**CS673 Software Engineering**

**Team Rocket - Project RCM: Rental Car Management**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alisa Belousova | Configuration Lead | *Alisa Belousova* | 9/9/2023 |
| Nick Cruz | Team Leader/ Requirements lead | *Nickolas Cruz* | 9/9/2023 |
| Devon Duddley | QA Lead and Design | *Devon Dudley* | 9/9/2023 |
| Kris | Security Lead | *Cangqing (Kris) Wang* | 9/9/2023 |
| Chenghao Ye | Design and Implementation | *Chenghao Ye* | 9/10/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Devon Dudley** | **9/25/2023** |  |
| **2** | **Nickolas Cruz** | **9/25/2023** | Added tests to a spreadsheet and linked them under Manual Testing Report. Added testing frameworks being used. |
| **3** | [Alisa Belousova](mailto:alisa007@bu.edu) | **10/08/2023** | Write testing summary, manual testing report, automated testing report, testing metrics, glossary sections |

[Testing Summary](#_sm5odwyvuk3j)

[Manual Testing Report](#_pqso2mbjyzx4)

[Automated Testing Report](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

Within our project, a rigorous testing strategy has been implemented to ensure the efficiency and reliability of the system. Employing a dual-method approach, our testing encompasses both automated and manual methods to validate functionality, usability, and performance.

# Manual Testing Report

Manual testing has played a pivotal role in ascertaining RCM robustness and efficacy. Incorporating thorough manual testing at multiple stages, developers are first required to validate their own code manually before introducing a Pull Request (PR). This developer-level testing involves checking new or altered code, ensuring functionality is as expected and that no regressions or unintended changes are introduced. This step forms a vital initial safeguard against bugs and facilitates a more stable development environment.

Also our testing team systematically performs a manual end of sprint check of the entire application. This includes both pre-defined test case execution and exploratory testing to assess the holistic impact of the sprint’s changes on the application. The specific tests executed, along with their detailed results, can be accessed through the following link: <https://docs.google.com/spreadsheets/d/1yO57h078uDUAO35GZ5VdClSbiMiaO6ZqQ5Mcq5Q4avg/edit#gid=0>.

Through these combined manual testing strategies, from initial development to sprint closure, we ensure all code, whether new or modified, undergoes rigorous validation before being deployed or progressed to subsequent development stages.

Here is a sample template that can be used for each test case. For system tests or acceptance tests, you may also include some screenshots.

* Test case ID, name
* New or old:
* Test items: (what do you test )
* Test priority (high/medium/low)
* Dependencies (to other test case/requirement if any):
* Preconditions: (if any)
* input data:
* Test steps:
* Postconditions:
* Expected output:
* Actual output:
* Pass or Fail:
* Bug id/link: (this should link to your github issue id)
* Additional notes:

(You can use an additional spreadsheet for this section as well)

