KTS Solutions

Smart POS (Point of Sales) application
Master Test Plan

Version 1.0

Mentor - Dr. Sapumal Ahangama

Group Members

Kobinarth Panchalingam - 200307C

R. A. T. C. Kumara - 200321M

Sanjula Kumarasinghe - 200323V

KTS Solutions

Member	Contribution
200307C	• Evaluation Mission and Test Motivation • Target Test Items • Data and Database Integrity Testing • User Interface Testing • Load Testing • Deliverables • Risks, Dependencies, Assumptions, and Constraints • References
200321M	• Security and Access Control Testing • Failover and Recovery Testing • Configuration Testing • Test Evaluation Summaries • Deliverables • Risks, Dependencies, Assumptions, and Constraints • References
200323V	• Test Approach • Function Testing • Performance Profiling • Test Evaluation Summaries • Deliverables • Risks, Dependencies, Assumptions, and Constraints • References

Revision History

Date	Version	Description	Author
15/10/2023	1.0	initial version	KTS Solutions

Table of Contents

Master Test Plan

1. Evaluation Mission and Test Motivation

The project involves the development of a Smart POS (Point of Sales) application for supermarkets. This integrated solution comprises a web application along with a robust backend to streamline point-of-sale operations. The application targets a wide range of users, including cashiers, staff, managers, and customers, aiming to enhance operational efficiency and user experience.

In the current testing phase, the primary objective is to identify and rectify potential issues within the Smart POS application. The focus is on identifying significant problems that could impact the system's performance, as well as ensuring the application's security. The mission is to improve the application's functionality and safety for end-users. This testing effort is essential to validate the project's progress and overall quality.

The mission for the evaluation effort in this current iteration is to:

- Find and address as many bugs and issues as possible to enhance the reliability and functionality of the Smart POS application.
- Assess and mitigate perceived quality risks, with a focus on delivering a robust and dependable product.
- Advise on potential project risks, ensuring that the development process remains on track and any concerns are promptly addressed.
- Verify that the application aligns with the specified requirements and design, providing a clear understanding of its capabilities.
- Provide insights and recommendations on product quality to meet the expectations of stakeholders and end-users.
- Offer guidance on testing procedures and strategies to ensure thorough and effective evaluation.
- Fulfill any necessary process mandates for testing and quality assurance.

2. Target Test Items

User Interfaces:

Web Application Interface: The quality and usability of the web application interface should be thoroughly evaluated to ensure it provides an intuitive and responsive experience for users.

Backend Components:

API Endpoints: Validate the functionality, performance, and security of the backend API endpoints used for communication between the web and mobile applications.

Database: Test the database components for performance, data integrity, and data backup and recovery mechanisms.

Functional Components:

Sales Processing and Transaction Management: Verify that sales processing and transaction management functions are working correctly, adhering to specified requirements.

Inventory Management: Ensure that inventory management functions accurately track and update inventory items.

Employee Management: Test the features related to employee management, including task assignment and data integrity.

Customer Management: Validate customer management functions, including loyalty programs and customer data handling.

Report and Analytics: Check the reporting and analytics components to ensure they provide real-time insights and meet quality standards.

E-commerce Integration: Test the integration with e-commerce platforms, ensuring seamless operation and security.

Multi-Store Support: Verify that multi-store support features function correctly, allowing efficient management of multiple stores.

Performance and Security:

Response Time for a Transaction: Measure the response times for transactions to ensure they meet performance requirements.

Data Encryption: Assess data encryption mechanisms to protect sensitive information.

User Authentication and Authorization: Test the authentication and authorization mechanisms for security.

Resource Utilization: Monitor resource utilization, including CPU and memory usage, to ensure optimal performance.

Configuration:

Configurability: Ensure that the application functions correctly with different configurations, such as varying hardware setups and network settings.

Integration with Third-Party Services:

Payment Gateways: Verify the integration with third-party payment gateways for secure and efficient payment processing.

Barcode Scanners: Test integration with barcode scanning tools to ensure accurate product data handling.

3. Test Approach

Data and Database Integrity Testing:
Verify the accuracy of data interactions with the PostgreSQL database by executing validation queries, monitoring for anomalies, and using automated data checks. Assess data integrity and consistency.

