OULU UNIVERSITY OF APPLIED SCIENCES

Niran Manandhar

AUTOMATED COMPARATIVE REPORT GENERATION FROM THE TEST AUTOMATION RESULT

AUTOMATED COMPARATIVE REPORT GENERATION FROM THE TEST AUTOMATION RESULT

Niran Manandhar
Bachelor’s Thesis
Autumn 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme, Information Technology

Author: Niran Manandhar
Title of the bachelor’s thesis: Automated Comparative Report Generation from Test Automation Result
Supervisor: Pertti Heikkilä
Term and year of completion: Autumn 2019 Number of pages: 35

This thesis was a development project commissioned by Profilence Oy for creating a web application. The application required to show an Automated comparative report from the company’s pre-existing software Tau.

The aim of this thesis was to get the test automation results from Tau efficiently and make a web application from the retrieved data. The web application needed to provide users with a dashboard that had a comparative report generator. The application processed the analytical data from physical test stations connected to a data server and from the devices which were already deployed and used by a testing team.

During the thesis, many new user interfaces for the user were created. The user interfaces and new features were added to the Tau web application. The new features and user interface was successfully used to create automated comparative report from test automation results.

The development was done by using Angular (version 6 and above), with UI library: Primeng and customized charting library: Highcharts. The primary testing browsers were Google Chrome and Firefox. Most of the data coming from Database from Tau was managed as JSON (JavaScript Object Notation) and protocol buffers. The application had a responsive view which could be viewed on a larger display such as a big screen or smaller tablets and pc as required by developers and stakeholders.

Keywords: test automation, report generation, user interface

TABLE OF CONTENTS

ABSTRACT 3
TABLE OF CONTENTS 4
VOCABULARY 5
1 INTRODUCTION 6
2 GENERAL THEORY 7
2.1 Test automation and its importance 7
2.2 Tau – Test automation software by Profilence 8
2.3 Overview on Angular 9
2.4 Overview on Highcharts 9
2.5 Overview on Prime ng 11
2.6 Overview on JSON and Protobuf 12
2.7 Overview on RxJS 13
2.8 Overview of Visual Studio Code 15
2.9 Overview on Postman 16
3 UI DEVELOPMENT 18
3.1 UI development for Telemetry 18
3.1.1 UI development for analytics 19
3.1.2 UI development for history 22
3.2 UI development for Test runs 23
3.2.1 UI development for Issue list 24
3.2.2 UI development for test runs 28
3.2.3 UI development for requirements 31
3.3 UI development for Devices 32
4 CONCLUSION 34
REFERENCES 35

VOCABULARY

ABBREVIATIONS	EXPLANATION
ADB	Android Debug Bridge
API	Application Program Interface
HTML	Hypertext Markup Language
Js	Java Script
JSON	JavaScript Object Notation
OEM	Original Equipment Manufacturer
PC	Personal Computer
Protobuf	Protocol buffers
Rx	Reactive Extension
RxJs	Reactive Extension for Java Script
TS	TypeScript
UI	User Interface
VS code	Visual Studio Code

1 INTRODUCTION

This thesis is related to Profilence Oy’s request for a web application that could be used by developers and stakeholders to keep track of different software build versions and their test results from the Profilence’s Tau software.

Profilence is a software company founded by veteran test automation and profiling tool developers specializing in solutions for embedded systems. Profilence's Tau has been designed for Android OEMs, carriers, and app developers alike who need to make sure their product worked as expected. Tau Tests on real devices without any modifications to source code. The entire Android operating system was managed by the software and not bound to a single application. Multiple devices were supported in one test case; for example, testing of phone calls or messaging features between those devices.

Tau is a desktop application made in *.NET* and it mostly held massive logs of data from the test runs that go through the devices. As Tau was being used by developers, a need for the developers arose to get an overview of the test run. A simplified view of the analytics of the test runs rather than rows of data was needed; such simplified analysis could also be shared with stakeholders for various managerial purposes.

Furthermore, having a web application opened new possibilities to use the data in various devices ranging from a small handheld device to keep track of data to huge display screens that are viewable by all developers in real-time in a development or a conference room.

2 GENERAL THEORY

2.1 Test automation and its importance

Test automation is the automation of tasks that are repetitive but necessary tasks in a testing process. A lot of time and resources are taken by manual testing hence it would be inefficient and resource intensive. Test automation is critical for a continuous delivery and continuous testing. An Android system like any other operating system or software comes with an iterative development process which comes with a series of updates and versions.

Android devices are manufactured by different companies and all of them have their own Android versions that differ from the stock Android. Hence, a lot of customized testing is involved for each of these manufacturers which is eased with test automation.

Developing a bug-free software is often impossible; even with a series of manual testing and an iterative development process, the delivered software has some defects and bugs. Test Automation software is the best way to increase the effectiveness, efficiency, and coverage of software testing to minimize bugs.

Manual software testing is performed mostly by a human sitting in front of a computer or a test device carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior, and recording their observations.

Manual tests are often repeated during development cycles for source code changes and other situations, e.g. multiple operating environments and hardware configurations. More constraints are added eventually, such as bandwidth, battery percentage and this makes data collection tedious and massive.

An automated testing tool can playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created, they can easily be repeated, and they can be extended to perform tasks that otherwise are impossible with manual testing.

2.2 Tau – Test automation software by Profilence

Tau was developed by Profilence Oy, Oulu Finland. Tau was purposefully designed to be able to run tests unsupervised for days to model how software really behaves eventually.

Real life events on a device such as recovery from device failures, connection issues, handling pop-up events are monitored and logged as closely as possible and hence the data collected is more accurate and usable for testing real-life events in Tau.

Tau consists of two components – the PC side Tau client and the lightweight Tau service inside the Android device. Once a device is connected to a PC running the Tau client, it automatically sets up the Tau service on the device through the ADB (Android Debug Bridge) and starts collecting data. Whereas the devices that are already deployed to send data through the Tau service running as a background service remotely.

The image below (FIGURE 1.) shows a basic diagram of how a device is connected to the Tau software.

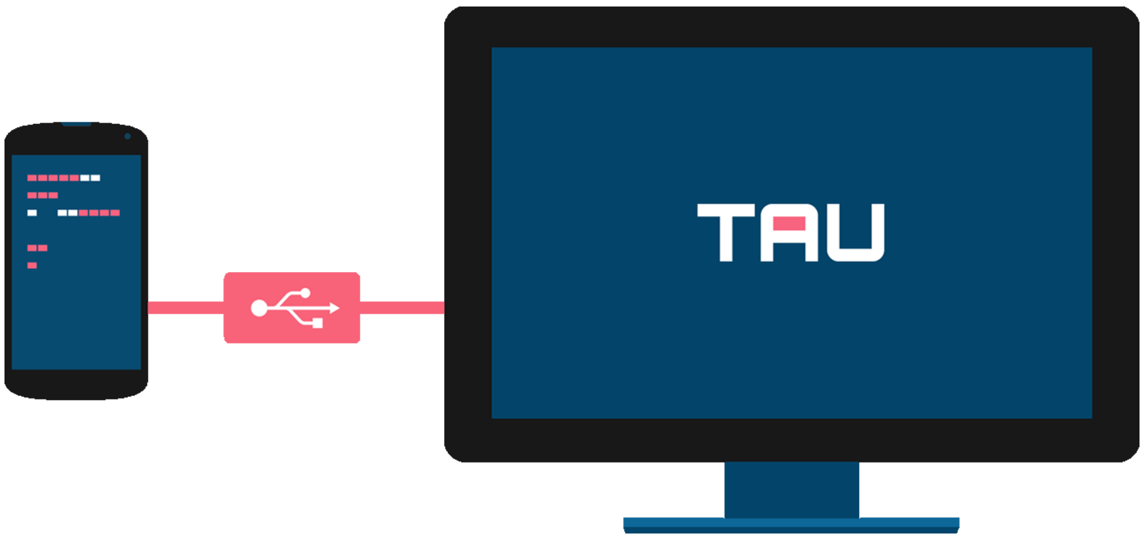


FIGURE 1. The Tau software by Profilence

2.3 Overview on Angular

Angular is a platform and framework for building client applications in HTML and TS (TypeScript). Angular code is mostly written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that can be imported into the web app.

