
Nischhal Shrestha

# Enhancing Readability, Accessibility, and Shareability of Robot Framework Test Results

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

1 October 2023

## Abstract


| | |
|---|---|
| Author: | Nischhal Shrestha |
| Title: | Enhancing Readability, Accessibility, and Shareability of Robot Framework Test Results |
| Number of Pages: | 45 pages + 5 appendices |
| Date: | 1 October 2023 |
| Degree: | Bachelor of Engineering |
| Degree Program: | Information Technology |
| Professional Major: | Mobile Solutions |
| Supervisors: | Joeffrey Chan, Technical Lead<br>Daniel Girmay, Team Manager<br>Amir Dirin, Senior Lecturer |


---


A Web Report Improvements feature was developed to enhance clarity and accessibility of Robot Framework (RF) test results. This feature was built using the React library in the frontend while Node.js was utilized to build RESTful APIs in the backend. Recharts, a widely used library in React was used for constructing charts. The dashboard page was transformed into a PDF document through the utilization of PDFKit, a Node.js library designed for generating PDF documents.

Central to this implementation is a sophisticated dashboard page for presenting the Robot Framework (RF) test results with enhanced clarity and accessibility. Additionally, this feature also includes Portable Document Format (PDF) export functionality, which will transform test result details into a PDF document, and a dropdown menu through which users can navigate between different test results effortlessly. The dashboard provides intuitive user experience with functionalities including result tabulation, an image display for test artifacts, graphical representations of complex data for users in a simpler manner, enabling comparison between different data points, and a dedicated segment for user equipment (UE) analysis results.

The Web Report Improvement feature provides user-friendly design and enhances user experience by providing a compact view, clear data visualization and streamlined navigation for test case results. Furthermore, the PDF export functionality elevates the shareability of the test case results, enhancing collaboration between the stakeholders.



Keywords: Dashboard, React, Node.js, Robot Framework


---

# Contents

"In order to maintain the confidentiality of Nokia, both the graphical user interface (GUI) and the source code have been modified for the purposes of this thesis. Specific details of these modifications have been intentionally omitted."

Appendices

## List of Abbreviations

| | |
|---|---|
| API: | Application Programming Interface |
| DOM: | Document Object Model. Object model for Hypertext Markup Language (HTML) which defines HTML elements properties, events, and methods. |
| GUI: | Graphical User Interface. Visual part of a web application or software that users interact with. |
| HTML: | Hypertext Markup Language |
| TAT: | Test Automation Tool. Nokia's internal website dedicated to the creation, execution, and viewing of corresponding test results for Robot Framework tests. |
| PDF: | Portable Document Format |
| REST: | Representational State Transfer |
| RF: | Robot Framework |
| UE: | User Equipment (mobile). Mobiles devices capable of running modern applications and accessing the internet, typically known as smartphones. |
| UI: | User Interface. Specific aspect of a web application or software that users interact with. |
| UX: | User Experience. Experience and satisfaction a user has when interacting with a product, system, or service. |

# 1 Introduction

Test Automation Tool (TAT) is a multi-platform, distributable, scalable, and service-based solution based on Robot Framework. TAT provides users with a web-based Graphical User Interface (GUI) for the creation and execution of Robot Framework tests, eliminating the necessity for prior programming skills. Following the test execution, corresponding test results are generated, which are accessible through the designated test result detail page. However, users have encountered issues with navigation as they were earlier required to navigate between multiple tabs to view a single test result in detail. In addition, navigating between different test results was also difficult and time-consuming as users had to go back and forth between web pages to access multiple test results. Furthermore, the lack of PDF exporting functionality added to the complexity of sharing test results, hindering collaboration and decision-making processes.

In response to this challenge, a dashboard page, PDF exporting functionality and a dropdown menu were introduced in the final year project described in this thesis as a part of the Web Report Improvements feature for TAT, which is the focal point of this thesis. The dashboard page was designed with the aim of enhancing the readability and accessibility of test result details by consolidating test outcomes within a unified user interface (UI).

The dashboard page contains several sections that are discussed below:

- Test case summary section: This section displays a summary of test case results in a compact view.

- Test case analysis section: This section includes results of test case analyses which are not related to UE.

- Screenshot section: This is a section that contains a list of screenshots generated during test execution.

- UE view: This view is dedicated to displaying UE analysis results in detail. The UE view integrates graphs using the React Rechart library to display analysis data visually, which enhances the readability of the test results.

Through the implementation of a dropdown menu, the navigation process between different test results is streamlined, reducing the time and effort required for users to locate desired test result details. Additionally, the integration of PDF exporting functionality facilitates the seamless sharing of test outcomes, thereby enhancing collaboration and easing the process of data dissemination.

This thesis will delve into the technical aspects of the dashboard page, the dropdown menu, and the implementation of the PDF exporting functionality as a part of the Web Report Improvements feature. In addition, the methodologies employed for evaluation and the reflections on the achievements and challenges encountered throughout the development process will be discussed.

# 2 Need for Web Report Improvements Feature

UX Design is the process of designing physical or digital products that are useful, easy to use, and provide a great experience in interacting with them (Canziba, 2018, p.8). The current test results and test result detail page lack satisfactory user experience and user-friendly design. The test result does not provide users a navigation system to navigate between different test results. Similarly, the result detail page does not provide a compact view of the test results.

Figure 1 illustrates the current design of the test results detail page. As observed in the figure, this page lacks sufficient options for the users. The lack of an option for users to view test results in a detailed compact view is notable. Users must navigate to different tabs, such as the Graphs tab for viewing graphs or the Logs tab for accessing test result logs.

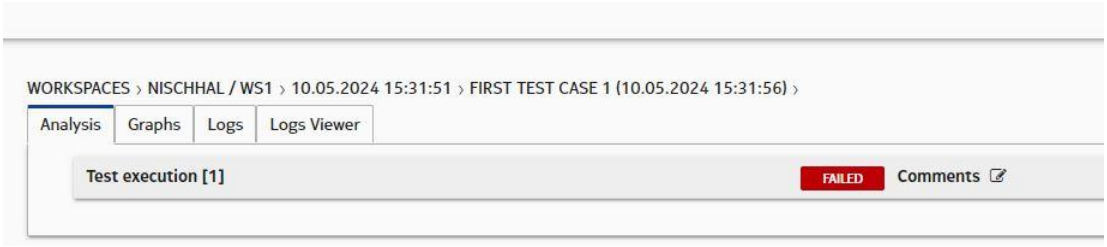


Figure 1. Illustration of the test result detail page display of a test case detail

Figure 2 illustrates the test result page, facilitating users' navigation to the desired test result detail page. As already observed in Figure 1, the lack of a navigating option between multiple test results on this page necessitates users to return to the test result page, to select the desired test result from the list, and to subsequently navigate to the respective test result detail page.

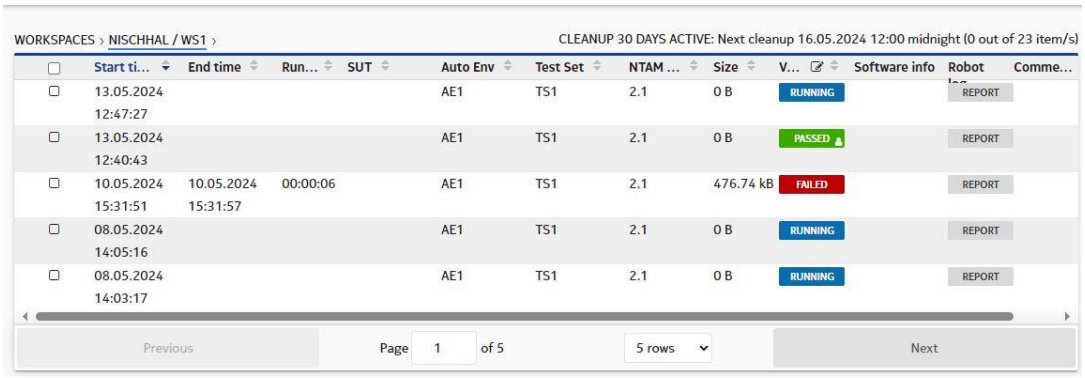


Figure 2. Illustration of the test result page displaying a list of test cases

As observed in Figure 3, the lack of a PDF file-sharing option on the current TAT web page poses a significant impediment to collaborating and sharing information among the stakeholders. This limitation restricts users' access and sharing test results in a universally accessible format (PDF), thus reducing the possibility of making informed decisions, which drastically hampers the project outcomes.

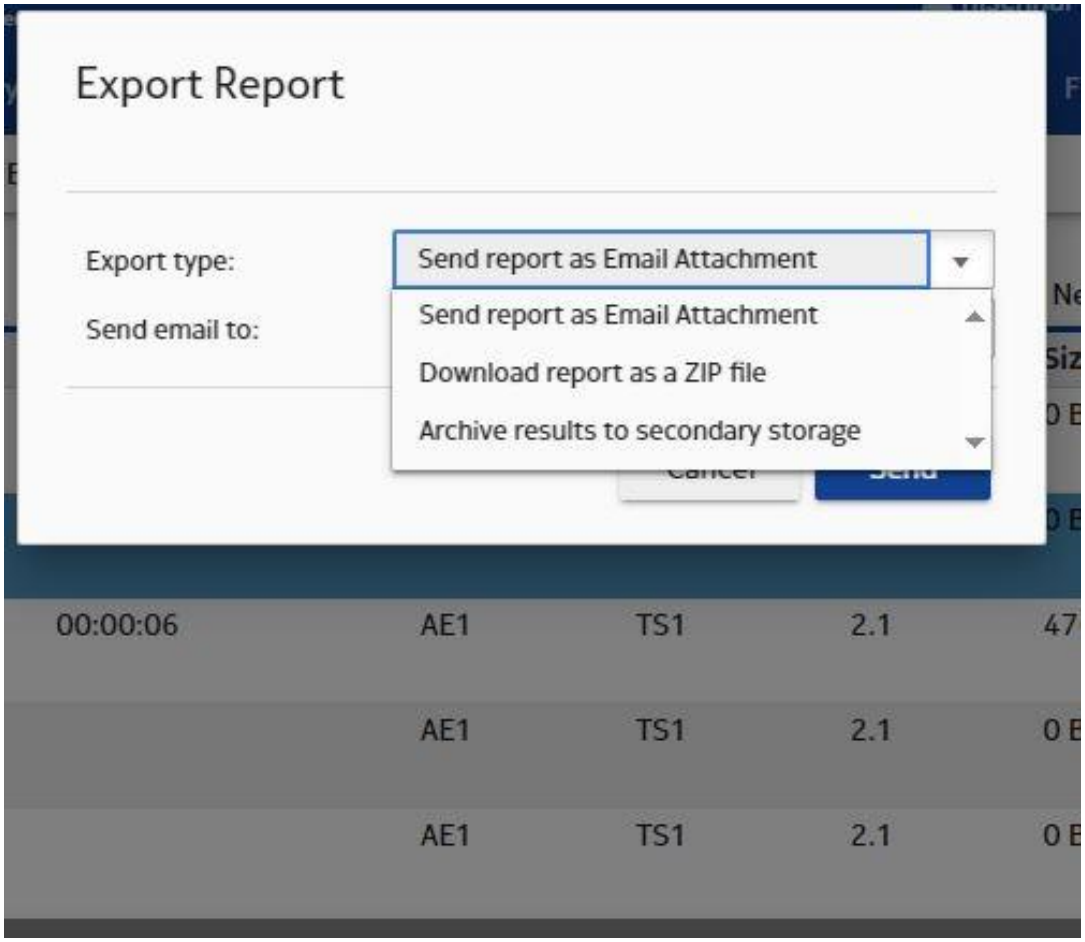


Figure 3. Illustration of report exporting options on the test result page

To address all the issues outlined above, a team at Nokia has considered the implementation of the Web Report Improvements Feature which includes a dashboard page, a dropdown menu and PDF exporting functionality. Colleagues involved in this project have dedicated their time and expertise in designing the UI for the dashboard page, the dropdown menu and the PDF export to implement the desired feature within the TAT page. This implementation aims to introduce a user-centric solution that enhances users' experience by introducing seamless navigation options between different test results, a single page that displays test result details and an option for sharing test results effortlessly in a universal, accessible format between stakeholders.