Function Testing:
Conduct black-box testing to ensure the application handles data, enforces business rules, and produces expected results. Use unit testing to validate specific functions and their interactions.

User Interface Testing
Assess user interfaces for design, functionality, and usability, covering components, forms, navigation, responsiveness, and user experience across different devices and browsers.

Performance Profiling:
Evaluate system performance under various workloads through load testing, measure response times, and assess database performance. Ensure the system meets acceptable response time standards.

Load Testing:
Simulate transaction loads to observe system behavior and performance under normal, peak, and extreme conditions. Monitor the system's ability to handle various workloads without performance degradation.

Security and Access Control Testing:
Validate authorization, authentication, and security measures. Ensure users are restricted to specific

functions and only authorized individuals can access the system and applications.

Failover and Recovery Testing:
Test the system's ability to failover and recover from hardware, software, or network malfunctions without data loss. Create scenarios to invoke recovery processes and verify data integrity.

Configuration Testing:
Verify the application's operation on various software and hardware configurations, including different web browsers, platforms, and devices. Test the system in diverse environments to ensure compatibility.

3.1 Testing Techniques and Types

3.1.1 Data and Database Integrity Testing

The integrity of testing the database is the evaluation of processes, operations, and methods, used in accessing, managing, and updating the database, also data integrity testing is mainly focused on the accuracy and constancy of the stored data so the data gives the anticipated outcomes. Using certain procedures, database integrity is to evaluate how correctly and compatible the data is accessed, processed and updated, or manipulated, is the goal of conducting the data and database integrity. This testing will be able to examine whether the data is valid, corrupted, lost, modified, added, etc.

Technique Objective:	● The objective of Data and Database Integrity Testing is to rigorously evaluate the PostgreSQL database access methods and processes, independently of the user interface, specifically focusing on the data tables created using Spring Boot's ORM. The goal is to identify and log any incorrect functioning or data corruption.
Technique:	● Execute each PostgreSQL database access method and process that interacts with the data tables created through Spring Boot's ORM. Provide them with valid and invalid data or data requests. ● Scrutinize the PostgreSQL database, particularly focusing on the data tables created by Spring Boot, to ensure that data has been populated as intended, and all database events have occurred correctly. ● Review the returned data to confirm that the correct data was retrieved for the appropriate reasons..
Oracles:	● Validation Queries: Verify the results of predefined validation queries (e.g., SQL queries) executed before and after the test specifically on the Spring Boot-generated data tables to detect inconsistencies. ● Logs and Alerts: Continuously monitor system logs and alerts to identify anomalies, such as errors or exceptions, during the test, with a focus on the data tables. ● Data Checks: Employ automated data validation scripts tailored to the data tables created by Spring Boot's ORM to ascertain data integrity throughout the testing process, highlighting any discrepancies or corruption.
Required Tools:	● Test Script Automation Tool ● Base configuration imager and restorer ● Backup and recovery tools ● Installation-monitoring tools (registry, hard disk, CPU, memory, etc.) ● Database SQL utilities and tools ● Data-generation tools
Success Criteria:	● The technique must be able to validate all critical database access methods and processes that interact with data tables created using Spring Boot's ORM effectively.

Special Considerations:	● Testing may necessitate a DBMS development environment or drivers to directly input or modify data in the PostgreSQL database, particularly targeting the Spring Boot-generated data tables. ● Processes should be manually invoked. ● To enhance the visibility of any non-acceptable events, it's advisable to use small or minimally sized data tables with a limited number of records.

3.1.2 Function Testing

Function testing of the target application mainly checks if it meets specific requirements tied to use cases, business functions, and rules. These tests ensure that the app correctly handles data, follows rules, and produces the expected results. We use black box methods, meaning we test the app without looking at how it works internally. Instead, we interact with the Graphical User Interface (GUI) and check what comes out.