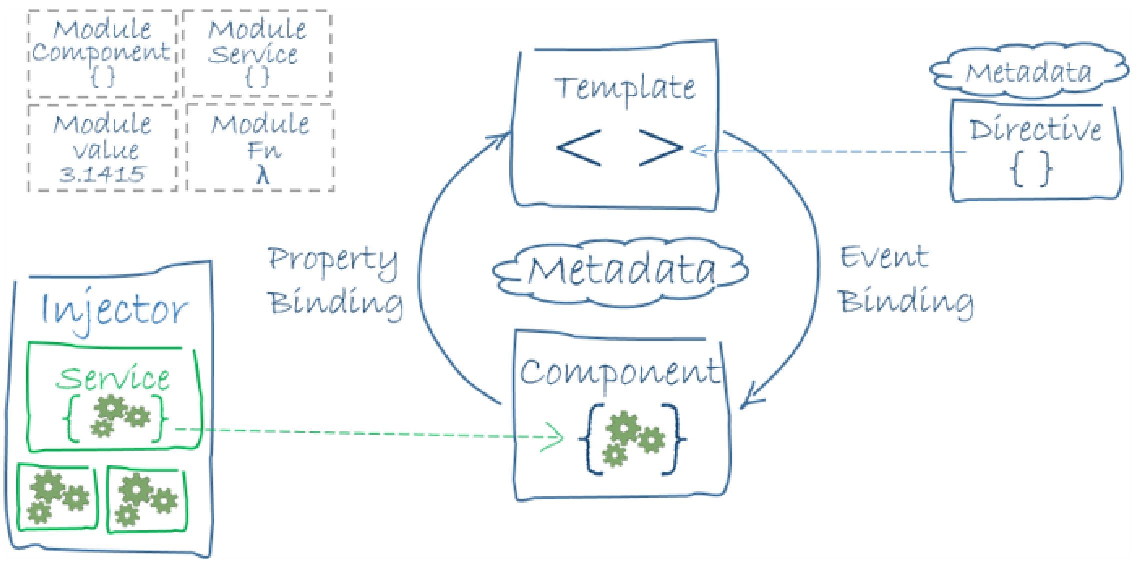


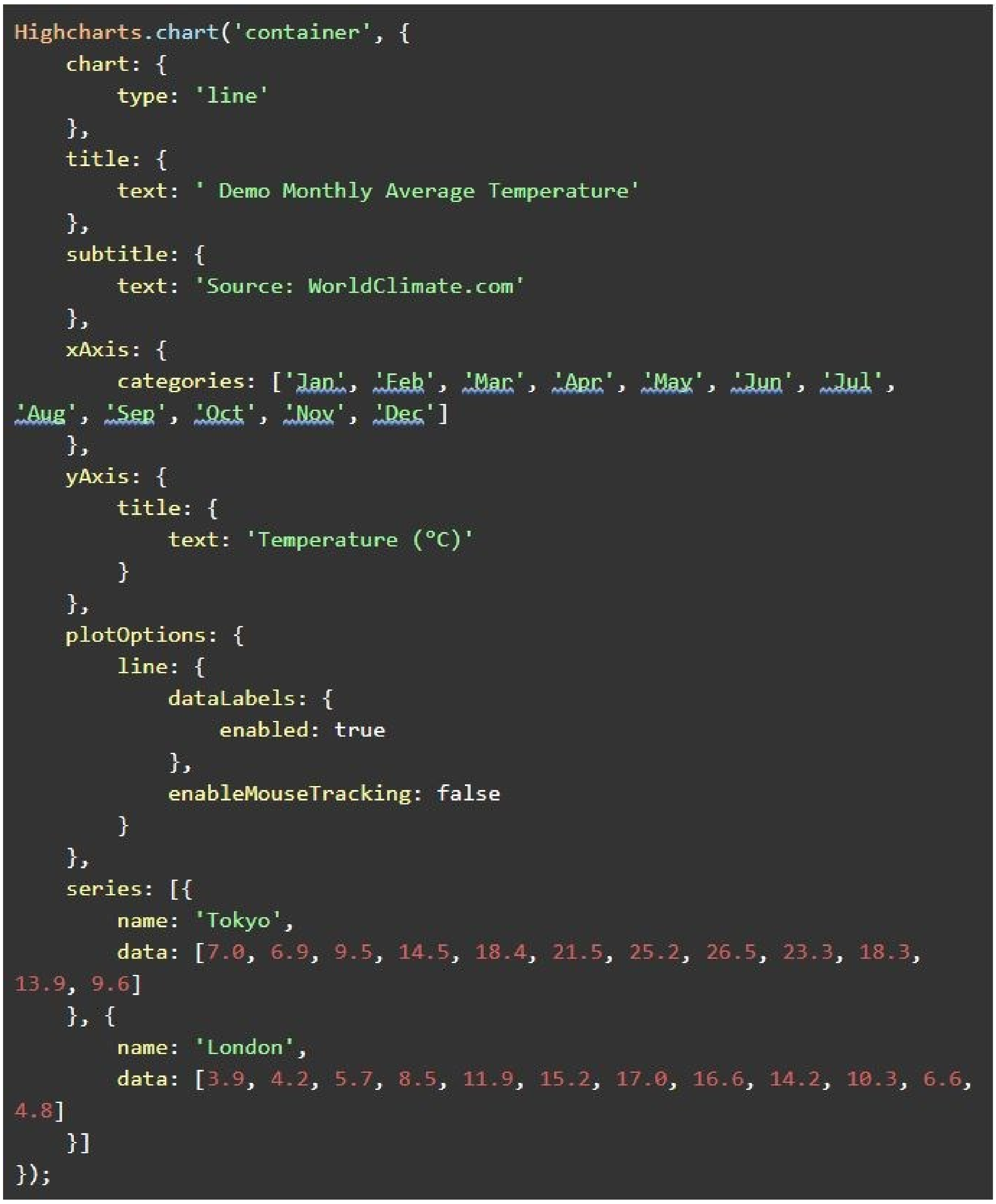
FIGURE 2. The Architecture of an Angular application (5).

The diagram above (FIGURE 2.) shows the architecture of an angular application. The angular application consists of main components written in typescript which is later converted to plain JavaScript by angular and deployed to the client.

2.4 Overview on Highcharts

Highcharts is a library written in pure JavaScript. It was developed by Highsoft, Norway, and is a globally used as a charting library. Highcharts is lightweight and has a huge customer base.