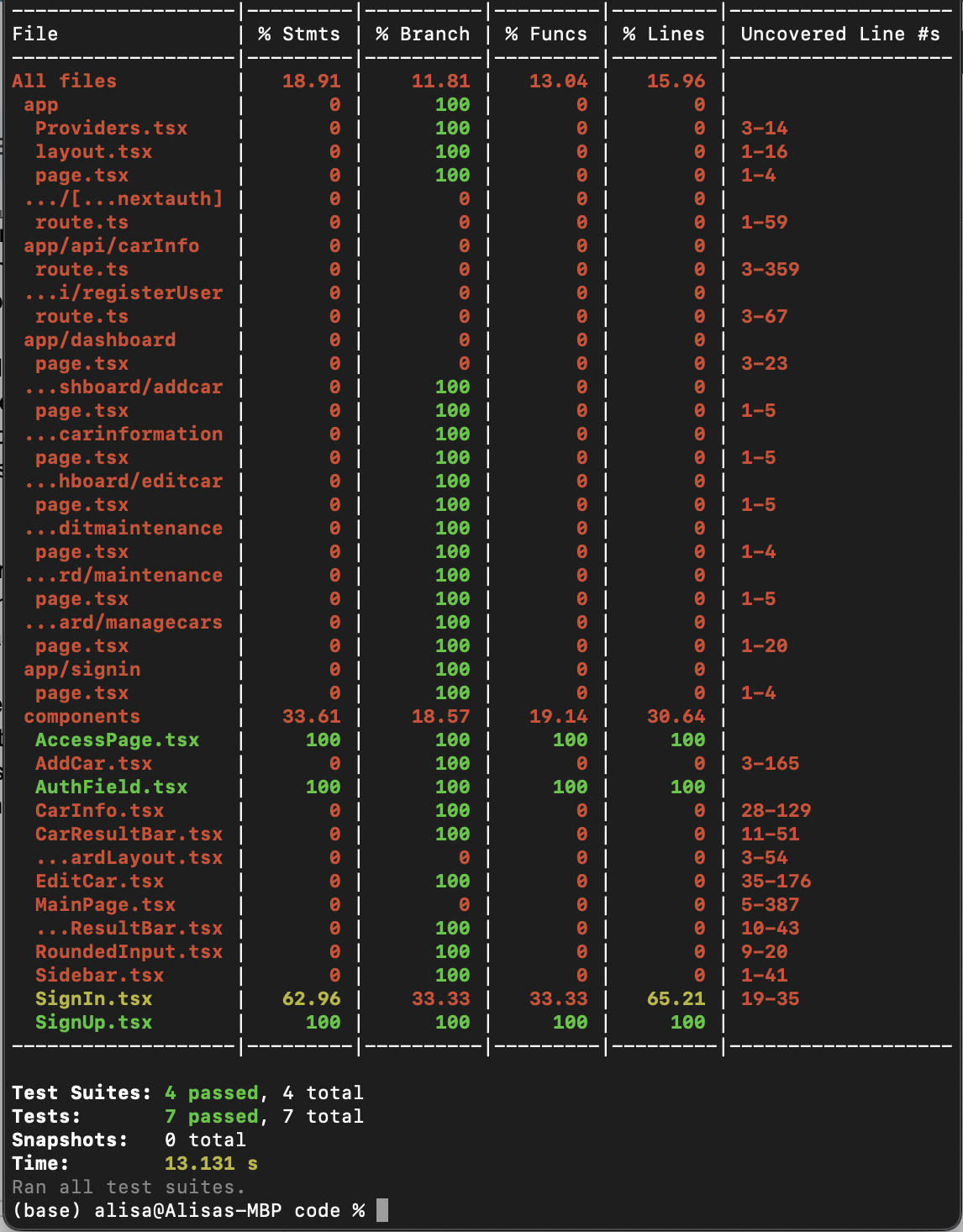
# Automated Testing Report

Automated testing is instrumental in facilitating a robust, resilient, and regression-proof software environment. To ensure consistent and reliable code integration, all automated tests are run through GitHub Actions during our CI process. Detailed reports of the test executions, along with screenshots and logs where applicable, are stored as artifacts within GitHub Actions, providing transparent and accessible insights into each testing phase and allowing for easy debugging when required.

The CI/CD pipeline is structured to execute Jest unit tests and Playwright end-to-end tests in the following scenarios:

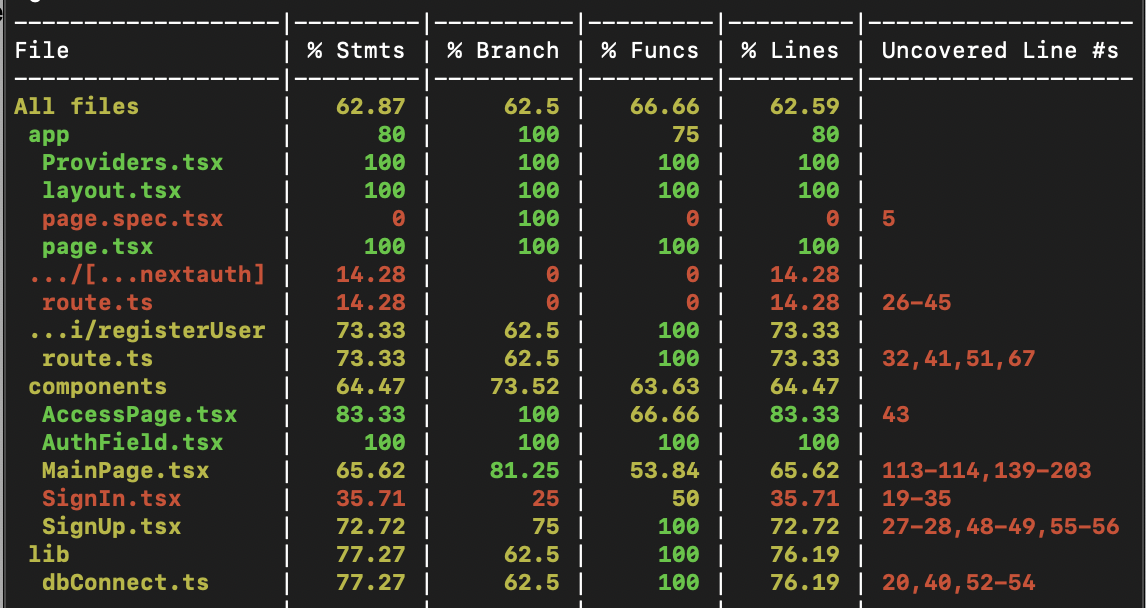
* For Every Pull Request (PR): Whenever a PR is created, GitHub Actions automatically triggers the running of the entire suite of automated tests. This step ensures that all proposed changes, even before being merged, adhere to the expected functionality and do not introduce regressions.
* After Merge to Master: Subsequent to any merge into the master branch, GitHub Actions initiates another round of automated tests to validate that the integrated code continues to uphold the established quality and functionality benchmarks. This safeguard ensures that the master branch, representative of our production code, is consistently stable and reliable.

Utilizing Jest for unit testing, we have structured tests that validate both pure and asynchronous functions, ensuring the logic at the smallest units of code adheres to expected outcomes. The test files for Jest have spec.ts(x) extension and are located next to the tested files.



Unit test coverage (10/09/2023)

In terms of end-to-end testing, Playwright has been instrumental in mimicking user behaviors and journeys across different browser environments to validate integrations and holistic system functionality. The Playwright test scripts are housed in the /e2e directory. Screenshots are generated and stored in the /e2e/snapshots directory, providing a transparent and tangible record of our automated testing outcomes. Please refer to the repository for a deeper dive into the specific test cases and their respective outcomes.



E2e test coverage (10/09/2023)

# Testing Metrics

Metrics have been leveraged to quantitatively evaluate the effectiveness, completeness, and efficiency of the testing processes. Some of the pivotal metrics incorporated in our testing life cycle are:

* Number of Test Cases: The total count of both manual and automated test cases created and executed throughout the project development.
* Test Coverage: Assessed using Jest’s built-in coverage tool, this metric shows the percentage of code exercised by the automated unit tests, and is crucial in identifying untested parts of the codebase.

# References

# Glossary

1. RCM: Rental Car Management
2. Repo: Repository
3. PR: Pull Request
4. IDE: Integrated Development Environment
5. CI: Continuous Integration
6. CD: Continuous Deployment