Technique Objective:	● We thoroughly explore every aspect of the system, such as navigating through it, entering data, processing tasks, and retrieving information, with the aim of observing and documenting its behavior.
Technique:	Performing the following checks for each use-case scenario, using both valid and invalid data: ● Confirm that expected results are achieved with valid data ● Ensure that the system displays the correct error or warning messages when invalid data is used. ● Validate that every business rule is correctly enforced. ● If there was an error the database consistency is maintained. ● Automate unit testing for each function for asserting the invocation of other expected functions with expected arguments. ● Automate the GraphQL Queries and Mutation using GraphQLTester
Oracles:	We will rely on JUnit as the primary tool for running automated tests. JUnit is well-suited for unit and integration testing within a Spring Boot application. It helps us quickly identify whether our code changes work as intended or introduce any problems.
Required Tools:	● JUnit5 ● Mocktio ● GraphQlTest ● GITHUB ● Postman
Success Criteria:	All scheduled tests have been carried out, and any discovered issues have been effectively dealt with. This approach is well-suited for examining: ● All vital use-case scenarios ● All fundamental features The successful execution of these use cases and features serves as a significant benchmark for success.

Special Considerations:	When we're testing the system with automated tools, how fast the system responds is crucial. This is especially true when the system needs to get data from real web services or databases. But there are a few problems with this: Speed Impact: Making the system talk to online databases can make the tests run slowly rather than using a local database. Covering Everything: It's really hard to test all the different ways things can go right or wrong when using real online databases because they can behave in unpredictable ways.

3.1.3 User Interface Testing

Testing User Interfaces ensure the better user experience by checking the navigation, functioning of components and the proper implementations. This test will evaluate how well system performs to satisfy end user needs

Technique Objective:	● Check the functionalities of components like buttons, dropdowns, checkboxes, forms etc. ● Check the random navigations and flow of functionalities ● Check the dynamic rendering and consistency of the content. ● Ensure the simplicity and user acceptance ● Ensure the responsiveness for different gadgets ● Web application: using different PCs, laptops, tablets and mobile phones
Technique:	● Evaluate the design and functionality of all components ● Check all the forms with valid and invalid test inputs ● Check the trigger actions with keyboard, mouse and by other pointing devices ● Check the navigation and consistency of User Interface ● Evaluate the proper display of contents in the component ● Evaluate the usability with different users
Oracles:	Human interaction is essential to test the usability and user acceptances of the system
Required Tools:	● Manual Testing ● Jest for React component testing ● React Testing Library to component testing ● Test Renderer to render React components to pure JavaScript objects.
Success Criteria:	● All the test cases for the functionality of the components passed the test. ● All the pages remain dynamic and consistent. ● System is consistent and having a proper navigation ● Accepted by the users in manual testing
Special Considerations:	● Random user interaction is needed to test the user acceptance ● All the possible input combinations must be tested and attaining the desired outcome is confirmed

3.1.4 Performance Profiling

Performance profiling for the POS system is like checking how quickly and efficiently the system operates. The main aim is to make sure it meets its speed and performance standards. This process involves testing the system under various conditions, like during peak usage or on different hardware setups, to ensure it operates smoothly and swiftly.

Technique Objective:	To make sure the Smart POS system meets the performance and the capacity criteria, evaluate its performance under various workloads and scenarios.
Technique:	● Load Testing: Simulate different transaction loads to observe and log the system's behavior and performance under varying loads. Test scenarios may include peak hours, typical usage, and off-peak periods. ● Response Time Testing: Measure the time taken for the system to respond to user actions and transactions, ensuring it meets acceptable response time standards. ● Database Performance Testing: Assess the database's performance under various scenarios, including data insertion, updates, and retrieval, to ensure it can handle transaction data efficiently.
Oracles:	The performance testing results should meet predefined criteria and benchmarks. These criteria can be based on acceptable response times, throughput, error rates, and system resource utilization
Required Tools:	● Performance Testing Tools: Apache JMeter is a tool used to simulate user interactions and evaluate system performance under various loads. ● Database Profiling Tools: Utilize database profiling tools to measure the performance of database operations and optimize SQL queries.
Success Criteria:	● The Smart POS system should handle anticipated workloads without degrading performance. ● Response times should meet or exceed predefined standards. ● The system should successfully handle multiple concurrent users and transactions without critical failures. ● Memory and CPU usage should remain within acceptable limits.
Special Considerations:	The unique factors that must be taken into account when performance testing your Smart POS system. These factors are necessary for a complete analysis of the system's performance and dependability. ● An accurate workload Replicating real-world situations is crucial when performing performance testing. This includes imitating various consumer actions, transactions, and usage patterns. Think about the typical load throughout the day and in various situations. Use should be made of realistic data inputs, such as product specifications, costs, and customer data. ● Testing for Scalability: Testing for scalability evaluates the system's performance as it develops. Reviewing the system's capacity to handle a larger number of transactions, clients, and locations is part of this process. The system must be able to expand and handle more load without suffering performance degradation. Planning for capacity benefits from scalability testing.