Highcharts comes with a series of updates that are managed by the open-source community. A few bugs were encountered during the development phase of the project but they were easily resolved by communicating with the developer's community.



```
Highcharts.chart('container', {
    chart: {
        type: 'line'
    },
    title: {
        text: ' Demo Monthly Average Temperature'
    },
    subtitle: {
        text: 'Source: WorldClimate.com'
    },
    xAxis: {
        categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    },
    yAxis: {
        title: {
            text: 'Temperature (°C)'
        }
    },
    plotOptions: {
        line: {
            dataLabels: {
                enabled: true
            },
            enableMouseTracking: false
        }
    },
    series: [{
        name: 'Tokyo',
        data: [7.0, 6.9, 9.5, 14.5, 18.4, 21.5, 25.2, 26.5, 23.3, 18.3,
13.9, 9.6]
    }, {
        name: 'London',
        data: [3.9, 4.2, 5.7, 8.5, 11.9, 15.2, 17.0, 16.6, 14.2, 10.3, 6.6,
4.8]
    }]
});
```

FIGURE 3. A Highcharts code sample

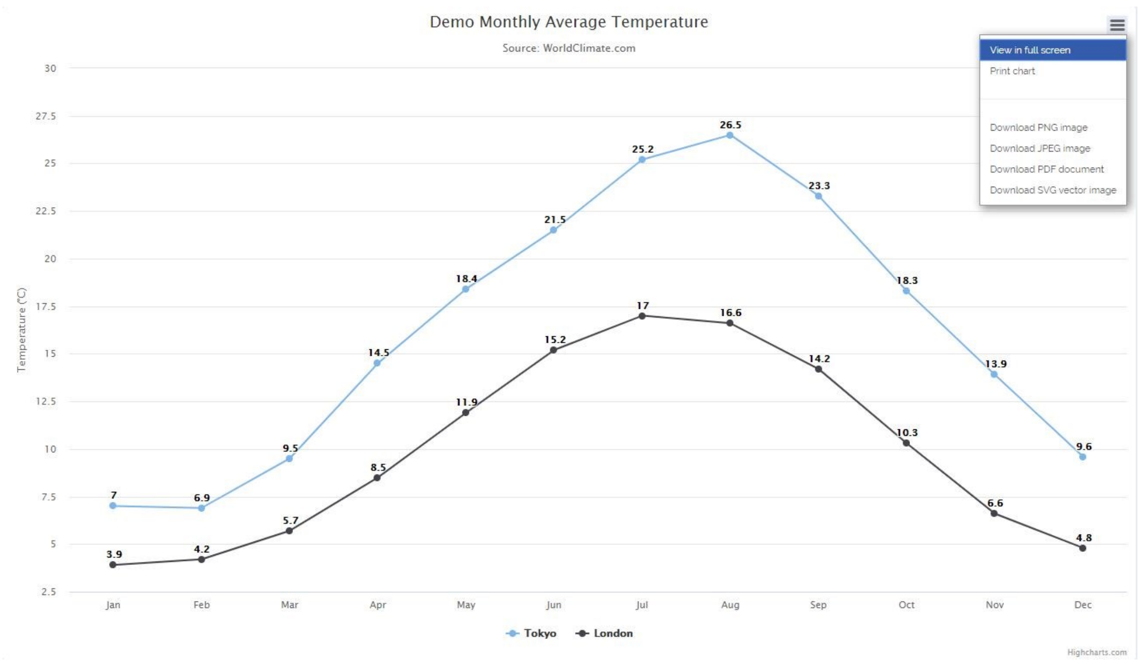


FIGURE 4. A generated sample chart by highcharts library

The code sample above (FIGURE 3.) shows how easily the code for generating a chart is shown and the chart generated by the code is presented in the chart (FIGURE 4.). The chart can be made so that the chart is responsive. The chart also offers custom features, such as exporting and printing.

2.5 Overview on Prime ng

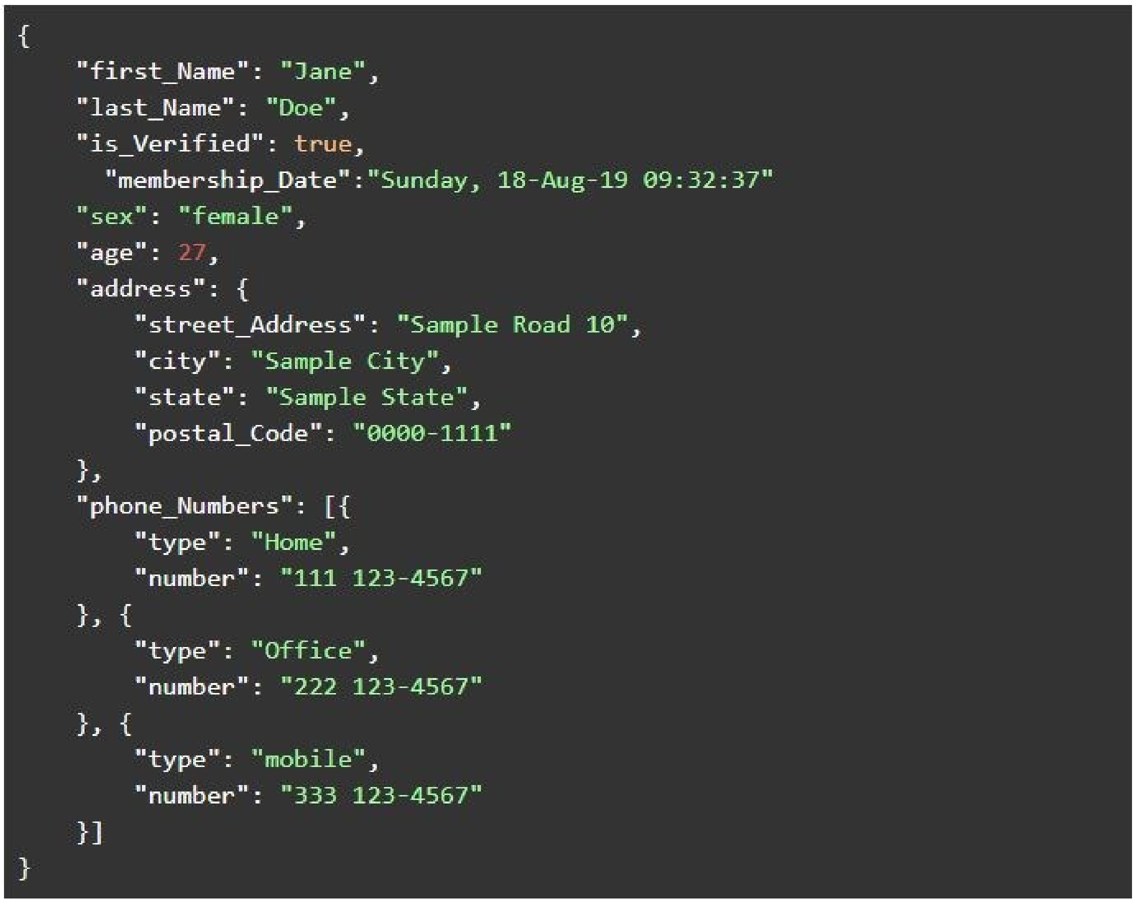
It is vital for web developers today to develop web applications that run seamlessly across iOS, Android devices, Windows, and countless web browsers they come with. This challenge requires developers to develop a user interface that can be used across all platforms while supporting the right look and feel. Making cross-platform UI components that work on all platforms is difficult and resource-consuming, therefore, using a third-party UI library is effective.

UI libraries today come with patterns that a developer can follow; these patterns can be compared to Legos. Legos are little plastic shapes that can be manipulated into a wide variety of forms and objects. The same little block of Lego can be put into multiple contexts to mean something different. It can make up part of a dinosaur or the roof of a car.

Prime ng is a UI library with rich UI components for Angular. All widgets are open source and free to use under the MIT License. Prime ng is currently as of August 2019 running on the version 8.0.2 and has other versions under development. It has a good developer support and plenty of users and global community support. Hence, Prime ng was used as a primary UI library for this project.

2.6 Overview on JSON and Protobuf

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is based on a subset of JavaScript. JSON is a text format that is completely language independent, hence it is often used to send and receive data for client-server communication and it is applicable for the project.