As Nokia's team finalized the feature and evaluated the time needed to complete it, the current task involves implementing the feature within the TAT page. Using the coding skills, communication abilities, and teamwork developed during academic studies, I am committed to completing this feature within the estimated time frame.

# 3 Background Study

## 3.1 Current Used Technologies

Upon conducting research on the overall TAT project, it was discovered that this feature should be implemented using the React library, which will display the RF test results to users, and Node.js in the backend, which acts as a bridge between the front end and the database.

### 3.1.1 React

React is a library for helping developers build user interfaces (UIs) as a tree of small pieces called components. A component is a mixture of HTML and JavaScript that captures all the logic required to display a small section of a larger UI. Each component can be built up into successively complex parts of an application (Baer, 2018, p.1). Each React component is independent and encapsulates its style, structure, and behaviours. The reusability of React components facilitates easier code management and maintenance. Figure 4 illustrates the component-based architecture of ReactJS, highlighting the interactions and communication between components and the server-side.

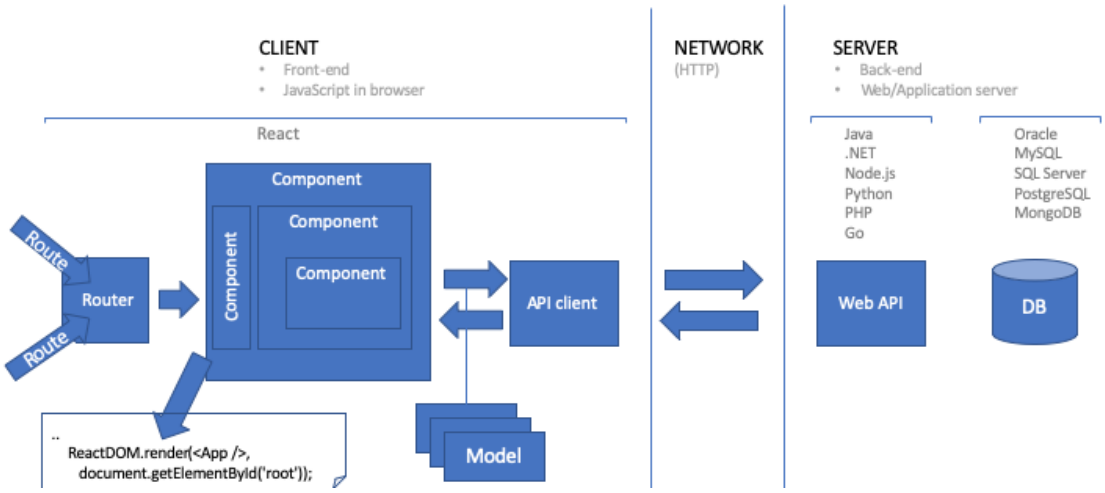


Figure 4. Illustration of ReactJS – Architecture (Hands-On React, 2024)

There are numerous libraries such as Angular, Vue, and Vanilla for developing the UI. The primary reasons for using React in the TAT project are mentioned below.

The Document Object Model is an object model for the Hypertext Markup Language (HTML) which defines the properties, events, and methods of the HTML elements. The Virtual DOM is a lightweight version of a web page's actual DOM, serving as a virtual duplicate. React uses this Virtual DOM to minimize direct changes to the real DOM and to manage the user interface's state. When a component's state changes, React generates a new Virtual DOM and then uses a process called "diffing" to compare this new version with the existing one. This comparison helps React determine the fewest changes needed to update the real DOM. The use of the Virtual DOM is a key factor in React's ability to scale effectively, render efficiently, and optimize performance (Dragusin, 2023). JavaScript itself is a secure language. React, based on the JavaScript library, inherits the security features provided by the JavaScript language. React also uses a one-way data flow, which makes it easier to manage and control data changes within an application making a more secure development process. In addition to these security features, React also supports the use of JavaScript XML (JSX) which is a syntax extension that helps to mitigate the risk of Cross-

Site Scripting (XSS) attacks by escaping user inputs by default (Budventure Technology).

React's component-based architecture is the key to its success. The React application is built by combining different independent components each handling a specific part of the UI. These components can be reused in different parts of the application, thus increasing the development efficiency and maintainability of the application (Budventure Technology). In React, data is passed from the parent component to a child component using properties (props). Figure 5 illustrates the rendering of the React child component 'Headline' within the main component 'App'.

```
import React from 'react';

const App = () => {
  const greeting = 'Hello Function Component!';

  return <Headline value={greeting} />;
};

const Headline = ({ value }) => {
  return <h1>{value}</h1>;
};

export default App;
```

Figure 5. Illustration of an arrow function component in React (Wieruch, 2019)

React applications are reliable due to their utilization of virtual DOM and one-way data flow. Furthermore, React is an open-source library created by Facebook having large and active community support, extensive documentation, and a rich ecosystem support for development, testing, and deployment of applications. As the React library is actively maintained by Facebook, any issues found in this library are quickly fixed hence making it more reliable to use. State is a mutable data source that can be used to store data in a React application and can change

over time and be used to determine how a component renders (Afonso & Mestre, 2023, p.1). React state can change over time. When there is change in state, the React component will re-render to display the changes in UI. React provides a clear and efficient mechanism for state management by centralizing the state in a common parent component. This method removes redundant or duplicate states, hence, simplifying statement management and reducing bugs related to state inconsistency.

### 3.1.2 Robot Framework

Robot Framework (RF) is an open-source automation framework designed for test automation and robotic process automation (RPA). Supported by the Robot Framework Foundation, it is widely used in the industry due to its versatile and human-friendly syntax. The framework uses keywords and supports extensibility through libraries written in Python, Java, and other languages. Additionally, Robot Framework integrates seamlessly with other tools, providing comprehensive automation solutions without incurring licensing fees. The framework is further strengthened by a vibrant community that contributes hundreds of third-party libraries, enhancing its functionality and reach (Robot Framework, 2024).

RF uses a readable, keyword-driven syntax that is easy to understand and write. This makes this framework easy to use by both technical and non-technical users. RF supports a wide range of test libraries written in Python and Java, and it can be extended with custom libraries as well, which makes RF easy to integrate with other tools and systems. RF runs on several platform, including windows. macOS and Linux. It is flexible in terms of execution and deployment environments. RF provides detailed reporting and logging features on test execution, which helps users to easily identify and diagnose issues quickly. RF uses a keyword-driven method. Keywords can be reused in multiple test cases which promotes reusability and maintainability of test scripts. Due to support from a large active community and the RF foundation, RF benefits from continuous improvement,

contributions, and a wide range of third-party libraries. Figure 6 illustrates RF test cases which validate the login credentials of the user for a certain web page.

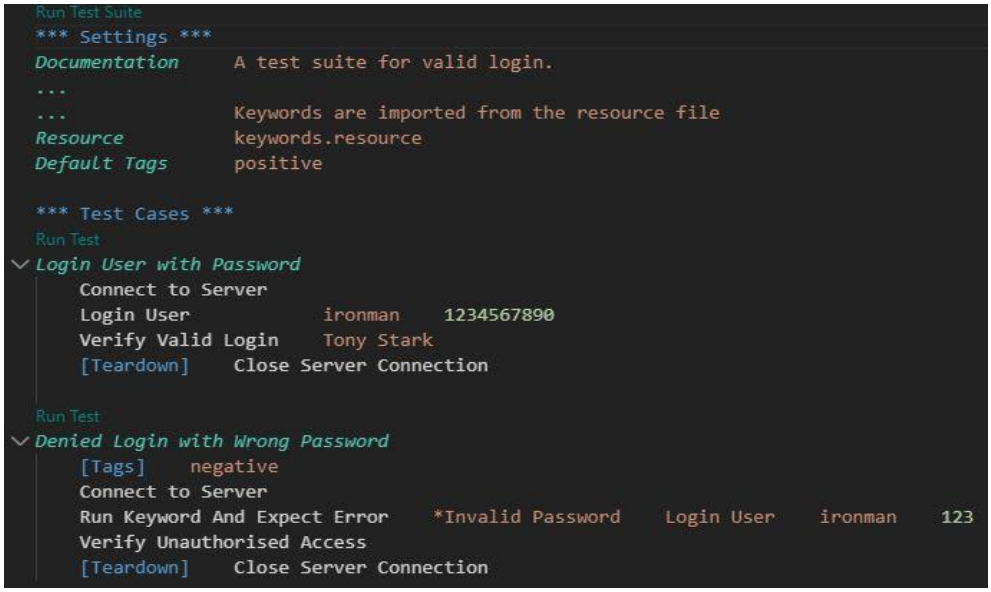


Figure 6. Illustration of the Robot Framework test case (Robot Framework, 2024)

### 3.1.3 Node.js

Node.js is an open-source JavaScript runtime environment that executes JavaScript code which is built on the V8 JavaScript engine (Mead, 2018, p.1). It is cross-platform and can run on Windows, Linux, macOS and more. This cross-platform compatibility makes Node.js a very versatile choice for developers. Even though it is single-threaded, its event-driven architecture allows it to handle multiple concurrent operations efficiently. Node.js non-blocking I/O operations enhance its efficiency in handling high-performing applications. It comes with a powerful node package manager (NPM) that hosts thousands of libraries and tools. This package makes it easy to manage projects, and dependencies and share code with other developers. Furthermore, Node.js is asynchronous and non-blocking meaning any heavy performing task such as reading files, querying a database, or making HTTP requests, does not block the execution of the code. Instead, the tasks run in the background using call-backs and promises, or async/

await to handle the asynchronous operations, which improves performance and scalability.