3.1.5 Load Testing

Varying workload will be send to evaluate the performance and the ability of the system to handle those workloads to work properly. Test the ability of the system to work beyond the maximum limit maximum load. Performance characteristics and response times were also evaluated

Technique Objective:	Exercise designated transactions or business cases under varying workload conditions to observe and log target behavior and the system performance.
Technique:	● Send a large quantity of request to the server ant then evaluate the availability of the system for new requests ● Modify the test cases to increase the number of retrievals and increase their frequency ● Evaluate the system performance in normal and extreme conditions ● Evaluate the system performance under sustained loads and instantaneous load scenarios·
Oracles:	● Results are obtained from the testing tools and assumed to be reliable ● Success or failure will be evaluated by comparing the requirement mentioned in the initial documentation with the final results from the testing tools
Required Tools:	● Apache JMeter for testing the web application ● Artillery for backend testing
Success Criteria:	● System passed the test under peak conditions and in normal conditions. ● In all the scenarios, the performance of the system in the accepted range
Special Considerations:	● Database should be scaled nearly to the size of actual database with mock data· ● Should be carried out on a dedicated machine so that we can control and accurate the measurements

3.1.6 Security and Access Control Testing

Security and Access Control Testing for a Smart Point of Sale(POS) Application focuses on two key areas of security. Application-level Security(Authorization), including Data or Business Functions access, and System-level Security.

Application-level security

Application-level security ensures that actors are restricted to specific functions or use cases, and they are limited in data availability.

System-level security

System-level security ensures that username-password authentication and remote access to the system is controlled through secure API endpoints.

Technique Objective:	● Authorization :A Particular user can access only the functions and data that are provided to that particular user type. ● Authentication/ System-level Security: Only those actors with access to the system and applications are permitted to access them. This will guarantee that Application Level can only perform only certain tasks and an intruder who broke into the application level cannot get to the database level without the system access level.

Technique:	● Security at the application level identify and list each user type, as well as the functions or data that each type has access to. ● Create tests for each user type and verify each permission by establishing all of the user type-specific functionalities. ● Change the user type and re-run the tests with the same people. Verify that the new functions or data are correctly available or restricted in each case. ● System-level Access creates a user account with System-Level Access enabled. Then access and change the database using database queries. Test accessing them from a user-level access account.
Oracles:	Access Control and Security Testing needs to be done manually because it cannot be completely automated. A great tool for identifying vulnerabilities in websites is OWASP ZAP. The system is always vulnerable to attacks because there are vulnerabilities that remain unfixed, yet OWASP ZAP testing will be sufficient for this system.
Required Tools:	● For security access control testing, use the OWASP ZAP tool. Additionally, it provides automated voluntary detection. ● Use the Chrome developer tool to review the status and request headers. ● Use the Google Lighthouse tool to check safety and trust.
Success Criteria:	● Show invalid credentials message when a user tries to login with wrong credentials. ● Successful login message if the user login with correct credentials. ● Identify the user type when login and provide only allowed functionalities for the user type.
Special Considerations:	The specific access levels are assigned to the job recruiter and the job seeker accounts, so the user is granted the specific privileges when they create the account for themselves.

3.1.7 Failover and Recovery Testing

Failover and recovery testing ensure that the target-of-test can successfully failover and recover from a variety of hardware, software, or network malfunctions with undue loss of data or data integrity.

For those systems that must be kept running, failover testing ensures that, when a failover condition occurs, the alternate or backup systems properly “take over” for the failed system without any loss of data or transactions.

Recovery testing is an antagonistic test process in which the application or system is exposed to extreme conditions, or simulated conditions, to cause a failure, such as device Input/Output (I/O) failures, or invalid database pointers and keys. Recovery processes are invoked, and the application or system is monitored and inspected to verify proper application or system, and data recovery has been achieved.