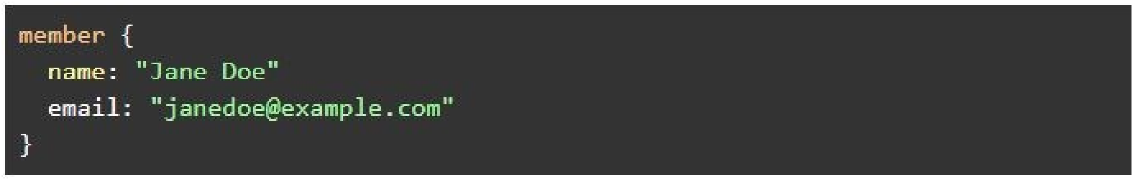


```
{
    "first_Name": "Jane",
    "last_Name": "Doe",
    "is_Verified": true,
      "membership_Date":"Sunday, 18-Aug-19 09:32:37"
    "sex": "female",
    "age": 27,
    "address": {
        "street_Address": "Sample Road 10",
        "city": "Sample City",
        "state": "Sample State",
        "postal_Code": "0000-1111"
    },
    "phone_Numbers": [{
        "type": "Home",
        "number": "111 123-4567"
    }, {
        "type": "Office",
        "number": "222 123-4567"
    }, {
        "type": "mobile",
        "number": "333 123-4567"
    }]
}
```

FIGURE 5. JSON data sample

The code sample (Figure 5.) shows an example of JSON. JSON defines only two data structures: objects and arrays. An object is a set of name-value pairs while an array is a list of values. JSON defines several value types: string, number, object, array, Boolean and null.

Like JSON, Protobuf (Protocol buffers) are language-neutral and platform-neutral data interchangers. In Protobuf the data is in a serialized structured. Serialization is the way of translating data structures or object state into a format that can be stored, transmitted, and reconstructed again. Google describes Protobuf as “You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.” (1)



```
member {
  name: "Jane Doe"
  email: "janedoe@example.com"
}
```

FIGURE 6. A Protobuf data sample

The code sample above (FIGURE 6.) shows a simple code sample of a Protobuf. The Protobuf is transferred as binary data and reconstructed again later. Sending and receiving in binary is much faster and efficient.

2.7 Overview on RxJS

RxJS (Reactive Extensions for JavaScript) is a library for reactive programming. Reactive programming is a programming way where the program reacts to data changes and self-updates whenever there is a change in data or user events.

In Reactive programming, data flows emitted by one component and the structure provided by the Rx libraries such as RxJS will propagate those changes to another component, which receives those data changes and react accordingly. Using RxJS, developers represent asynchronous data streams with Observables, query asynchronous data streams using many operators, and parameterize the concurrency in the asynchronous data streams using Schedulers. Simply put, RxJS = Observables + Operators + Schedulers. (2)

Observable: represents the idea of an invoke collection of future values or events, they listen to changes.

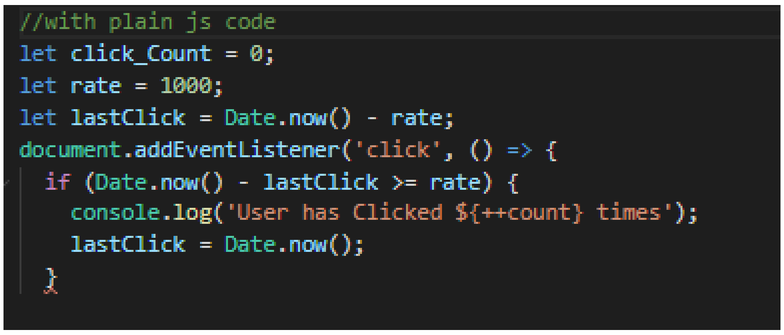
Observer: is a collection of callbacks that knows how to listen to values delivered by the Observable.

Subscription: represents the execution of an Observable, is primarily useful for canceling the execution.

Operators: are pure functions that enable a functional programming style of dealing with collections with operations such as map, filter, etc.

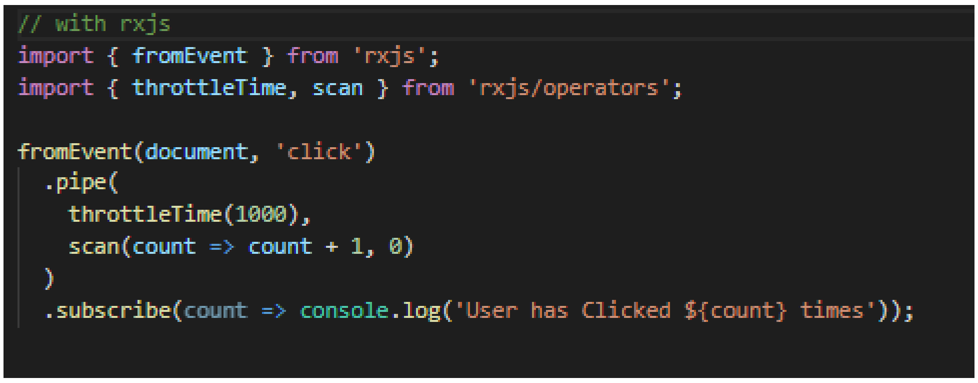
Subject: is the equivalent to an Event Emitter, and the only way of multicasting a value or event to multiple Observers.

Schedulers: are centralized dispatchers to control concurrency, allowing users to coordinate when computation happens on, e.g. “setTimeout()” (3)



```
//with plain js code
let click_Count = 0;
let rate = 1000;
let lastClick = Date.now() - rate;
document.addEventListener('click', () => {
  if (Date.now() - lastClick >= rate) {
    console.log('User has Clicked ${++count} times');
    lastClick = Date.now();
  }
```

FIGURE 7. Sample code to count user clicks with plain javascript



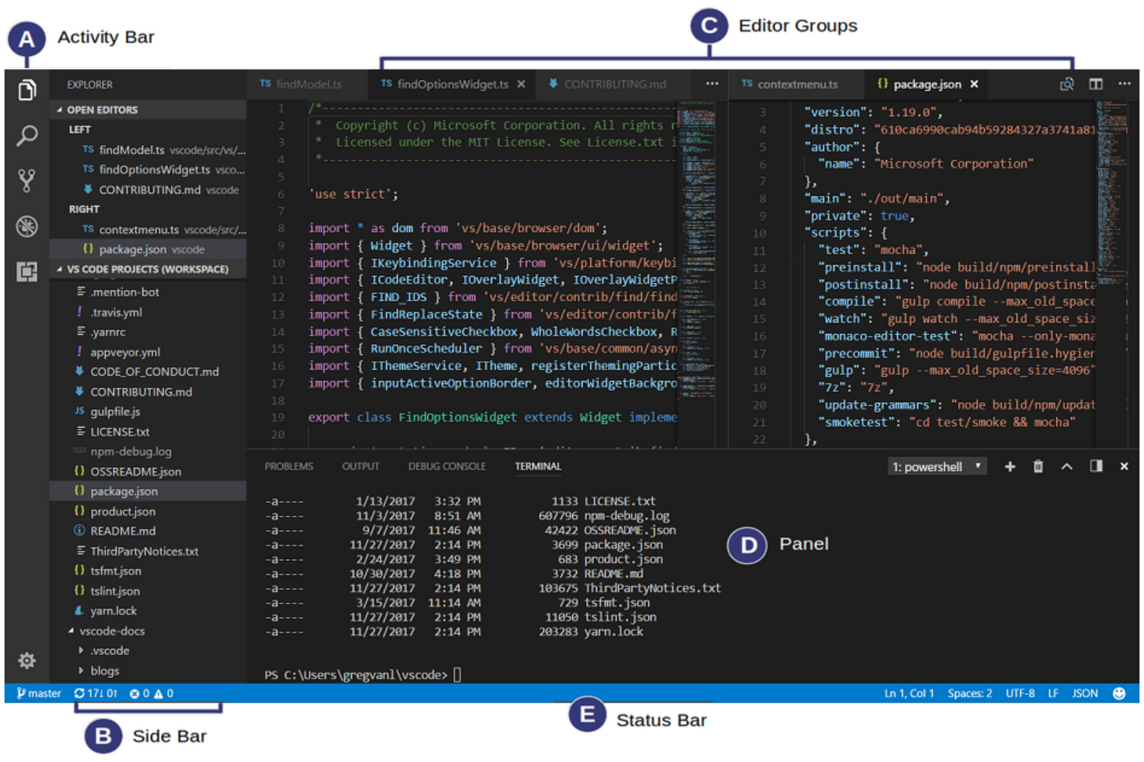
```
// with rxjs
import { fromEvent } from 'rxjs';
import { throttleTime, scan } from 'rxjs/operators';

fromEvent(document, 'click')
  .pipe(
    throttleTime(1000),
    scan(count => count + 1, 0)
  )
  .subscribe(count => console.log('User has Clicked ${count} times'));
```