As observed in Figure 7, the ‘asyncOperation’ function simulates an asynchronous operation performing a heavy task that takes two seconds to complete. The code in line 24 is executed before the code in line 21. This is evident from the output log. This behaviour occurs because the ‘asyncOperation’ function runs in the background which allows code afterwards to execute. Finally, when the ‘asyncOperation’ function completes the task, the result is printed in the output log.

```
1  /**
2  * Simulates an asynchronous operation that takes 2 seconds to complete.
3  *
4  */
5  const asyncOperation = () => {
6    return new Promise((resolve) => {
7      // Use setTimeout to simulate a time-consuming
8      // operation (e.g., network request, database query, etc.)
9      setTimeout(() => {
10       // Resolve the promise with a completion message after 2 seconds
11       resolve('Async operation has completed.');
12     }, 2000);
13   });
14 };
15
16 const asyncCall = async () => {
17   console.log('Starting async operation.');
18
19   const promise = asyncOperation();
20   promise.then((result) => {
21     console.log(result);
22   });
23
24   console.log('This code runs before the async operation.');
25 };
26
27 asyncCall();
28
```

Run ›

Reset

```
> "Starting async operation."
> "This code runs before the async operation."
> "Async operation has completed."
```

Figure 7. Illustration of asynchronous operation using Node.js

### 3.1.4 Portable Document Format (PDF)

A PDF (Portable Document Format) is a versatile and widely used file format for documents that preserve their original appearance and formatting across different devices and platforms. Unlike other file formats, a PDF is designed to be platform-independent, meaning it can be viewed, shared, and printed on various operating systems without the need for the original software or fonts used to

create the document (Adobe, 2024). PDFs are easy to share by email or cloud storage, which simplifies the process of sharing test results between team members.

## 3.2 Alternative Technologies

There are also alternative technologies available besides the ones mentioned in the previous section which can achieve equivalent results. These alternative technologies are discussed below.

### 3.2.1 Angular

Angular is a widely adopted framework used for developing not only web applications but also mobile and desktop applications. This framework, backed by Google, is used by Google and millions of other applications (Ayaz & Obaid, 2021, Preface). It is a comprehensive framework that can serve as an alternative to React for building web applications. Angular is built on TypeScript which provides a robust framework for building scalable applications. It is a component-based framework that contains a collection of well-integrated libraries to cover a wide variety of features. This framework also includes a suite of developer tools to develop, build, test and update code (Angular, 2024). Figure 8 illustrates the Angular code for rendering two components: ‘Paragraph’ and ‘AppComponent’. The UI displays corresponding data according to the specified styles for each component.

```
import { Component } from '@angular/core';
@Component({
  selector: 'Paragraph',
  template: `
  <p><ng-content></ng-content></p>
  `,
  styles: ['p { border: 1px solid #c0c0c0; padding: 10px }']
})
export class Paragraph {
}
@Component({
  selector: 'app-root',
  template: `
  <p>
  <Paragraph>Lorem ipsum dolor sit amet, consectetur adipiscing elit. </Paragraph>
  <Paragraph>Praesent eget ornare neque, vel consectetur eros. </Paragraph>
  </p>
  `,
  styles: ['p { border: 1px solid black }']
})
export class AppComponent {
  title = 'welcome to app!';
}
```

Figure 8. Illustration of Angular Component (Clow, 2018, p.8)