This Smart POS system is mainly based on database transactions. Hence it should be possible to make sure that the system remains in a consistent state even if some failure happens. Here consistent state implies that the system should maintain data integrity.

Technique Objective:	System will be tested in following conditions. ● Power interruption in server ● Power interruption of the client ● Incomplete Cycles ● Communication interruptions via network servers

Technique:	● Power interruption in the server can be simulated by running a local server and stopping it. ● Power interruption of the client can be simulated by either turning off the PC or mobile. ● Communication interruptions can be simulated by putting airplane mode in the devices
Oracles:	Using the above-mentioned simulations, we can test the system again using unit testing.
Required Tools:	● External power supplies such as Uninterruptible Power Supply (UPS) devices. ● External hard disks for backing up data ● Installation monitory tools such as Task Manager
Success Criteria:	One or more simulated recoveries involving one or more combinations of the application, database, and system to a known desired state
Special Considerations:	Specifications of the devices needed to be considered.

3.1.8 Configuration Testing

Configuration testing verifies the operation of the target-of-test on different software and hardware configurations. In most production environments, the particular hardware specifications for the client workstations, network connections, and database servers vary. Client workstations may have different software loaded_for example, applications, drivers, and so on_and, at any one time, many different combinations may be active using different resources.

Doing configuration testing is very important because Smart POS System will be operated in many devices and many platforms

Technique Objective:	Web application is to be tested in various web browsers on different platforms or devices.
Technique:	● Use Function test scripts ● Open and close various non-target-of-test related software either as part of the test or prior to the start of the test ● Execute selected transactions to simulate actors interacting with the target-of-test and the non-target-of test software.
Oracles:	By testing all functionalities tested by unit tests on each environment, we can improve the outcome of the configuration testing.
Required Tools:	Web Browsers such as, ● Google Chrome ● Firefox ● Microso
Success Criteria:	● The technique supports the testing of one or more combinations of the target test items running in expected, supported deployment environments ● Web application: Testing the web application on both PCs and mobile devices with different web browsers.
Special Considerations:	Hardware specifications of the testing environment

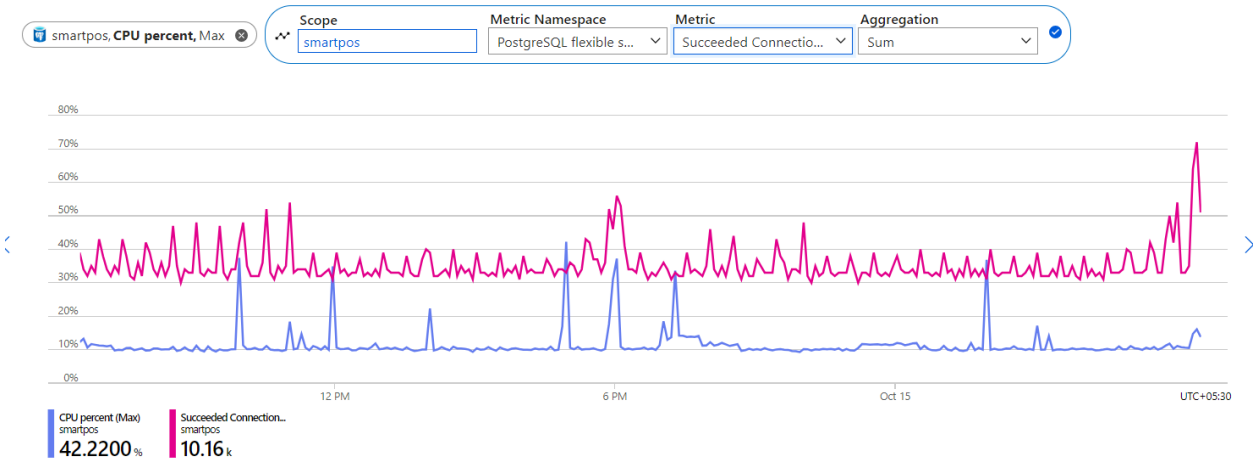
4. Deliverables

Automated test cases report with the test inputs, expected outputs and their results will be provided for the future usage and for various stakeholders

