | |
| **Expected results:**   1. The valid user will have their login accepted and they will have access to specific subsystems. 2. An office manager login will have access to all subsystems within BAPERS 3. Shift manager login will have access to all subsystems of BAPERS except for BAP-PAYM, BAP-CUST and BAP-ADMN 4. Receptionist login will have access to BAP-ACCT subsystem 5. Technician will have access to BAP-PROC subsystem | |
| **Test:** Login to the BAPERS system entering Bapers1 and Password 123 and pressing login. Make sure that after the details are entered correctly the system lets you into it. | |
| **Test record:** Unfortunately, much like Alerts, user authorisation was something we could not complete in time, due to member contribution. However, its importance is undeniable in our system for security reasons and in theory this test should give the user a prompt to enter their account username and their password. On success, the user at hand will have access to the relevant subsystems of BAPERS based on their role. | |
| **Date:** - | **Tester:** - |
| **Result:** Testing Not Required- | |

*As the test case ‘Login’ could not be managed, it also means that its alternative flows also were not implemented within the system. It must also be noted that although login has not been implemented, the base user in BAPERS is using a Office Manager account, hence why I have mentioned the office manager in previous set ups for test cases.*

|  |  |
| --- | --- |
| **Use case ID:** UC21.1 | **Use case name:** UnregisteredUser **(Alt Flow)** |
| **Test Number:** 10 | |
| **Objective:** Test the alternative flow to see if the system rejects a user trying to login with an Unregistered account. | |
| **Set up:**  User must have attempted to previously log into BAPERS. BAPERS must be operational. | |
| **Expected results:**   1. The system will not sign the user in and not let them access the BAPERS system. 2. An error message will display saying “Account not recognised, please try again or create a new account”. 3. The user will have to re-enter login details of a valid account. | |
| **Test:** Testing not required | |
| **Test record:** As mentioned above, since login functionality was not working in time, our alternative flows cannot function either. However, if it was to be working, this alt flow would occur from when the user attempts to login to an account that is not recognised. The system will throw up a response stating to either re-enter their login details or to register a user account.  Both would have labelled buttons to indicate these separate functions. | |
| **Date: -** | **Tester: -** |
| **Result:** Testing not required | |

|  |  |
| --- | --- |
| **Use case ID:** UC21.2 | **Use case name:** IncorrectLoginDetails (Alt Flow) |
| **Test Number:** 10 | |
| **Objective:** Test the alternative flow to check if incorrect login details for a registered user account are entered, whether the system will reject them and display the correct error message. | |
| **Set up:**  BAPERS must be operational and the user must have attempted to previously log into BAPERS for this alternative flow to occur. | |
| **Expected results:**   1. The system will not sign the user in and not let them access the BAPERS system. 2. An error message will display to the user saying, “Invalid login details, please try again”. 3. The system will detect whether the username is correct but not the password when displaying this message. | |
| **Test:** Testing not required- | |
| **Test record:** If we had implemented this feature, much like UnregisteredUser, the system would throw an alert stating that the user account details are invalid. However, unlike UnregisteredUser, the system will detect that the user exists if they enter the correct username but invalid password. In this scenario, the system would throw a different alert stating, ‘Invalid password detected, please retry’. | |
| **Date: -** | **Tester:** - |
| **Result:** Testing not required | |

# Non-Functional requirements using Volere template

# 4.2 Security testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement ID: 1** | **Requirement Type:** NFR (Security) | | **Event/Use Case #N/A** | |
| **Description:** The BAPERS system **should** only allow an authorised desktop user to have access to data relevant to a customer account. | | | | |
| **Rationale:** If the data collected by BAPERS is accessible to anyone other than authorised staff members, the security of the system will be at risk. This could lead to damaging effects to the company’s reputation and business.  Legal issues arise from this too as any third-party users that have access to private information of customers can lead to legal disputes. | | | | |
| **Source:** The initial statement of requirements and a subsequent interview with Mr. Lancaster, our client. | | | | |
| **Fit Criteria:** The way that user account security can be maximized during development is by avoiding the direct storage of passwords (or perhaps even usernames) in the database and instead storing the ‘representation’ of a password. Such a representation can be created through hashing the password using a one-way algorithm and throwing away the original password (so never storing it anywhere in the system). With this approach, in order to verify passwords, the value entered by a user can be hashed using the same algorithm and compared to the entry stored in the database.  As it stands, there is no feasible way to test for account security, other than hiring an ethical hacker and prompting them to attempt and acquire user account records from the BAPERS database. | | | | |
| **Customer Satisfaction: 5** | | **Customer Dissatisfaction: 4** | | |
| **Priority:** This is an essential requirement. | | **Conflicts:** None | | |
| **Supporting Material:** None | | | | Volere  Source: Atlantic Systems Guild |
| **History:** A new requirement | | | |

# Performance Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement ID:2** | **Requirement Type:** NFR (Performance) | | **Event/Use Case #** | |
| **Description:** The BAPERS system **must** consistently manage aresponse time to requests at under 500ms on our localhost:3000, ensuring that the system runs efficiently and quickly. Note that this version of BAPERS is just a prototype. | | | | |
| **Rationale:** If the response time to a request for BAPERS is not fast enough then that will have a large effect on user experience. As BAPERS is currently simply just prototype on a localhost, any future releases as a web-based project with a slow response time will be frustrating for users who will use BAPERS. | | | | |
| **Source:** The initial statement of requirements and a subsequent interview with Mr.Lancaster. | | | | |
| **Fit Criteria:** The use of Google Chrome’s “developer tools” is a particularly effective way to measure the response time of our BAPERS system on localhost:3000. Within the ‘Network’ subsection holds a tool that effectively records any network activity of the current website tab. When the user performs a request, or refreshes the page, the response time is measured in milliseconds and so efficiently measures how fast the system is responding to requests. Developer Tools are also available in most web browsers, albeit minor changes, the developer tools are still wildly the same amongst common web browsers. As our system’s implementation has had a large focus on web-based programming, the developer tools will serve as extremely useful when monitoring the response time of our system. | | | | |
| **Customer Satisfaction: 4** | | **Customer Dissatisfaction: 2** | | |
| **Priority:** A non-essential requirement. | | **Conflicts:** None | | |
| **Supporting Material:** None | | | | Volere  Source: Atlantic Systems Guild |
| **History:** This is a new requirement | | | |