### 3.2.2 Vue

Vue.js, often referred to as Vue, is built on top of the JavaScript framework that provides a well-structured mechanism to build web applications. Vue follows the 'Model-View-ViewModel' pattern where 'ViewModel' is the binder that binds data between the 'View' and 'Model'. It is the fastest and most lightweight framework that offers less time-consuming downloading and better run-time performance from a browser perspective. Vue serves as another alternative framework to React for building web applications (Shavin, 2023, p.1). Figure 9 illustrates the Vue code for rendering the 'Button' component within the main component.

```
<script setup>
import ButtonCounter from './ButtonCounter.vue'
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

Figure 9. Illustration of Vue Component (Vue.js)

### 3.2.3 Laravel

Laravel is a powerful and flexible PHP framework designed to streamline the development of robust web applications. It stands out for flexibility, which simplifies complex tasks and enhances productivity. According to Laravel's creator Taylor Otwell, Laravel has simplicity in design, requiring minimal configuration, a powerful infrastructure suited for modern PHP development and strong community support (Sinha, 2019, p.1). Laravel can serve as an alternative to Node.js for backend development. Figure 10 illustrates a basic routing mechanism in Laravel which returns the string 'Hello World' when the routing path '/greeting' is accessed.

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Figure 10. Illustration Basic Routing Mechanism with Laravel (Laravel, 2024)

# 4 Web Report Improvements Feature Design

## 4.1 Design Enhancements and Customer Integration

This feature design was created by the Nokia team. During the design process, they incorporated several critical enchantments to improve the user experience and meet specific customer needs. This feature improvement plan and priorities were revised by conducting proof-of-concept evaluations and addressing both current and new customer needs.

The primary focus of this feature design was the improvement of the readability and accessibility of test case results in TAT. The Web Report Improvements feature was designed to be flexible to display new test data as part of a broader feature without affecting the existing layout. Additionally, this feature was designed to maintain consistency with current components and styles to preserve the cohesive look and feel across the application. The feature design capabilities were demonstrated to customers, and customer trials were facilitated through meetings and email discussions. These methods helped to refine the feature design based on customer feedback. During the design process, allocation of resources and scheduling were considered to ensure successful implementation and timely delivery of the product.

## 4.2 Feature Designs

Figure 11 illustrates the overall structure of the dashboard page. On the top right side of this page, there is a drop-down menu through which users can navigate between different test results. Below the drop-down menu, right in the middle of the page, there is a summary section which clearly displays the overall result of a test execution. The test execution section displays the overall verdict of the test execution. If a test fails during the execution, the list of failed keywords will be displayed right below this section. Below the test execution section, there is an analysis section which displays the summary of a test case analysis. This section

displays the list of passed and failed analysis keywords which are not related to UEs. Also, this section is optional as it will be visible when a test case contains non-UE analysis results.

The screen-shots section displays the list of images horizontally. The test case may or may not generate image during test execution. If the test case contains images, then this section will be visible in the UI, if not then this section will not be displayed in UI.

The next section is the ue-view section, which is one of the most important functionalities of the dashboard page. In this section UEs are aligned horizontally in a reversed order whereas the analysis results generated by the UEs are displayed vertically. The analysis results are mapped and placed right under their respective UEs that have generated the data. Analysis keywords are situated on the left side of the analysis results which were used to generate the analysis results during test execution. The analysis keywords and results are aligned horizontally. This section will only be visible if a test case contains UE analysis data.

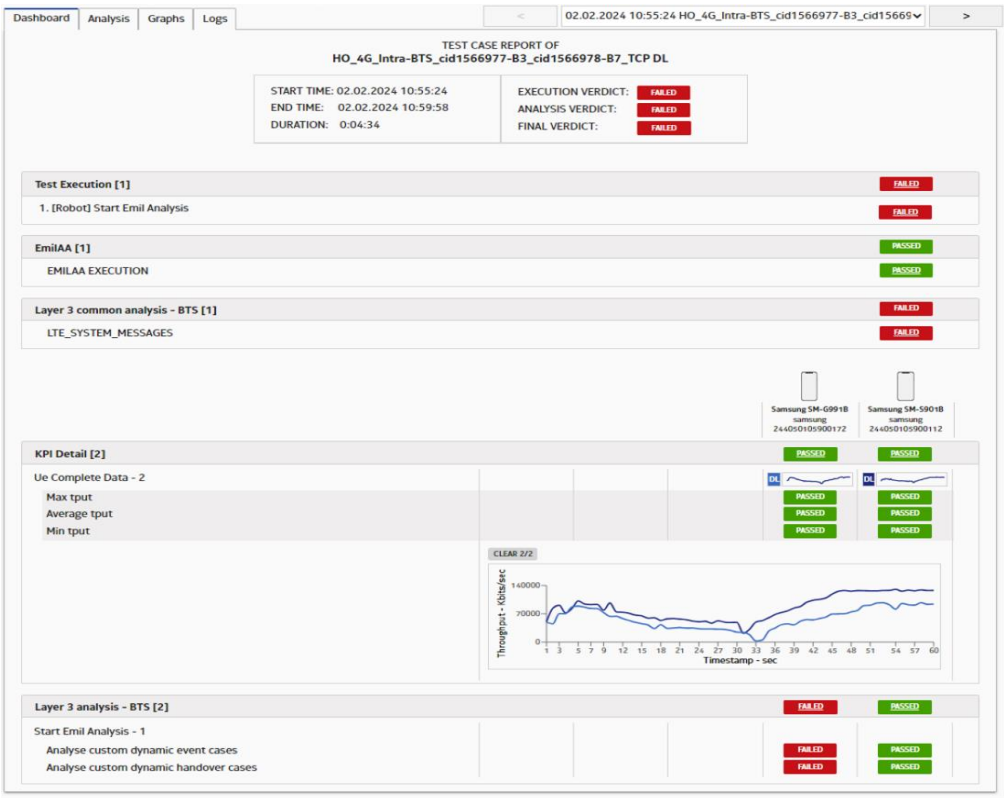


Figure 11. Illustration of Dashboard Page Design

### 4.2.1 Dropdown Menu

The primary purpose of the dropdown menu is to make navigation between different test results effortless. Additionally, the dropdown menu aims to eliminate the need of toggling between the test result page and the test case detail page to access different test results. Hence, the dropdown menu will save the user's time and effort and enhance the user UI navigation experience. The dropdown menu contains three major components, left and right navigation buttons on the sides and a central option that displays a list of the test case results.

As observed in Figure 12, there are two methods available for navigating between test results. The first method is to select a test case directly from the list and to display the result in the dashboard page. The second option is to navigate by using the previous and next buttons located on left and right side of the option list.

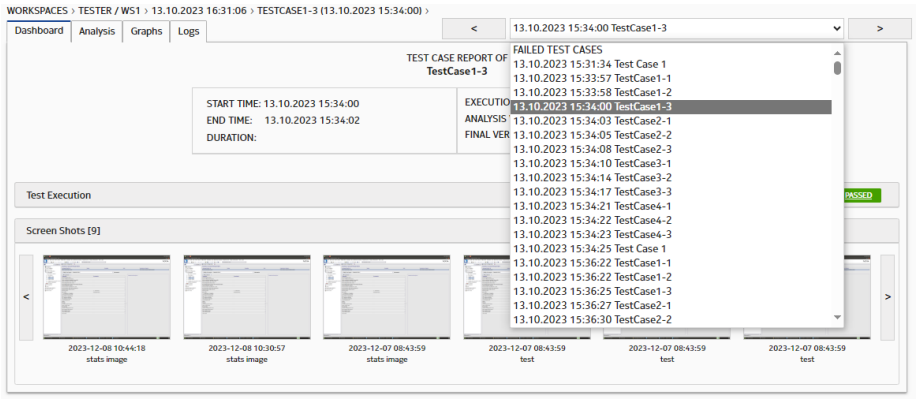


Figure 12. Illustration of the Drop-down Menu

Furthermore, the dropdown menu also includes an option ‘FAILED TEST CASES’. Upon selecting this option, the option list will exclusively display a list of failed test cases. This option aids users in efficiently navigating between failed test cases specifically instead of scrolling through the entire test case list. By implementing this option, users can focus on failed test cases only and examine the robot keywords or logs associated with the test case and identify the reasons for the failures.

### 4.2.2 Test Case Summary Section

As observed in Figure 13, the test case summary section displays a summary of test case results in a compact view. This section displays details such as the start time, end time, and duration of the test case execution as well as a summary of test case analysis verdicts.

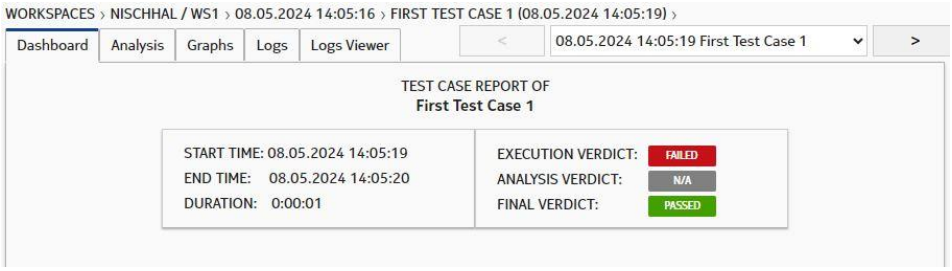


Figure 13. Illustration of the Test Case Result Summary Section

### 4.2.3 Test Case Analysis Section

As observed in Figure 14, the test case analysis section includes test case analysis results that are not related to UE. In this section, it is mandatory to display test execution analysis as it will always be present in test execution. In contrast, other analyses are optional. For optional analyses, both passed and failed RF keywords are displayed as a list, while for the test execution analysis, only failed RF keywords are shown. By clicking the result button on the right-hand side of the analysis result list, users can navigate directly to that keyword within the RF HTML log file. Users have the option to display or hide the list of keywords depending on their needs by clicking the test analysis section, thus enhancing the user-friendliness of the UI.

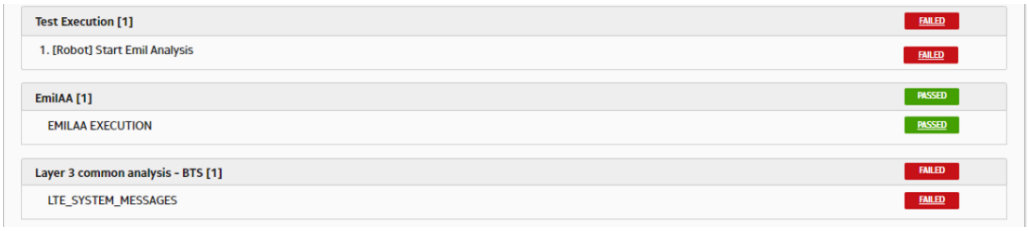


Figure 14. Illustration of Test Case Analysis

### 4.2.4 Screen Shots Section

As observed in Figure 15, the screen shots section displays a list of images generated during the test execution in a horizontal view. The maximum number

of visible images is adjusted according to the user's computer screen size. If the total number of images exceeds the limit, only the maximum number of visible images will be displayed, and swipe buttons will appear on the left and right sides of the view. By clicking the swipe buttons, users can navigate between the images and access non-visible images. Furthermore, swipe buttons will be disabled if there are no image to swipe on each respective side.

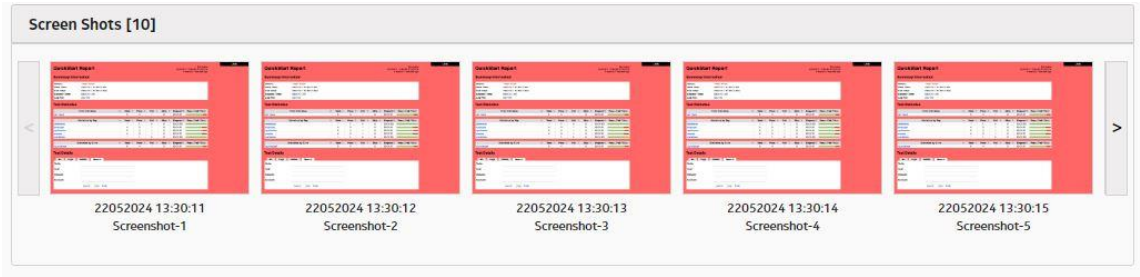


Figure 15. Illustration of the Horizontal List of Test Result Images

Figure 16 illustrates the zoomed image. Since the images in a list are not visible, users have the option to view a zoomed version of each image. When users click the desired image, a window will pop up displaying the zoomed version of that image. This window will also include a left and right swipe button on the sides of the zoomed image to navigate between the images, and a close button on the top right to close the pop-up window when clicked.

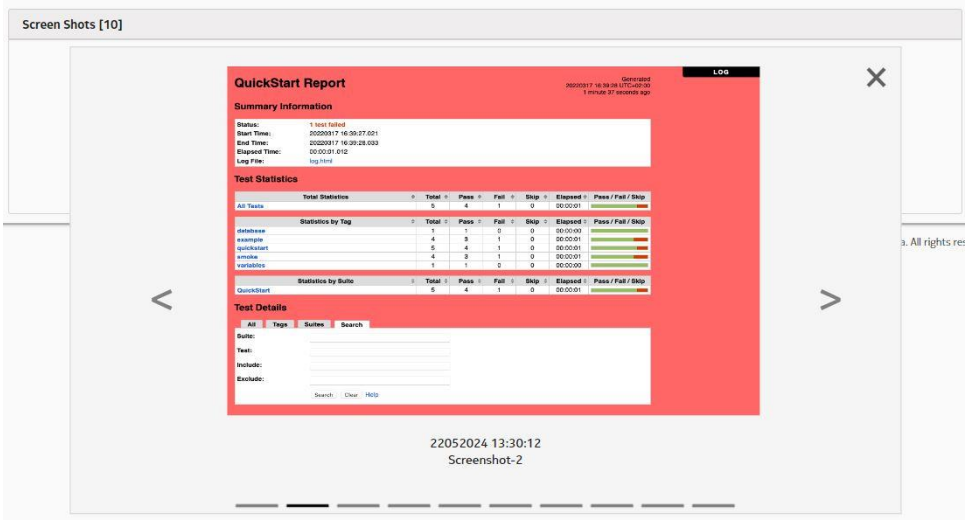


Figure 16. Illustration of Zoomed Test Result Image

### 4.2.5 UE View Section

As observed in Figure 17, a horizontal list of UEs, used during test execution, is positioned at the top of this section. The corresponding UE analysis results are aligned vertical with the UEs. Below the UE analysis results, the UE graph thumbnails and event results are also aligned vertically with the UEs. On the left-hand side, analysis keywords are aligned horizontally with their respective graph thumbnails, while event names are aligned horizontally with their corresponding even results.

UEs may or may not generate all combined analysis data during test execution. For instance, only the first UE ‘Samsung SM-S901B’ generated ‘Ue Complete Data-1’ analysis data, which is displayed in the UE view with a corresponding graph thumbnail and analysis data. The rest of the UEs did not generate any data related to the ‘Ue Complete Data-1’ analysis, resulting in empty fields in the UE view analysis. Additionally, only the fourth UE ‘OnePlus IV2201’ generated ‘Ue Complete Http-1’ analysis data. Since no graph was associated with this analysis data, the graph thumbnail is absent in the UE view for this fourth UE. Users can select a graph arbitrarily for the UE view through free selection, or by clicking on the event name section for horizontal graphs selection. When users select a

graph by clicking on the event name section, the background colour of that section changes to grey indicating graphs associated with that section are visible in the chart. These observations can be seen in Figure 17.


Figure 17. Illustration of the UI of the UE View Displaying Graphs data of Selected UEs

### 4.2.6 PDF Export

This PDF export options functionality was added to the TAT test result page. This functionality allows users to export test case results in a PDF document and share it among the colleagues and stakeholders easily. As observed in Figure 18, PDF export has two options to export the test case results.

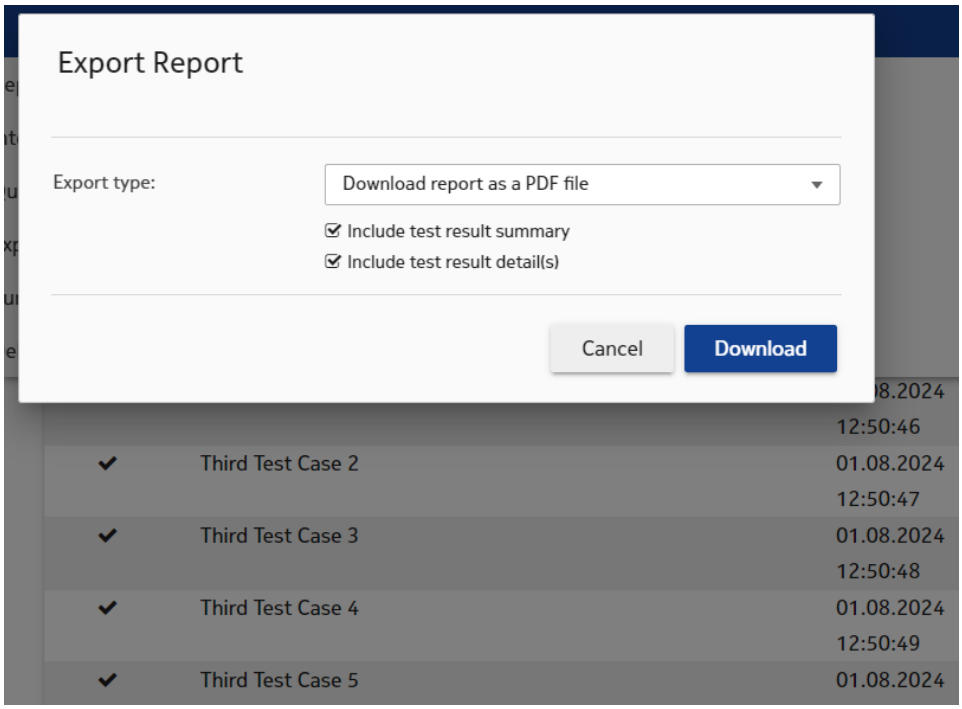


Figure 18. Illustration of the UI of the PDF Export Functionality

The first option is ‘include test result summary’. When users select this option, a list of test case result summaries is generated in a PDF document as observed in Figure 19.

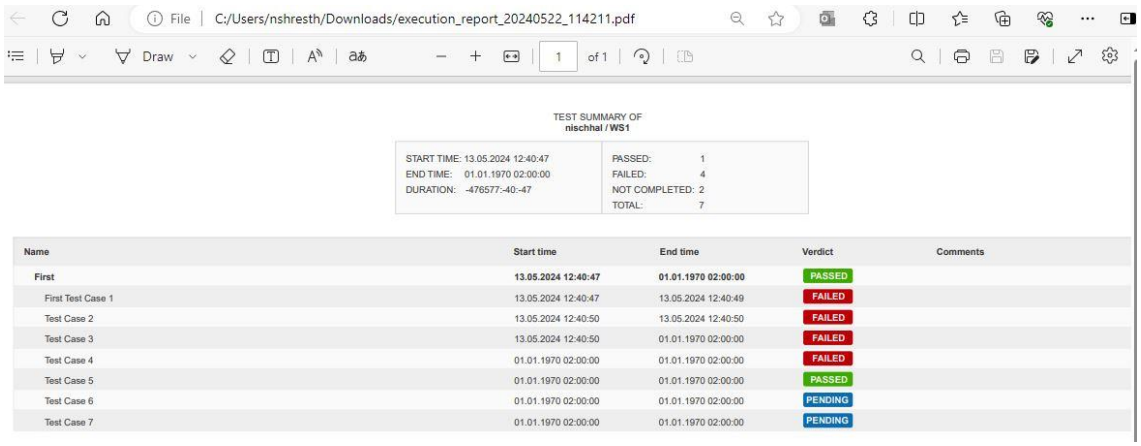


Figure 19. Illustration of Test Result Summary in PDF Document

The second option is ‘include test result detail(s)’ which enables the user to export a single test case result detail or multiple test case result details in a PDF document by selecting the test case result checkbox. This option almost mimics the UI of the dashboard page in the PDF document as observed in Figure 20.

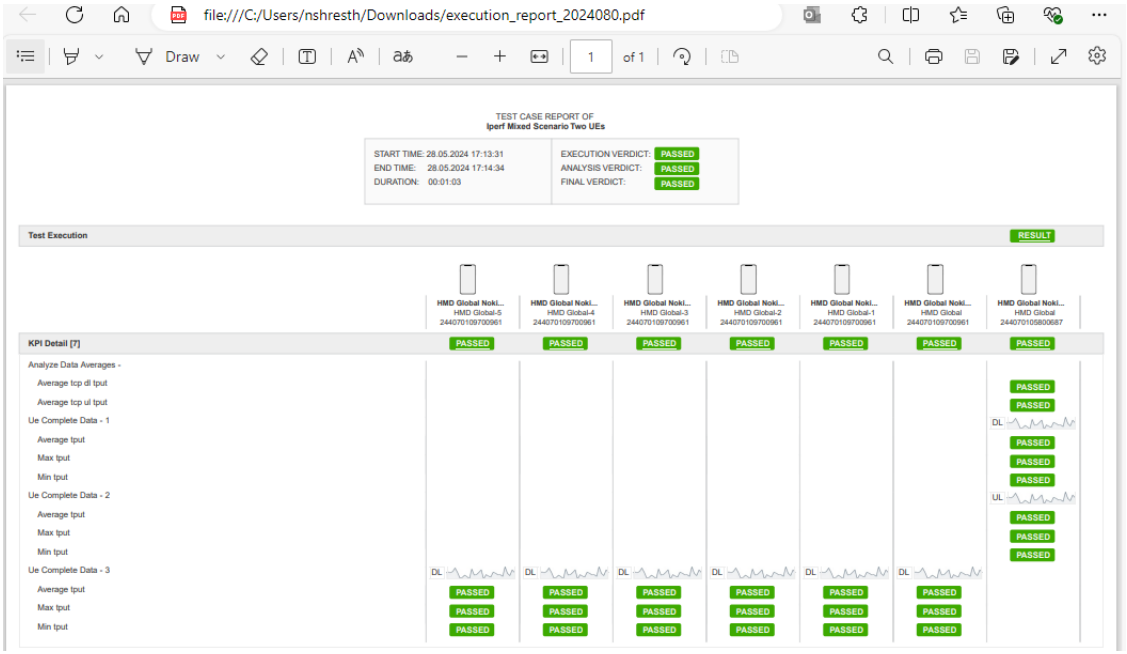


Figure 20. Illustration of Test Result Details in the PDF Document

# 5 Web Report Improvements Feature Implementation

The Web Report Improvements feature includes a range of functionalities designed to enhance the readability and accessibility of test case results to enhance user experience. These functionalities are described below.

## 5.1 Dropdown Menu

Figure 21 illustrates the logic of drop-down menu functionality. The ‘handleOptionChange’ function handles the direct selection of a test case from the drop-down menu. When users select the desired test case from the drop-down menu options, the ‘handleOptionChange’ function is executed. This

function updates the ‘setTestCaseIndex’ state, which then stores the selected test case ID. Then an API call is done from the frontend to the backend to retrieve data via a test case ID as illustrated in Figure 22. If data fetching is successful, then the dashboard UI is re-rendered displaying corresponding test case result details. From the drop-down menu options, users can also select the ‘FAILED TEST CASES’ option as well. This will filter the option list displaying only failed test cases to enhance analysis of failed test case keywords.

The ‘previousTestCase’ and ‘nextTestCase’ functions handle the selection of test cases one by one. When users click the next button on the right side of the dropdown menu, the ‘nextTestCase’ function is triggered. This function increments the test case index by one, updates the state with ‘setTestCaseIndex’, and then performs an API call to the backend to retrieve the data for the next test case, as shown in Figure 21. The retrieved data is then displayed on the dashboard page. Conversely, when users click the previous button on the left side of the dropdown menu, the ‘previousTestCase’ function is executed. This function decreases the test case index by one, and similarly, an API call is made to fetch the corresponding test case data from the backend, which is then displayed on the dashboard page.