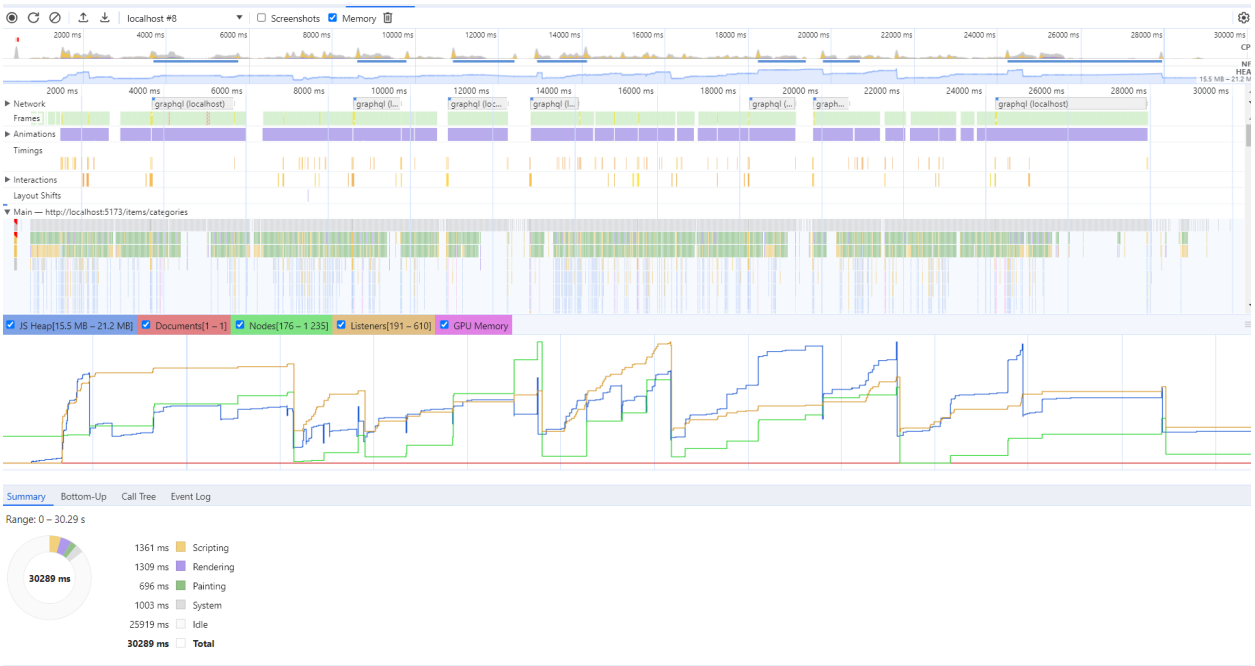
4.1 Test Evaluation Summaries

These are the detailed logs that give the outcome of every test that is written for the functionalities. Suppose a single test log is given the output as a failure, then that functionality is recorrected and all the tests are run again and the test logs are checked for faults. After a new functionality is added or a change to an existing functionality is made, test logs will be provided very often soon