FIGURE 8. Sample code to count user clicks with rxjs

The code sample (FIGURE 7.) shows a sample code to count user clicks written in plain JavaScript and a similar functioning code sample is also written with the RxJs library and presented in code sample (FIGURE 8.).

2.8 Overview of Visual Studio Code

Visual Studio Code is a code editor developed by Microsoft for Windows, Linux, and macOS. VS code has a good support for debugging, embedding version control tools, such as GitHub, and it comes with an easier syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also highly customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add added functionality. The source code for VS code is free and open-source and released under the permissive MIT License. The UI of VS code is familiar, and it is also popular with the members of the development team in Profilence. Hence, the VS code was used as the primary code editor for this project.



```
/*---------------------------------------------------------------
 *  Copyright (c) Microsoft Corporation. All rights r
 *  Licensed under the MIT License. See License.txt i
 *---------------------------------------------------------------

'use strict';

import * as dom from 'vs/base/browser/dom';
import { Widget } from 'vs/base/browser/ui/widget';
import { IKeybindingService } from 'vs/platform/keybi
import { ICodeEditor, IOverlayWidget, IOverlayWidgetP
import { FIND_IDS } from 'vs/editor/contrib/find/find
import { FindReplaceState } from 'vs/editor/contrib/f
import { CaseSensitiveCheckbox, WholeWordsCheckbox, R
import { RunOnceScheduler } from 'vs/base/common/asyn
import { IThemeService, ITheme, registerThemingPartic
import { inputActiveOptionBorder, editorWidgetBackgro

export class FindOptionsWidget extends Widget impleme
```

```
  "version": "1.19.0",
  "distro": "610ca6990cab94b59284327a3741a81
  "author": {
    "name": "Microsoft Corporation"
  },
  "main": "./out/main",
  "private": true,
  "scripts": {
    "test": "mocha",
    "preinstall": "node build/npm/preinstall
    "postinstall": "node build/npm/postinsta
    "compile": "gulp compile --max_old_space
    "watch": "gulp watch --max_old_space_siz
    "monaco-editor-test": "mocha --only-mona
    "precommit": "node build/gulpfile.hygier
    "gulp": "gulp --max_old_space_size=4096"
    "7z": "7z",
    "update-grammars": "node build/npm/updat
    "smoketest": "cd test/smoke && mocha"
  },
```

```
-a----        1/13/2017   3:32 PM           1133 LICENSE.txt
-a----        11/3/2017   8:51 AM         607796 npm-debug.log
-a----         9/7/2017  11:46 AM          42422 OSSREADME.json
-a----       11/27/2017   2:14 PM           3699 package.json
-a----        2/24/2017   3:49 PM            683 product.json
-a----       10/30/2017   4:18 PM           3732 README.md
-a----       11/27/2017   2:14 PM         103675 ThirdPartyNotices.txt
-a----        3/15/2017  11:14 AM            729 tsfmt.json
-a----       11/27/2017   2:14 PM          11050 tslint.json
-a----       11/27/2017   2:14 PM         203283 yarn.lock

PS C:\Users\gregvanl\vscode> []
```

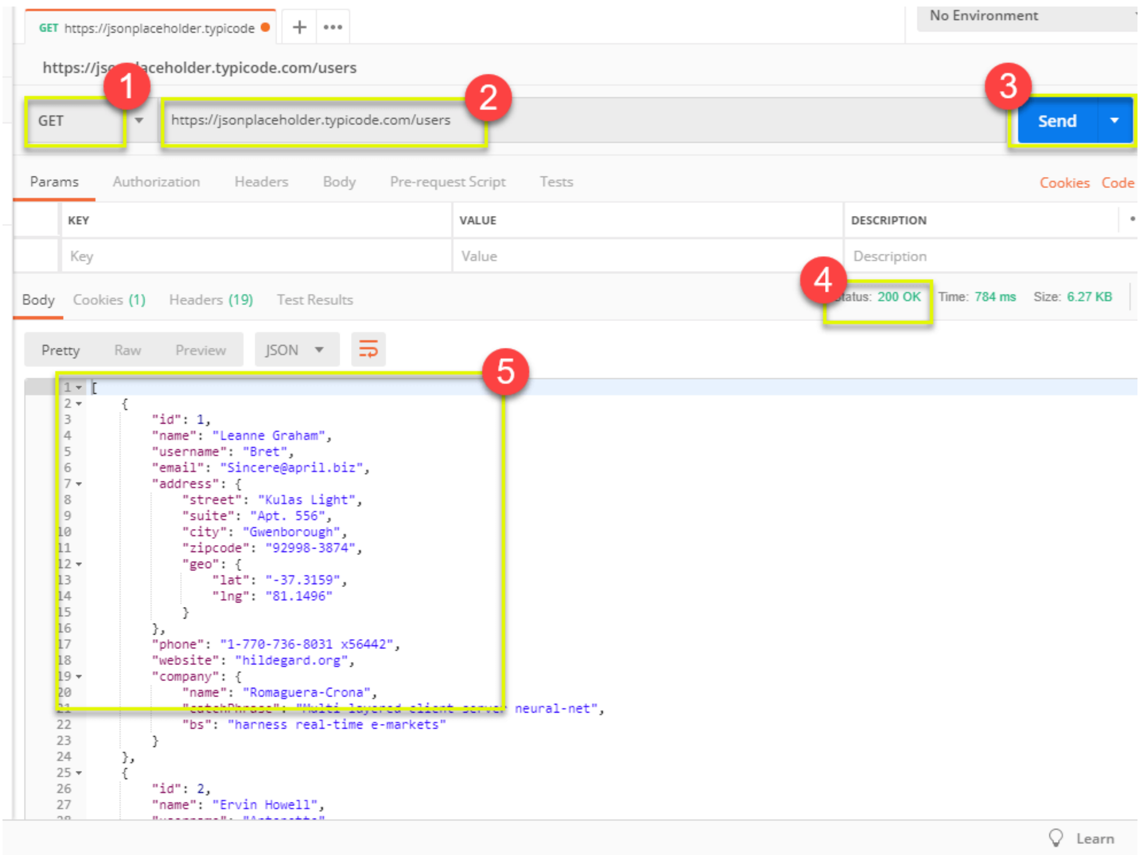
FIGURE 9. The Visual Studio Code UI (6)

The image above (FIGURE 9.) shows UI for visual studio code. The UI has various user-customizable components and can be tailored to meet the specification of a specific programming language.

2.9 Overview on Postman

Postman is a development tool used to get results from RESTful APIs. It offers a user-friendly user interface which makes HTML requests easily, without having to deal with the hassle of writing a bunch of code just to test the functionality of an API.