```
const handleOptionChange = (event) => {
  const value = Number(event.target.value);
  setDropDownMenu({
    title: value === -1 && failedTestCaseList.length > 1 ? 'ALL TEST CASES' : 'FAILED TEST CASES',
    value: value === -1 ? -2 : -1,
  });
  setIsFailedTestCaseListSelected(value === -1 && failedTestCaseList.length > 0);

  if (value === -1) {
    setTestCaseId(failedTestCaseList[0].id);
  } else if (value === -2) {
    setTestCaseId(testCaseList[0].id);
  } else {
    setTestCaseIndex(value);
  }
};

const previousTestCase = () => {
  if (testCaseIndex > dropDownMenuMinIndex) {
    setTestCaseIndex(testCaseIndex - stepSize);
  }
};

const nextTestCase = () => {
  if (testCaseIndex < dropDownMenuMaxIndex) {
    setTestCaseIndex(testCaseIndex + stepSize);
  }
};
```

Figure 21. Illustration of Drop-Down Menu Code

```
Effect('getTestCaseDetails', async (caseId) => {
  try {
    const { data } = await axios.get(`/api/results/cases/${caseId}`);
    if (!_.isEmpty(data)) {
      Actions.updateState({ collection: 'dashboard', data });      Shrestha Nischhal, last month
    }
  } catch (error) {
    Actions.addNotification({
      type: 'error',
      text: isTimeout(error)
        ? 'Timeout occurred while retrieving test case details.'
        : _.get(error, ['response', 'data', 'message'], 'Unable to retrieve test case details.'),
    });
  }
});
```

Figure 22. Illustration of an API call from the frontend

## 5.2 Test Case Result Summary Section

Figure 23 illustrates the determination of test case verdicts and time. When the value of the modified verdict of a test case is null, then the 'finalVerdict' value is taken from the test case verdict. If the value of modified verdict value is not null, then the 'finalVerdict' value is equal to the modified verdict value. 'executionVerdict' and 'analysisVerdict' are determined based on arrays of execution and analysis details, respectively. If these arrays are empty, then default values are used. Otherwise, verdicts are resolved by considering both modified and original verdicts. To calculate the test case, the start time and end time 'convertTimeFormat' function is used. This function takes test case 'startTime' or 'endTime' a unix timestamp and 'utcOffset' in minutes to adjust time accordingly as parameters and returns the test case start or end time in the 'hours:minutes:seconds' format. On the other hand, 'getTestDuration' returns the difference between 'endTime' and 'startTime' in the 'hours:minutes:seconds' format.

```
const finalVerdict = isNil(testCase.modifiedVerdict) ?
testCase.verdict : testCase.modifiedVerdict;

const executionVerdict =
executionDetails.length === 0
  ? 1
  : resolveVerdict(executionDetails.map((a) =>
    (isNil(a.modifiedVerdict) ? a.verdict : a.modifiedVerdict)));

  const analysisVerdict =
others.length === 0
  ? 2
  : resolveVerdict(others.map((a) =>
    (isNil(a.modifiedVerdict) ? a.verdict : a.modifiedVerdict)));

const startTime = convertTimeFormat(testCase.startTime, testCase.utcOffset);
const endTime = convertTimeFormat(testCase.endTime, testCase.utcOffset);
const duration = getTestDuration(startTime, endTime);
```

Figure 23. Illustration of Test Case Analysis Verdict Calculation

## 5.3 Test Case Analysis Section

Figure 24 illustrates the rendering of test case analysis in the ‘CommonAnalysis’ component. If ‘commonAnalysisEvents’ is an array, and its length is greater than zero, only the ‘CommonAnalysis’ component is rendered. This component is grouped by analysis type so that the same analysis element is grouped in the same section.

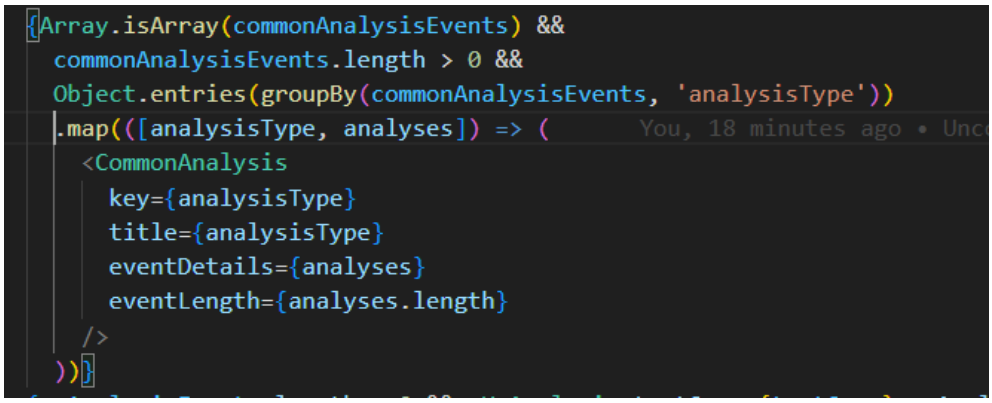


Figure 24. Illustration of the Test Case Analysis Component

## 5.4 Screenshot Section

As observed in Figure 25, ‘DisplayZoomedImage’, ‘SwipeButton’ and ‘DisplayImage’ are components which are used to construct the screen shots section. ‘DisplayZoomedImage’ displays the images in a zoomed view whereas the ‘DisplayImage’ component displays the images in the horizontal listview. The ‘SwipeButton’ component is used to navigate between images in the list.