1. Postgresql database performance



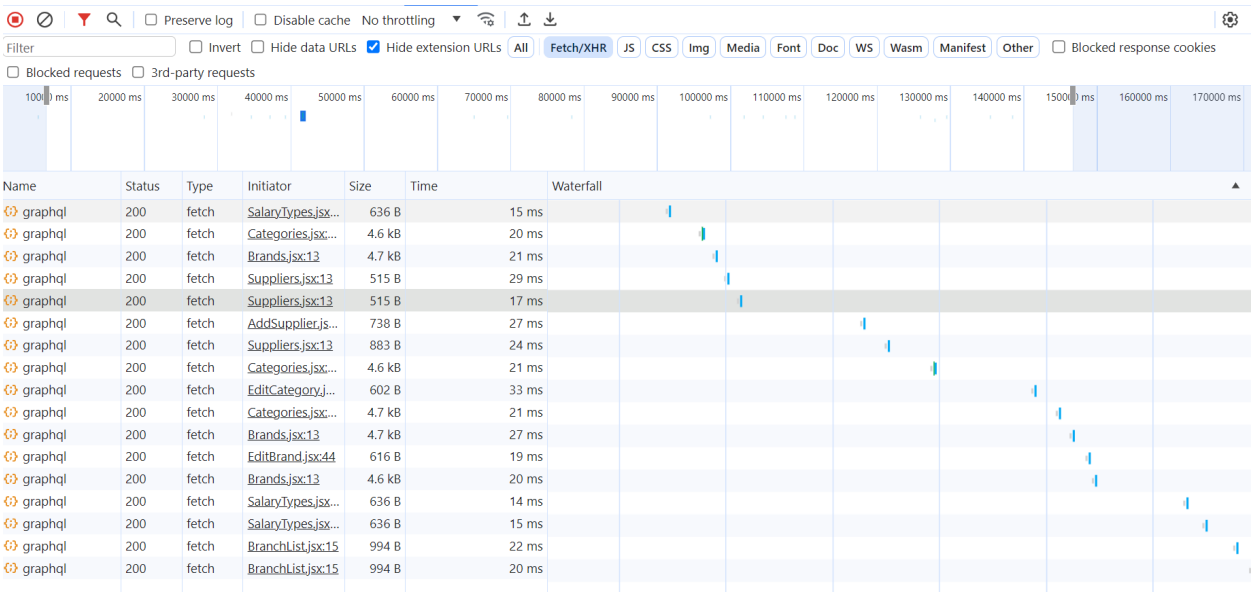
2. Browser Performance Details



3. PostgreSQL database benchmark

```
Password:
starting vacuum...end.
transaction type: <builtin: simple update>
scaling factor: 70
query mode: simple
number of clients: 20
number of threads: 4
duration: 60 s
number of transactions actually processed: 85752
latency average = 13.996 ms
tps = 1428.940490 (including connections establishing)
tps = 1431.652976 (excluding connections establishing)
```