For instance, if a GET request to get was to be made to get Twitter feeds from twitter with all Tweets made with a hashtag *#Oulu*. To test a GET request against this API without using Postman, would take a lot of time just to set the environment and then writing code which would make the API request, but with Postman, such a test is much more streamlined and easily done. The image below (FIGURE 10.) shows a screenshot of the UI for Postman.



```
[
    {
        "id": 1,
        "name": "Leanne Graham",
        "username": "Bret",
        "email": "Sincere@april.biz",
        "address": {
            "street": "Kulas Light",
            "suite": "Apt. 556",
            "city": "Gwenborough",
            "zipcode": "92998-3874",
            "geo": {
                "lat": "-37.3159",
                "lng": "81.1496"
            }
        },
        "phone": "1-770-736-8031 x56442",
        "website": "hildegard.org",
        "company": {
            "name": "Romaguera-Crona",
            "catchPhrase": "Multi-layered client-server neural-net",
            "bs": "harness real-time e-markets"
        }
    },
    {
        "id": 2,
        "name": "Ervin Howell",
```

FIGURE 10.The Postman UI (3)

1. Set your HTTP request to GET.
2. In the request URL field, input link
3. Click Send
4. You will see 200 OK Message
5. There should be 10 user results in the body which indicates that your

(3)

3 UI DEVELOPMENT

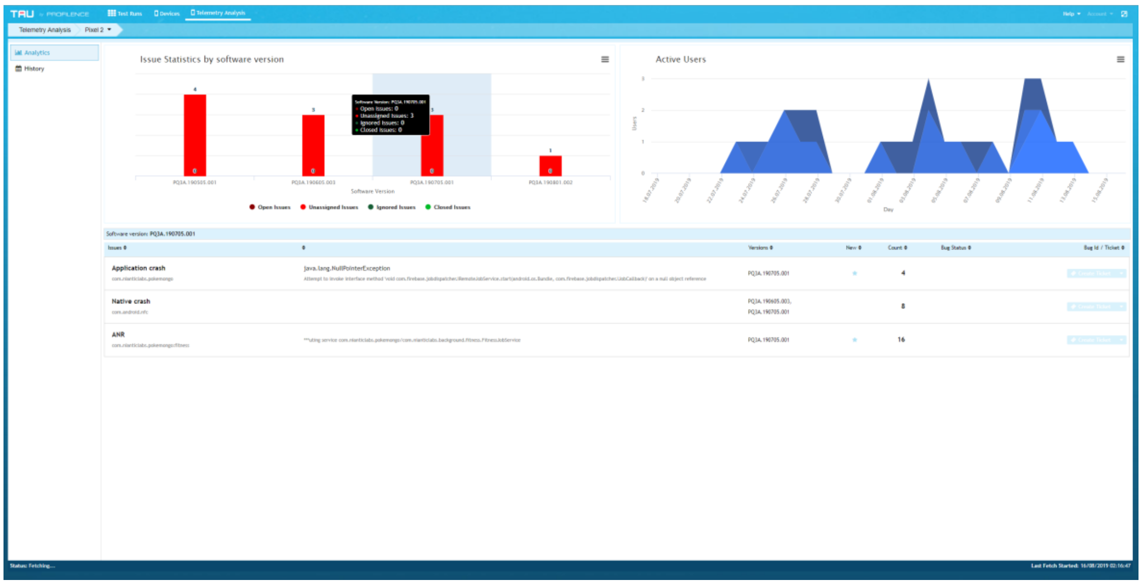
The UI development in the project was focused on simplifying the data as much as possible. Getting meaningful data with minimal information, yet having all the necessary information was the main goal.

The best way to represent data is by graphs and charts. There is magic in graphs. The profile of a curve reveals in a flash a whole situation -the life history of an epidemic, a panic, or an era of prosperity. The curve informs the mind, awakens the imagination and convinces. (4)

The UI was designed in such that results are separated into 3 main views: Telemetry, Test runs analysis and Devices.

3.1 UI development for Telemetry

The telemetry view has the reports for the software versions of devices that have already been deployed and are under testing by a testing team. The devices have an Android app which collects log data running as a background application on an Android device and it sends the crash and log data to the remote data server.



Software version: PQ3A.190705.001

Issues		Versions	New	Count	Bug Status	Bug Id / Ticket
Application crash com.nianticlabs.pokemongo	java.lang.NullPointerException Attempt to invoke interface method 'void com.firebase.jobdispatcher.IRemoteJobService.start(android.os.Bundle, com.firebase.jobdispatcher.IJobCallback)' on a null object reference	PQ3A.190705.001	★	4		
Native crash com.android.nfc		PQ3A.190605.003, PQ3A.190705.001		8		
ANR com.nianticlabs.pokemongo:fitness	**uting service com.nianticlabs.pokemongo/com.nianticlabs.background.fitness.FitnessJobService	PQ3A.190705.001	★	16		

FIGURE 11. The Telemetry view

The Figure 11 shows a UI for the Telemetry Analysis. Telemetry currently contains two tabs, which contains analytics and history. The charts in this view were made by using the Highcharts library.

The charts in the UI along with the issue list shown on the image (Figure 11.) update as soon as there is a new issue reported and logged on the server, and the changes are reflected in real-time. To achieve such real-time data processing, RxJS was used. RxJS code listens to the server changes continuously and passes the data as soon as there is a change and the changed data triggers the highcharts to display the changes.

3.1.1 UI development for analytics

The analytics view on the telemetry view shows the analytics of the device. The data retrieved from the backend was processed and shown as:

Issue Statistics by software version

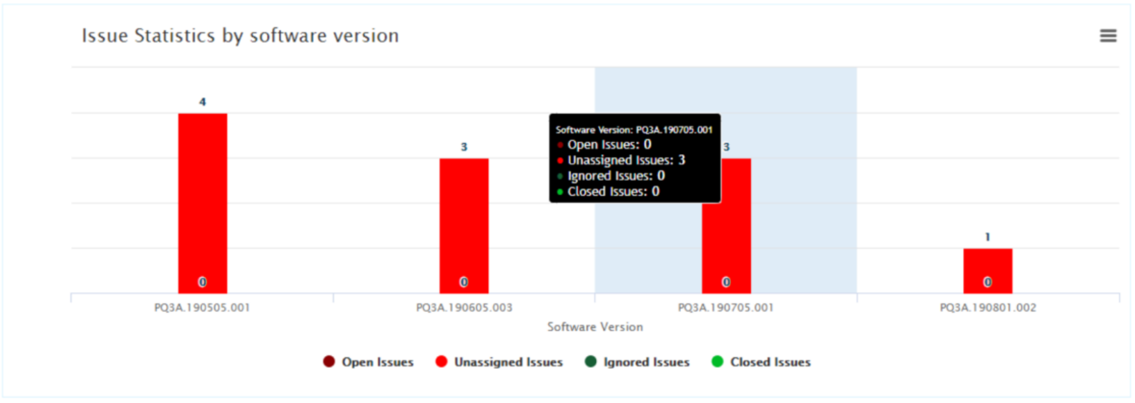
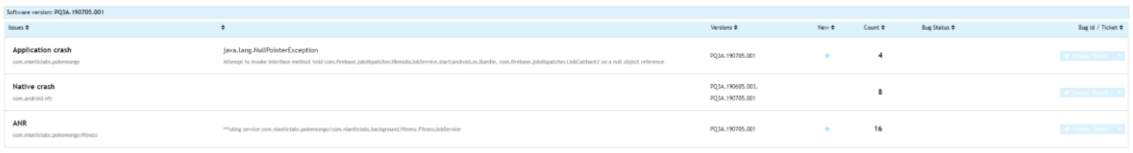


FIGURE 12. Issue Statistics by software version

The Bar chart (Figure 12.) contains a chart that shows analytics on the issues categorized under Open, Unassigned, Ignored and Closed. The issues are color-coded for convenience of the viewer on the bar chart and categorized per software version. In the figure 12. the current software version selected is “PQ3A.190705.001” and it has only three unassigned issues. Those unassigned issues are critical and need a further examination and need an immediate examination and need to be assigned to a developer and hence color-coded as red.