```
<div className="screen-shots-content">
  <div className="screen-shots-title">
    <p>Screen Shots [{logs.length}]</p>
  </div>
  <div className="screen-shots-container">
    {imageZoomed ? (
      <DisplayZoomedImage
        logId={zoomedImage.logId}
        executionId={executionId}
        imageZoomed={imageZoomed}
        setImageZoomed={setImageZoomed}
        logs={logs}
        imageDetails={imageDetails}
        screenWidth={screenWidth}
      />
    ) : (
      <>
        <SwipeButton
          className={classNames('screen-shots-swipe left', {
            invisible: logs.length <= maxVisibleImages,
            visible: logs.length > maxVisibleImages,
          })}
          swipeDirection="left"
          handleSwipe={handleSwipe}
          isDisabled={imageIndex === MIN_IMAGE_INDEX}
          buttonLocation="ScreenShotTab"
          btnText="&lt;"
        />
        <div
          className={`screen-shots-images-container ${
            logs.length < maxVisibleImages ? 'flex-start' : 'space-around'
          }`}
        >
          {logs.slice(imageIndex, imageIndex + maxVisibleImages).map((log) => (
            <DisplayImage
              key={log.id}
              logId={log.id}
              executionId={executionId}
              description={log.description}
              imageZoomed={imageZoomed}
              setImageZoomed={setImageZoomed}
              setZoomedImage={setZoomedImage}
              totalImage={logs.length}
              filePath={log.name}
              imageDetails={imageDetails}
            />
          ))}
        </div>
        <SwipeButton
          className={classNames('screen-shots-swipe right', {
            invisible: logs.length <= maxVisibleImages,
            visible: logs.length > maxVisibleImages,
          })}
          swipeDirection="right"
          handleSwipe={handleSwipe}
          isDisabled={imageIndex === logs.length - maxVisibleImages}
          buttonLocation="ScreenShotTab"      Shrestha Nischhal, 4 months ago • 
          btnText="&gt;"
        />
      </>
    )}
  </div>
</div>
```

Figure 25. Illustration of the Screenshots Component

Figure 26 illustrates the UI of ‘DisplayImage’ component. This component is dynamically generated based on the ‘logs’ array. When users click a certain image from the list, the ‘handleImageClick’ function is executed. This function then opens the ‘DisplayZoomedImage’ component which displays the selected image in a zoomed view, allowing users to view the image in greater detail.

```
const handleImageClick = (id) => {
  setImageZoomed(!imageZoomed);
  setZoomedImage({ logId: id });
};

return (
  <div style={{ marginRight: totalImage < 9 ? '10px' : '0px' }}>
    <img
      className="group-image"
      src={imageUrl}
      alt="execution screenshot"
      crossOrigin="anonymous"
      onClick={() => handleImageClick(logId)}
      role="presentation"
    />
    {imageObj !== undefined &&
    <p className="screen-shot-time">{imageObj.mtime}</p>}
    {description === null ? (
      <p>Description not available</p>
    ) : (
      <>
        {description.length <= maxDescriptionLength ? (
          <p>{description}</p>
        ) : (
          <ModifiedMarker description={description}
          maxDescriptionLength={maxDescriptionLength} />
        )}
      </>
    )}
  </div>
);
```

Figure 26. Illustration of the Display Image Component

## 5.5 UE View Section

As illustrated in Figure 27, the ‘UeList’ component displays a horizontal list of all UEs utilized during test execution. The list is displayed in reverse order, starting from the right side of the dashboard and extending towards left. This component displays the UE image and details which are aligned vertically. This component also contains swipe buttons to navigate between the different UEs. These swipe buttons are displayed in the dashboard only if the maximum number of UEs is greater than the total number of UEs to display in the dashboard. UE analyses are then grouped by analysis type. These grouped analyses data are displayed via the ‘UeAnalysisTitle’ and ‘UeAnalysisDetailedResult’ components. The total number of ‘UeAnalysisTitle’ and ‘UeAnalysisDetailedResult’ components will be based on the total number of analysis types.

The ‘UeAnalysisTitle’ component will display the UE view section header which contains the analysis type and analysis length as a title followed by analysis results. These results are also displayed in a reverse order, and they are aligned horizontally, starting from the left side of the dashboard.

The ‘UeAnalysisDetailedResult’ component displays all the crucial data related to the analysis. On the left-hand side of this component, there is a UE keywords path followed by the UE event name which is aligned vertically. On the right-hand side, a list of graph thumbnails and analysis results is displayed in a reverse order from right to left. These graph thumbnails and analysis results are aligned vertically to each other. On the other hand, the graphs and their respective UE keywords path and analysis results are aligned horizontally with their respective UE event names.

```
<div className="ue-analysis-container">
  <UeList testCaseUes={testCaseUes} ueIndex={ueIndex} maxUeToDisplay={maxUeToDisplay}
  setUeIndex={setUeIndex} />
  {Array.isArray(modifiedUeAnalysisEvents) &&
    modifiedUeAnalysisEvents.length > 0 &&
    Object.entries(groupBy(modifiedUeAnalysisEvents, 'analysisType'))
    .map(([analysisType, analyses], index) => (
      <div
        className={classNames('ue-analysis-title-result-container', {
          'add-top-margin': index !== 0,
        })}
        key={analysisType}
      >
        <UeAnalysisTitle
          ueIndex={ueIndex}
          title={analysisType}
          eventDetails={analyses}
          maxUeToDisplay={maxUeToDisplay}
          ueEventLength={analyses.length}
          hideAnalysisResults={hideAnalysisResults}
          setHideAnalysisResults={setHideAnalysisResults}
        />
        {!hideAnalysisResults.includes(analysisType) && (
          <UeAnalysisDetailedResult
            ueIndex={ueIndex}
            testCase={testCase}
            title={analysisType}
            eventDetails={analyses}
            graphWidth={graphWidth}
            graphHeight={graphHeight}
            screenWidth={screenWidth}
            maxUeToDisplay={maxUeToDisplay}
            hideAnalysisResults={hideAnalysisResults}
          />
        )}
      </div>
    ))}
  {Array.isArray(ueAnalysisWithAllEmptyEvents) &&
    ueAnalysisWithAllEmptyEvents.length > 0 &&
    Object.entries(groupBy(ueAnalysisWithAllEmptyEvents, 'analysisType'))
    .map(([analysisType, analyses]) => (
      <div className="ue-analysis-title-result-container add-top-margin"
      key={analysisType}>
        <UeAnalysisTitle
          ueIndex={ueIndex}
          title={analysisType}
          eventDetails={analyses}
          maxUeToDisplay={maxUeToDisplay}
          ueEventLength={analyses.length}
          hideAnalysisResults={hideAnalysisResults}
          setHideAnalysisResults={setHideAnalysisResults}
        />
      </div>
    ))}
</div>
```

Figure 27. Illustration of the UE View Component

The UE analysis chart is displayed using React Recharts library. It is an open-source library for creating data visualizations in React applications. This library supports various chart types such as line, bar, pie and radar chart (Recharts, 2024). As observed in Figure 28, Ue analysis chart is only visible when the length of 'graphList' array is greater than zero. This 'graphList' can be update via users by selecting or deselecting the graphs. Currently, two methods of graph selection are provided to users: horizontal graph selection and free graph selection.

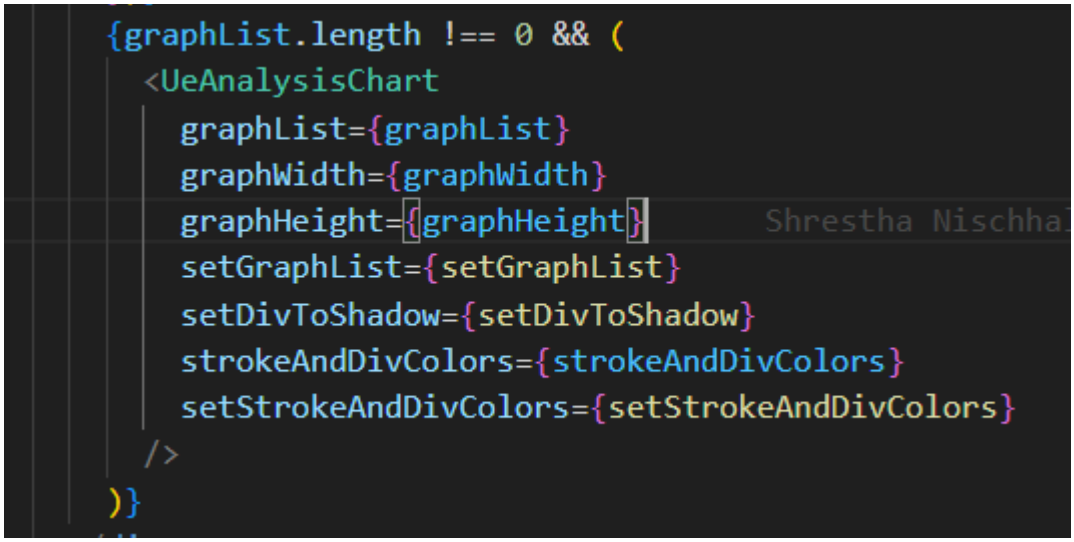


Figure 28 Illustration of the Analysis Chart Component

### 5.5.1 Horizontal Graph Selection

When users click on 'UeAnalysisTitle' on the left-hand side of the dashboard page, the 'handleHorizontalGraphSelection' function is executed as illustrated in Figure 29. This function first stores the list of graphs with valid IDs in the 'graphsWithValidId' array. After that, the length of the 'graphsWithValidId' array is checked to ensure that this array has at least one valid graph. If this array does not contain any valid graph, then the analysis chart will not be displayed. If it contains valid graphs, then the 'graphList' state is updated via 'setGraphList'. As soon as the 'graphList' updates the corresponding graph, the data are displayed in the chart.

When the chart is displayed successfully, the ‘handleDivShadow’ function is executed as observed in Figure 25. This function gives the selected ‘UeAnalysisTitle’ component a grey background colour, so that users know which component is selected as illustrated in Figure 17. To view data for a different graph, users can click on another ‘UeAnalysisTitle’ component in the list. This action updates the ‘graphList’ with new data, which results in the chart displaying the updated information. If the user clicks the same component again, the data associated with that component is removed from the ‘graphList’, and the chart is subsequently removed from the UI.

To add colour to data displayed in the chart, users can click the small square box located in left side of the graph thumbnail. Users can click the small square box and the respective colour will be added to that data in the chart if that graph data is active in the chart. When users click the same square box for the second time, colour will be removed from that data. Similarly, active graph data can be removed from the chart by clicking the relative graph thumbnail, and a new graph can be added from the same horizontal graph list by clicking the graph the data of which is not yet visible in the chart.

```
const handleHorizontalGraphSelection = (selectedGraphs) => {
  const graphsWithValidId = selectedGraphs.filter(
    (graph) => graph && graph.id !== null && graph.id !== INVALID_GRAPH_ID
  );

  if (graphsWithValidId.length === 0) return;
  setStrokeAndDivColors([]);

  const graphsToDisplay = graphsWithValidId.filter((graph) =>
    new Set(visibleGraphList.map((visibleGraph) => visibleGraph.id)).has(graph.id)
  );

  const horizontalGraphList = graphsToDisplay.map((graphToDisplay) => {
    const graphDetail = graphs.find((graph) => graph.id === graphToDisplay.id);

    return graphObject(graphDetail);
  });

  setGraphList(horizontalGraphList);
};

const handleDivShadow = (divGraphList, throughputTitle) => {
  const foundValidGraphId = divGraphList.some((graph) =>
    graph.id !== null && graph.id !== INVALID_GRAPH_ID);
  if (!foundValidGraphId) return;      Shrestha Nischhal, 4 months ago • Full dashboa
  if (divToShadow === throughputTitle) {
    setDivToShadow(null);
  } else {
    setDivToShadow(throughputTitle);
  }
};
```

Figure 29. Illustration of Horizontal Graphs Selection Logic

### 5.5.2 Free Graph Selection

As illustrated in Figure 30, when a graph is selected arbitrarily from the UE view, the ‘setFreelySelectedGraphs’ function is executed. This function first evaluates whether horizontal graph selection is currently active by examining the value of the state variable ‘divToShadow’. If the horizontal graph selection is active, then this function adheres to the logic associated with the horizontal graph selection logic. If the horizontal graph selection is not active, then users can add or remove a graph by clicking on graph thumbnail. The colouring of the graph data in the chart follows the same principles as those used in the horizontal graph selection.