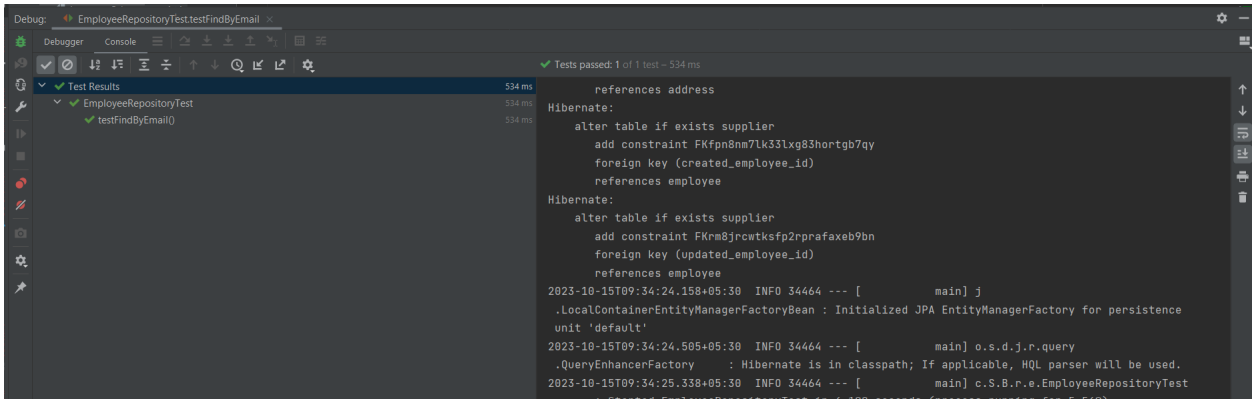
4. Data fetching response times



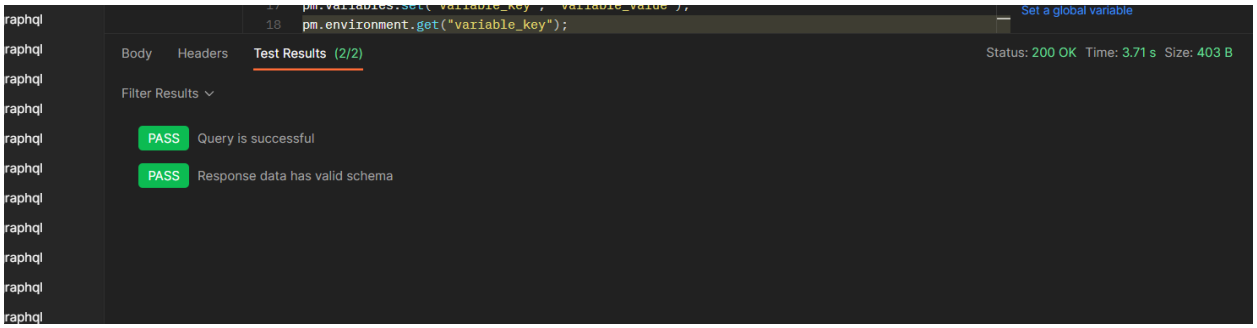
5. Frontend UI testing

```
PS E:\Project\pos-web-app> npm test

> pos-web-app@0.0.0 test
> jest

 PASS  src/state/tests/sidebar.test.js
 PASS  src/state/tests/salaryTypes.test.js
 PASS  src/state/tests/customer.test.js
 PASS  src/state/tests/employee.test.js
-------------------|---------|----------|---------|---------|-------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-------------------
All files          |   96.72 |        0 |   95.23 |   96.72 |
 src               |      60 |        0 |       0 |      60 |
  ApolloClient.js  |      60 |        0 |       0 |      60 | 10-12
 src/graphql       |     100 |      100 |     100 |     100 |
  customers.js     |     100 |      100 |     100 |     100 |
  employees.js     |     100 |      100 |     100 |     100 |
 src/state/reducers |     100 |      100 |     100 |     100 |
  customer.js      |     100 |      100 |     100 |     100 |
  employee.js      |     100 |      100 |     100 |     100 |
  salaryTypes.js   |     100 |      100 |     100 |     100 |
  sideBar.js       |     100 |      100 |     100 |     100 |
-------------------|---------|----------|---------|---------|-------------------

Test Suites: 4 passed, 4 total
Tests:       15 passed, 15 total
Snapshots:   0 total
Time:        2.319 s
```

6. Functional unit testing



7. Postman GraphQL api testing



4.2 Reporting on Test Coverage

Testing will happen through the construction and transition phases of the RUP(Rational Unified Process) model. Testing will happen multiple times throughout the process. Testing result will consist of graphs and screenshots. Functional testing will happen after every major function development. Code will be improved according to the test results. Performance testing and the load testing will be happen after deploying the server code to the cloud environment

For each testing endeavor, we prepare comprehensive reports that include the following details:

1. Date
2. User who logged in for testing
3. Test case field
4. Number of test cases executed
5. Number of test cases passed
6. Number of test cases failed
7. Pass percentage
8. Fail percentage
9. Additional comments

These reports are meticulously generated for both successful and unsuccessful test cases, with special emphasis placed on testing cases related to use cases and non-functional requirements, particularly security aspects.

5. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Prerequisite entry criteria is not met.	Developers will define the prerequisites that must be met before Load Testing can start. Users will endeavor to meet prerequisites indicated by Developers.	● Meet outstanding prerequisites ● Consider Load Test Failure
Test data proves to be inadequate.	Users will ensure a full set of suitable and protected test data is available. Developer will indicate what is required and will verify the suitability of test data.	● Redefine test data ● Review Test Plan and modify ● components (that is, scripts) ● Consider Load Test Failure
Database requires refresh.	Admin or the Developer will endeavour to ensure the Database is regularly refreshed as required by Users .	● Restore data and restart ● Clear Database
Data is corrupted	Data is corrupted System Admin backs up the database regularly in order to prevent data losses Restore the latest backup to the database server	● Data is corrupted ● System Admin backs up the database ● regularly in order to prevent data losses ● Restore the latest backup to the ● database server

6. References

[1] “Jest documentation” Internet: Getting Started · Jest (jestjs.io) [Oct 15, 2023]
[2] “React test renderer” Internet: react-test-renderer - npm (npmjs.com) [Oct 15,2023]
[3] *Pgbench* (2023b) *PostgreSQL Documentation*. Available at: https://www.postgresql.org/docs/current/pgbench.html (Accessed: 15 October 2023).
[4] *The 5th major version of the programmer-friendly testing framework for Java and the JVM* (no date) *JUnit 5*. Available at: https://junit.org/junit5/ (Accessed: 15 October 2023).
[5] *Mockito Framework Site* (no date) *Mockito framework site*. Available at: https://site.mockito.org/ (Accessed: 15 October 2023).
[6] *Postman api platform* (no date) *Postman API Platform*. Available at: https://www.postman.com/ (Accessed: 15 October 2023).