Issues list



Software version: PQ3A.190705.001

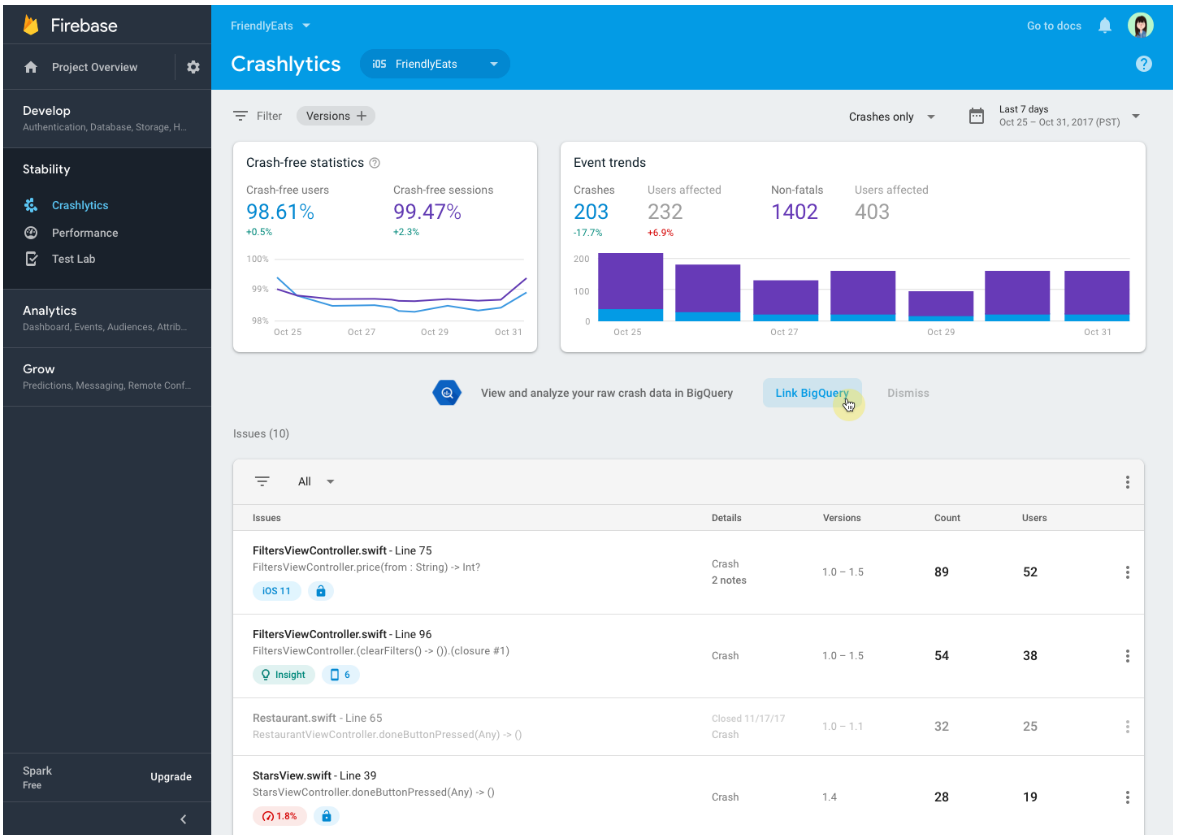
Issues		Versions	New	Count	Bug Status	Bug Id / Ticket
Application crash com.nianticlabs.pokemongo	java.lang.NullPointerException Attempt to invoke interface method 'void com.firebase.jobdispatcher.IRemoteJobService.start(android.os.Bundle, com.firebase.jobdispatcher.IJobCallback)' on a null object reference	PQ3A.190705.001	★	4		Create Ticket
Native crash com.android.nfc		PQ3A.190605.003, PQ3A.190705.001		8		Create Ticket
ANR com.nianticlabs.pokemongo:fitness	***uting service com.nianticlabs.pokemongo/com.nianticlabs.background.fitness.FitnessJobService	PQ3A.190705.001	★	16		Create Ticket

FIGURE 13. The Issue List

The Figure 13 shows the issue list for the software version selected on the chart (Figure 12.) the list contains details of the issue and the process with which the issue is seen on. The list also shows whether the detected issue is a new issue and if so, it is marked with a star. Contemporary issues signify a new bug which was not detected in any of the previously tested versions.

The number of issue instances are also shown on the table. Issues can be clicked and navigated further to get more details on the issues.

It was difficult to display the issue list since displaying descriptive information about the issue without overwhelming the user with unnecessary data was important. With further study of the data and a few meetings with the client and refines with the Firebase Crashlytics UI, this view was optimized. The Figure 14. shows the reference that was used for the issue list data display.



Crash-free statistics

Crash-free users	Crash-free sessions
98.61%	99.47%
+0.5%	+2.3%

Event trends

Crashes	Users affected	Non-fatals	Users affected
203	232	1402	403
-17.7%	+6.9%		

Issues (10)

Issues	Details	Versions	Count	Users
FiltersViewController.swift - Line 75 FiltersViewController.price(from : String) -> Int? iOS 11	Crash 2 notes	1.0 – 1.5	89	52
FiltersViewController.swift - Line 96 FiltersViewController.(clearFilters() -> ()).(closure #1) Insight 6	Crash	1.0 – 1.5	54	38
Restaurant.swift - Line 65 RestaurantViewController.doneButtonPressed(Any) -> ()	Closed 11/17/17 Crash	1.0 – 1.1	32	25
StarsView.swift - Line 39 StarsViewController.doneButtonPressed(Any) -> () 1.8%	Crash	1.4	28	19

FIGURE 14. The Firebase Crashlytics UI (7)

Active users list

The active users list shows the area chart of the active users who are evaluating the device for a specific version. This chart shows an easy analytical view on whether a software version has undergone a test on a specific date. The chart also shows how many users are actively using a specific test version for testing.