```
const setFreelySelectedGraphs = (graph) => {
  if (!graph) return;
  /**
   * When graph dataset is selected horizontally
   * Free selection of graph is enabled for that horizontal row
   */
  if (!isNil(divToShadow)) {
    if (!checkKeywordName(graph.keywordName, graphList[0].keywordName)) return;
    addOrRemoveGraph(graph, graphList, setGraphList, maxUeToDisplay, setStrokeAndDivColors);
    return;
  }

  if (graphList.length === 0) {
    setGraphList([...graphList, graphObject(graph)]);
    setStrokeAndDivColors([...strokeAndDivColors, { id: graph.id }]);
    return;
  }

  if (graph.type !== graphList[0].type) return;      Shrestha Nischhal, 4 months ago • Full
  if (!checkKeywordName(graph.keywordName.replace(/\d$/, ''),
  graphList[0].keywordName.replace(/\d$/, ''))) return;
  addOrRemoveGraph(graph, graphList, setGraphList,
      MAX_STROKE_AND_DIV_COLOR, setStrokeAndDivColors);
};
```

Figure 30. Illustration of Free Graphs Selection Logic

## 5.6 PDF Export

The test result data are converted to a PDF file using the PDFKit library. PDFKit is a versatile library designed for generating PDF files, suitable for both Node.js environments and a web browser (PDFKit). This library facilitates the creation of complex, multi-page, printable documents with ease. Figure 31 illustrates the ‘generateSummaryTableContent’ function which is used to format and insert text into the PDF document. ‘doc.font’ sets the font type for text, ‘doc.fontSize’ sets the font size for the text, and ‘doc.fillColor’ sets the colour of the text. In addition, doc.text sets the content of the PDF document specified by variable ‘text’ in the horizontal and vertical positions specified by variables ‘xPos’tion’ and ‘yPosition’.

```
export const generateSummaryTableContent = (doc, text, xPosition, yPosition, fullPageWidth) =>
{
  doc
    .font('Helvetica')
    .fontSize(12)
    .fillColor('#eeeeee')
    .text(text, xPosition, yPosition, { width: fullPageWidth });
};
```

Figure 31. Illustration of text generation in the PDF document using PDFKit

As observed in Figure 33, ‘generateAnalysisSummaryInPdf’ and ‘generateAnalysisDataInPdf’ are crucial for generating two distinct types of PDF documents. The ‘generateAnalysisSummaryInPdf’ function is executed when users select the ‘include test result summary’ option, producing a summary of test case results in a PDF document. Conversely, the ‘generateAnalysisDataInPdf’ function is executed when users select the ‘include test result detail(s)’ option, creating detailed analysis of selected test cases in the PDF document. Users can generate PDF documents using two methods: selecting each option separately or choosing both options simultaneously.

When users select either the ‘include test result summary’ or ‘include test result detail(s)’ option, the corresponding backend functions ‘generateAnalysisSummaryInPdf’ or ‘generateAnalysisDataInPdf’ are executed. Upon completion of these functions, a temporary PDF document is generated on the backend with the file path stored in the ‘filePath’ variable. This document is then transmitted to the front-end to be downloaded as a PDF document. Subsequently, the temporary PDF document is removed by the

'handleFileStreamResult' function to ensure that no residual documents remain on the backend as observed in Figure 32.

```
const handleFileStreamResult = (res, fileStream, filePath) => {
  fileStream.on('error', (err) => {
    logger.error(`(Download PDF Reports) sending pdf report ${err}`);
    res.status(500).json({ status: 'error', message: 'Error encountered while downloading report'
  });

  fileStream.on('close', () => {
    if (existsSync(filePath)) unlinkSync(filePath);
  });
};

export const sendPdfFile = (doc, res, executionIdsWithError, filePath, fileName) => {
  doc.on('end', () => {
    if (executionIdsWithError.length > 0) {
      if (existsSync(filePath)) unlinkSync(filePath);
      res.status(500).json({ status: 'error',
        message: 'Error encountered while downloading report' });
    } else {
      const fileStream = createReadStream(filePath);
      res.setHeader('Content-Type', 'application/pdf');
      res.setHeader('Content-Disposition', `attachment; filename=${fileName}`);
      fileStream.pipe(res);
      handleFileStreamResult(res, fileStream, filePath);
    }
  });
```

Figure 32. Illustration of PDF Document Creation

When users select both options, 'include test result summary' or 'include test result detail(s)', simultaneously, the functions 'generateAnalysisSummaryInPdf' and 'generateAnalysisDataInPdf' are executed concurrently. As observed in Figure 33, upon successful execution of these functions, temporary PDF documents are created on the backend with the file path stored in the 'testResultSummaryPath' and 'testResultDetailPath' variables. These PDF documents are then compressed into ZIP files using the Archiver library. Archiver is a Node.js library used for creating archive files in various formats, such as ZIP and TAR (Archiver, 2024). The compressed file is subsequently sent to the front-end to be downloaded as a ZIP file. Finally, the temporary files created on the backend are deleted.

```
const { testResultSummaryPath, testResultSummary } = generateAnalysisSummaryInPdf(
  testExecution,
  testSuites,
  tempfileDirPath
);

const { testResultDetailPath, testResultDetail } = generateAnalysisDataInPdf(
  req,
  testCases,
  tempfileDirPath,
  executionIds[0],
);

if (executionIdsWithError.length > 0) {
  deleteTempReportFile(testResultSummaryPath);
  deleteTempReportFile(testResultDetailPath);
  res.status(500).json({ status: 'error', message:
    'Error encountered while downloading report' });
} else {
  const archive = archiver('zip', { zlib: { level: 9 } });

  res.setHeader('Content-Type', 'application/zip');
  res.setHeader('Content-Disposition', `attachment;
    filename=${testResultSummary}_${testResultDetail}.zip`);

  archive.pipe(res);

  archive.append(createReadStream(testResultSummaryPath),
   { name: `${testResultSummary}_${getTimestamp()}.pdf` });
  archive.append(createReadStream(testResultDetailPath),
  { name: `${testResultDetail}_${getTimestamp()}.pdf` });

  archive.finalize();
  deleteTempReportFile(testResultSummaryPath);
  deleteTempReportFile(testResultDetailPath);
}
```

Figure 33. Illustration of the creation of PDF documents as a Zipped File

# 6 Evaluation

## 6.1 Usability Testing

The purpose of usability testing is to evaluate the ease of use and user-friendliness of a website or an application by observing real users as they interact with it. In the context of the Web Report Improvement Feature, usability testing was done by the TAT release testing team who was not involved with this feature,

to ensure unbiased assessment. A controlled testing environment was prepared to mimic the real-world usage conditions. During the usability testing period, various aspects of this feature were assessed, including:

- Exporting test case results as PDF documents: Testers verified the functionality of exporting test case results into the PDF format, ensuring the process was intuitive and met user needs.
- Functionality of drop-down menu: The usability of the drop-down menu for navigating between test cases was assessed to confirm that users could easily switch between different results without confusion.
- Responsiveness of the dashboard page: The dashboard's responsiveness was tested by having testers interact with the page across different screen sizes to ensure it displayed correctly and functioned smoothly.
- Calculation of test case analysis verdicts: Testers evaluated the accuracy of the test case verdicts and their alignment with the expected outcomes.
- UE view UI: The UI of the UE view was tested by injecting diverse types of UE analysis data from the backend. This was done to assess how well the UI handled various data inputs and maintained clarity and usability.
- Graph data: Testers verified that the graph data was accurately displayed in the charts, using both horizontal and free graph selection methods.

During the testing phase several bugs were reported, which include:

- Incorrect values of test case analysis verdicts: These analysis verdict values displayed in the dashboard were found to be incorrect. This bug was classified as severity level C-Minor.
- Incorrect calculation of test case duration: The test case duration value for some of the test cases were displayed as "undefined" in the dashboard. This bug was also classified as C-minor.
- Incorrect test case data display: When selecting the failed test case list from the drop-down menu, incorrect test case data was displayed in the dashboard. This bug was classified as severity level B-Major.
- Dashboard page crashing: Some of the test cases had the keyword path value as "null" which led to the crash of the dashboard page. This bug was classified as severity level A-Critical.

These issues were addressed and resolved before merging the feature in the main Git branch and subsequently deployed to production.

## 6.2 User Feedback and Satisfaction Analysis

Prior to the release of the Web Report Improvement feature, the development team at Nokia organized a demo session for the customers for whom this feature was specifically designed. During this demonstration, the senior architect provided a detailed, step-by-step walkthrough of this feature. The feedback on this initial demo was highly positive. After the feature was deployed to production and used by actual customers, it continued to receive favourable feedback. However, some minor bugs were reported, such as non-UE analysis data not being displayed in the UI and the test case report navigation link not working for certain test cases. These issues were promptly addressed to ensure that customer needs and expectations were met.

## 6.3 Comparison with Previous System

The Web Report Improvement feature represents a significant upgrade over the previous system. The previous system suffered from the lack of satisfactory user experience and user-friendly design in terms test results and the test result detail page. This new feature addresses the shortcomings of the previous system by introducing user-friendly design and functionality. This new feature offers enhanced user experience with an improved navigation system, a compact view for test case detail page alongside with PDF export options on the test result page. Web Report Improvement not only enhances the UI but also includes new functionalities which were unavailable in the previous system. Thus, providing more comprehensive and satisfying experiences for users.

# 7 Discussion and Conclusion

A Web Report Improvement feature was developed to enhance user experience, addressing previous system flaws. The primary goal of this project was to provide a user-friendly design and new functionalities to streamline test result management. The PDF export options allow users to transform test result data into a PDF document and share the document among the colleagues and shareholders with ease. The drop-down menu provides a seamless navigation option between different test results for users, eliminating the need to navigate back and forth between the pages. The dashboard provides a compact view of test case results in a single page.

During the demonstration of the Web Report Improvement feature, stakeholders were highly impressed by the functionalities introduced in this feature. The team involved in this feature was congratulated and applauded by stakeholders for achieving this milestone. After the feature was deployed to production, only minor bugs were reported by users, as mentioned earlier. This minimal report of post-deployment issues suggests that users are generally satisfied with the implementation and have found the new feature to meet their expectations effectively.

One of the primary challenges for implementing the Web Report Improvement feature was understanding the extensive and complex code base of TAT. The size and complexity of the code base made it difficult to integrate this feature without disrupting existing functionality. Another major challenge was the complexity of the TAT database, which served data from the backend to the front-end. This complexity of the database hindered the ability to learn about the relation between different tables in the database. Furthermore, organizing the complex data of the UE analysis proved to be one of the most challenging aspects of this feature. The analysis data needed to be re-organized perfectly to be displayed in the UE view. This required writing several helper functions to ensure that the analysis data was well organized and effectively displayed in the UI.

To improve this feature further in the future, the feature should be implemented using TypeScript, and the PDFKit library should be replaced with the Puppeteer library. The Web Report Improvement feature is currently implemented using React and JavaScript, which results in limitations regarding type safety and code maintainability. Converting this feature to TypeScript could address these issues. TypeScript is a superset of JavaScript, which includes JavaScript features alongside with additional features. TypeScript addresses the limitation of JavaScript as it includes features such as type annotations and advanced structural elements (Fenton, 2018, p.1). TypeScript can enhance type safety, improve code maintainability, and provide better development tools.

The PDFKit library should be replaced with the Puppeteer library for generating PDF documents in the backend. PDFKit requires using different syntax from Hypertext Markup Language (HTML) to generate the PDF document. Developers will be forced to learn a new language if bugs related to PDF document generation arise in the backend. In contrast, Puppeteer leverages HTML and CSS for PDF document generation, aligning closely with technologies used in React. This familiarity simplifies the fixing of bugs related to PDF document generation, as developers do not need to learn any new language. Additionally, PDFKit requires manual specification of the y-coordinate to position content in the PDF document, which complicates the layout management. Puppeteer, however, generates PDF documents directly from HTML and Cascading Style Sheets (CSS), thus making content formatting easier and ensuring the design of the PDF documents matches the web design. As the current Node.js version of TAT does not support Puppeteer, this library can be introduced once the Node.js version is upgraded to and compatible with Puppeteer.

In conclusion, the Web Report Improvement feature has significantly enhanced system design and user experience. By introducing new functionalities such as PDF export options, a drop-down menu for seamless navigation, and a compact dashboard view, this feature has addressed previous system flaws and streamlined the process of managing test case result data. The positive feedback from stakeholders during feature demonstration and the minimal post-

deployment issues underscores the success of the implementation. By adopting the recommended improvements, the feature can be further refined to ensure it remains adaptable, efficient, and aligned with evolving user requirements and technological advancements.

## References

Adobe. 2024. Why PDF Is the Best Format for Business. Available at: https://www.adobe.com/acrobat/hub/why-pdf-is-best-format-for-business.html [Accessed: 15 March 2024].

Afonso, D. & Mestre, R. 2023. State Management with React Query. Packt Publishing, Limited.

Angular. 2024. What Is Angular? Available at: https://v17.angular.io/guide/what-is-angular [Accessed 5 August 2024].

Archiver. 2024. Archiver Documentation. Available at: https://www.archiverjs.com/docs/archiver/ [Accessed: 5 August 2024].

Ayaz, M.A. & Obaid, N., 2021. Angular Cookbook. Packt Publishing.

Baer, E. 2018. What React Is and Why It Matters. O'Reilly Media, Inc.

Budventure Technology. 2023. Why Use React When Developing Websites. Budventure Technology. Available at: https://www.budventure.technology/blog/why-use-react-when-developing-websites [Accessed: May 20, 2024].

Canziba, E. 2018. Hands-on UX Design for Developers. Packt Publishing.

Clow, M. 2018. Angular 5 Projects. Apress.

Dragusin, C. 2023. Building Scalable Applications: Why React? Bit Blog. Available at: https://blog.bitsrc.io/building-scalable-applications-why-react-7a6c83073fab [Accessed: 22 February 2024].

Fenton, S., 2018. Pro TypeScript. Apress.

Hands-On React. 2024. React Architecture. Available at: https://handsonreact.com/docs/architecture [Accessed 5 August 2024].

Laravel. Routing. 2024. Available at: https://laravel.com/docs/11.x/routing [Accessed: 5 August 2024].

Mead, A. 2018. Learning Node.js Development. Packt Publishing.

PDFKit. Available at: https://pdfkit.org/ [Accessed 5 August 2024].

Recharts. 2024. API Documentation. Available at: https://recharts.org/en-US/api [Accessed: August 5, 2024].

Robot Framework. 2024. Robot Framework. Available: https://robotframework.org/ [Accessed: May 21, 2024].

Shavin, M. 2023. Learning Vue. O'Reilly Media, Inc.

Sinha, S. 2019. Beginning Laravel. Apress.

Vue.js. Component Basics. Available at: https://vuejs.org/guide/essentials/component-basics.html [Accessed: 5 August 2024].

Wieruch, R. 2019. React Function Components. Available at: https://www.robinwieruch.de/react-function-component/ [Accessed: May 20, 2024].

# UE Helper Functions that Re-organize UE Analysis Data

```
const eventObject = (name, eventGroup2, keywordPath) => ({
  id: null,
  graphId: null,
  eventInfo: {
    id: null,
    name,
    eventGroup2,
    keywordPath,
  },
});

const createUniqueEventsByAnalysisType = (ueAnalysisWithAtleastOneNonEmptyEvent) =>
  Object.entries(groupBy(ueAnalysisWithAtleastOneNonEmptyEvent, 'analysisType')).reduce(
    (acc, [analysisType, analyses]) => {
      const existingEventInfos = new Set();

      const uniqueEvents = analyses.reduce((eventsAcc, { events }) => {
        events.forEach((event) => {
          const { name, keywordPath, eventGroup2 } = event.eventInfo;
          const eventInfoString = `${extractKeywordName(keywordPath)}-${eventGroup2}-${name}`;
          if (!existingEventInfos.has(eventInfoString)) {
            existingEventInfos.add(eventInfoString);
            eventsAcc.push(event);
          }
        });
        return eventsAcc;
      }, []);

      acc[analysisType] = uniqueEvents;
      return acc;
    },
    {}
  );
```

```
export const createEventInfoForEmptyEvents = (ueAnalysisWithAtleastOneNonEmptyEvent) => {
  const uniqueEventsByAnalysisType = createUniqueEventsByAnalysisType(ueAnalysisWithAtleastOneNonEmptyEvent);

  const eventInfo = ueAnalysisWithAtleastOneNonEmptyEvent.map((analysis) => {
    const type = analysis.analysisType;
    if (
      analysis.events.length === 0 &&
      uniqueEventsByAnalysisType[type] &&
      uniqueEventsByAnalysisType[type].length > 0
    ) {
      return {
        ...analysis,
        events: uniqueEventsByAnalysisType[type].map((event) => {
          const { name, eventGroup2, keywordPath } = event.eventInfo;

          return eventObject(name, eventGroup2, keywordPath);
        }),
      };
    }
    return analysis;
  });
  return eventInfo;
};
```

```
export const getGroupedEventByGraphId = (eventDetails) => {
  return eventDetails.map(({ id, events, sutInfo }) => {
    const groupedEvents = events.reduce(
      (acc, { id: eventId, graphId, eventInfo, modifiedVerdict, verdict, expected,
        realized }) => {
        const event = acc.find((item) => item.graphId === graphId);
        if (event) {
          const eventInfoGroup =
            event.eventInfo.eventGroup2 === eventInfo.eventGroup2 &&
            event.eventInfo.keywordPath === eventInfo.keywordPath
              ? event.eventInfo
              : null;

          if (eventInfoGroup) {
            eventInfoGroup.eventData.push({
              eventId,
              name: capitalize(eventInfo.name).replace(/_/g, ' '),
              finalVerdict: isNil(modifiedVerdict) ? verdict : modifiedVerdict,
              expected,
              realized,
              unit: eventInfo.unit,
            });
          } else {
            event.eventInfo = {
              eventGroup2: eventInfo.eventGroup2,
              keywordPath: eventInfo.keywordPath,
              eventData: [
                {
                  eventId,
                  name: capitalize(eventInfo.name).replace(/_/g, ' '),
                  finalVerdict: isNil(modifiedVerdict) ? verdict : modifiedVerdict,
                  expected,
                  realized,
                  unit: eventInfo.unit,
                },
              ],
            };
          }
```

```
      } else {
        acc.push({
          graphId,
          eventInfo: {
            eventGroup2: eventInfo.eventGroup2,
            keywordPath: eventInfo.keywordPath,
            eventData: [
              {
                eventId,
                name: capitalize(eventInfo.name).replace(/_/g, ' '),
                finalVerdict: isNil(modifiedVerdict) ? verdict : modifiedVerdict,
                expected,
                realized,
                unit: eventInfo.unit,
              },
            ],
          },
        });
      }
      return acc;
    },
    []
  );
  return { id, sutInfo, events: groupedEvents };
});
};
```

```
export const getMergedEvents = (eventDetails, INVALID_GRAPH_ID) => {
  return getGroupedEventByGraphId(eventDetails).reduce((acc, eventInfo, eventInfoIndex) => {
    eventInfo.events.forEach((event, eventIndex) => {
      const existingKeywordPath = getExistingKeywordPathIndex(acc, event);

      if (existingKeywordPath !== -1) {
        const existingItem = acc[existingKeywordPath];
        existingItem.graphs.push({ id: event.graphId });

        if (!existingItem.originalKeywordPath.find((KeywordPath) =>
          KeywordPath.name === event.eventInfo.keywordPath)) {
          existingItem.originalKeywordPath.push({
            id: `${event.eventInfo.keywordPath}-${eventIndex}-${eventInfoIndex}`,
            name: event.eventInfo.keywordPath,
          });
        }

        existingItem.keywordPath = extractKeywordName(event.eventInfo.keywordPath);

        event.eventInfo.eventData.forEach((data, dataIndex) => {
          const existingData = existingItem.eventData[dataIndex];

          if (existingData) {
            existingData.eventDetails.push({
              eventId: data.eventId,
              finalVerdict: data.finalVerdict,
              expected: data.expected,
              realized: data.realized,
              ue: eventInfo.sutInfo.metadata.model,
              unit: data.unit,
            });
          } else {
            existingItem.eventData.push({
              eventDetails: [
                {
                  eventId: data.eventId,
                  finalVerdict: data.finalVerdict,
                  expected: data.expected,
                  realized: data.realized,
                  ue: eventInfo.sutInfo.metadata.model,
                  unit: data.unit,
                },
              ],
              name: data.name,
            });
          }
        });
```

```
    } else {
      acc.push({
        graphs: Array.from({ length: eventInfoIndex + 1 }, (_, currentIndex) => ({
          id: currentIndex === eventInfoIndex ? event.graphId : INVALID_GRAPH_ID,
        })),
        eventGroup2: event.eventInfo.eventGroup2,
        originalKeywordPath: [
          { id: `${event.eventInfo.keywordPath}#${eventIndex}-${eventInfoIndex}`,
          name: event.eventInfo.keywordPath },
        ],
        keywordPath: extractKeywordName(event.eventInfo.keywordPath),

        eventData: event.eventInfo.eventData.map((data) => {
          const eventDetailsArray = Array.from({ length: eventInfoIndex + 1 }, (_, currentIndex) => {
            if (currentIndex === eventInfoIndex) {
              return {
                eventId: data.eventId,
                finalVerdict: data.finalVerdict,
                expected: data.expected,
                realized: data.realized,
                ue: eventInfo.sutInfo.metadata.model,
                unit: data.unit,
              };
            }
            return { eventId: `${data.eventId}#${currentIndex}` };
          });
          return {
            eventDetails: eventDetailsArray,
            name: data.name,
          };
        }),
      });
    }
  });
  return acc;
}, []);
};
```