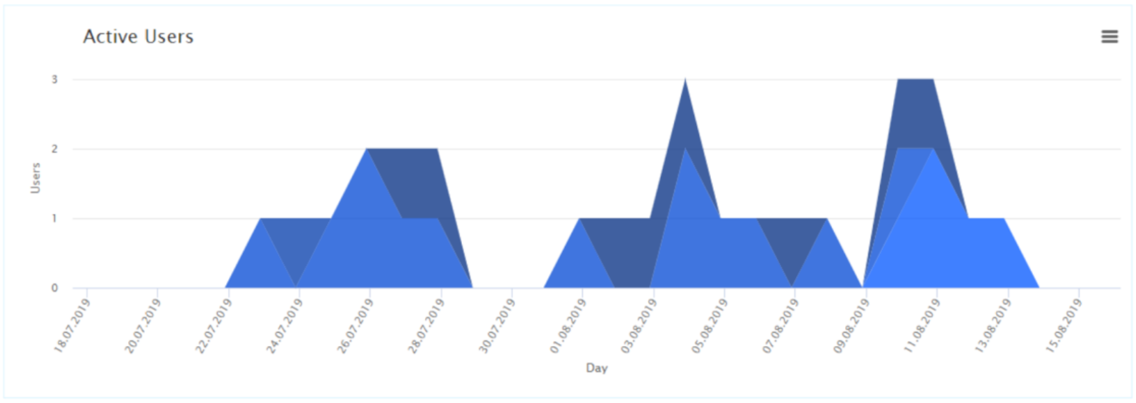
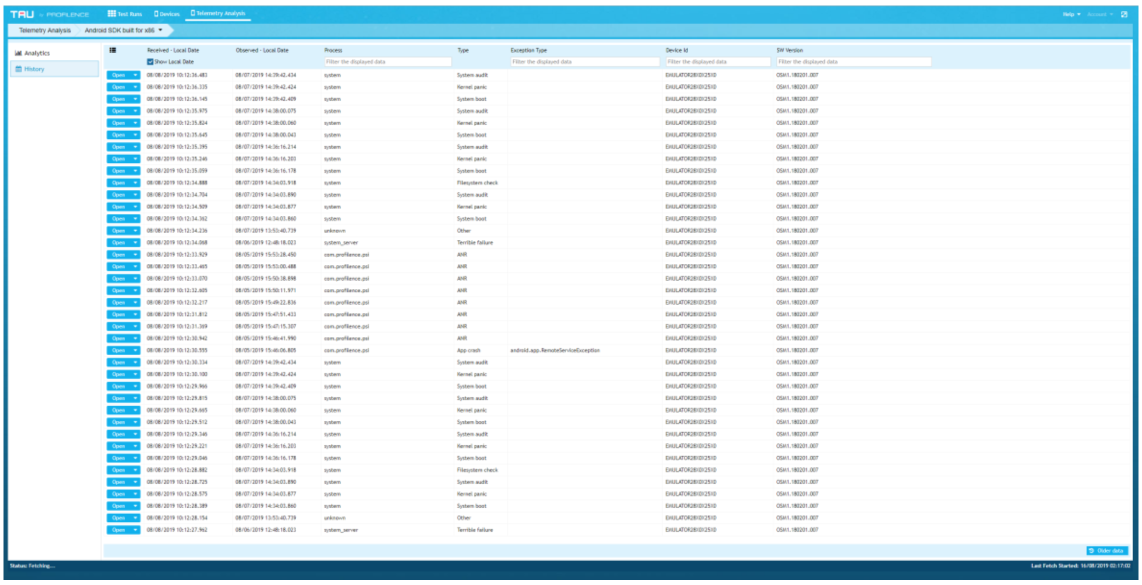


FIGURE 15. The Active users' chart

The area chart (Figure 15.) shows an area chart for the active users for the selected device on the chart. (Figure 12.)

3.1.2 UI development for history

The History view on the telemetry analysis shows the history log for all the selected test devices. As seen on Figure 16. below, an option for the user to localize the date-time row is also available. This feature tackles the problem where a test device is from a different time zone. The table is filterable, and users can choose to focus on only a certain software version, process or device ID.



	Received - Local Date	Observed - Local Date	Process	Type	Exception Type	Device Id	SW Version
Open	08/08/2019 10:12:36.483	08/07/2019 14:39:42.434	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:36.335	08/07/2019 14:39:42.424	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:36.145	08/07/2019 14:39:42.409	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.975	08/07/2019 14:38:00.075	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.824	08/07/2019 14:38:00.060	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.645	08/07/2019 14:38:00.043	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.395	08/07/2019 14:36:16.214	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.246	08/07/2019 14:36:16.203	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:35.059	08/07/2019 14:36:16.178	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.888	08/07/2019 14:34:03.918	system	Filesystem check		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.704	08/07/2019 14:34:03.890	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.509	08/07/2019 14:34:03.877	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.362	08/07/2019 14:34:03.860	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.236	08/07/2019 13:53:40.739	unknown	Other		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:34.068	08/06/2019 12:48:18.023	system_server	Terrible failure		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:33.929	08/05/2019 15:53:28.450	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:33.465	08/05/2019 15:53:00.488	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:33.070	08/05/2019 15:50:38.898	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:32.605	08/05/2019 15:50:11.971	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:32.217	08/05/2019 15:49:22.836	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:31.812	08/05/2019 15:47:51.433	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:31.369	08/05/2019 15:47:15.307	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:30.942	08/05/2019 15:46:41.990	com.profilence.psi	ANR		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:30.555	08/05/2019 15:46:06.805	com.profilence.psi	App crash	android.app.RemoteServiceException	EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:30.334	08/07/2019 14:39:42.434	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:30.100	08/07/2019 14:39:42.424	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.966	08/07/2019 14:39:42.409	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.815	08/07/2019 14:38:00.075	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.665	08/07/2019 14:38:00.060	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.512	08/07/2019 14:38:00.043	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.346	08/07/2019 14:36:16.214	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.221	08/07/2019 14:36:16.203	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:29.046	08/07/2019 14:36:16.178	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:28.882	08/07/2019 14:34:03.918	system	Filesystem check		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:28.725	08/07/2019 14:34:03.890	system	System audit		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:28.575	08/07/2019 14:34:03.877	system	Kernel panic		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:28.389	08/07/2019 14:34:03.860	system	System boot		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:28.154	08/07/2019 13:53:40.739	unknown	Other		EMULATOR28X0X25X0	OSM1.180201.007
Open	08/08/2019 10:12:27.962	08/06/2019 12:48:18.023	system_server	Terrible failure		EMULATOR28X0X25X0	OSM1.180201.007

Status: Fetching...

Last Fetch Started: 16/08/2019 02:17:02

FIGURE 16. The History View for telemetry analysis

3.2 UI development for Test runs

The Test runs view contains the active test runs ongoing on the test station. One or more devices are connected to a test server and a series of test cases are running on the device to collect logs of data. A test run can run ranging from a few hours to a couple of hundred hours. The data collected is real-time and this real-time data reflects on the test run view and analytics. The data coming from the server is handed as JSON and is parsed into objects and arrays.

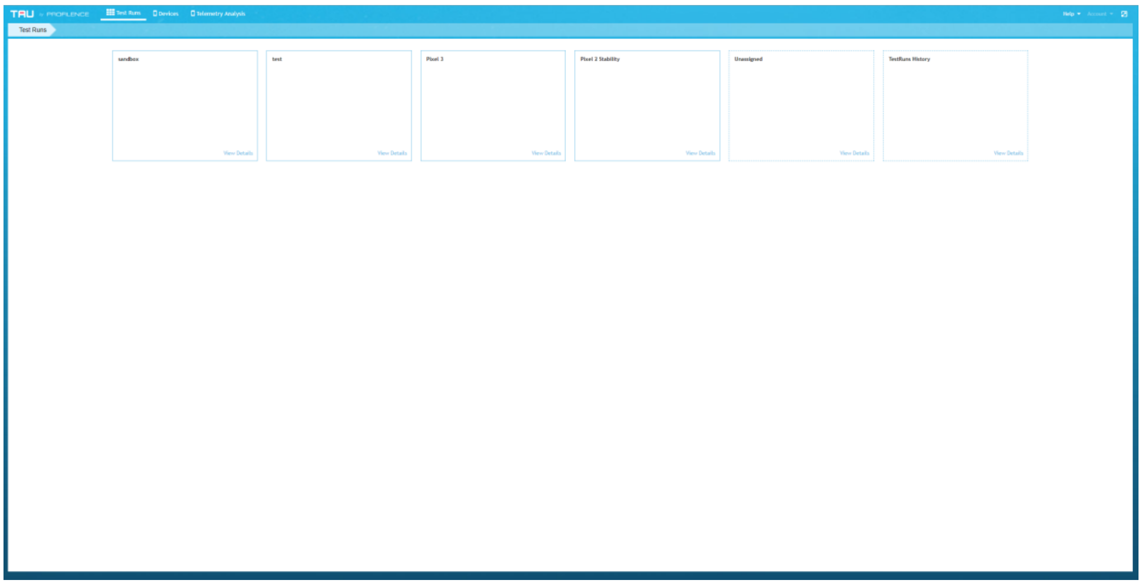
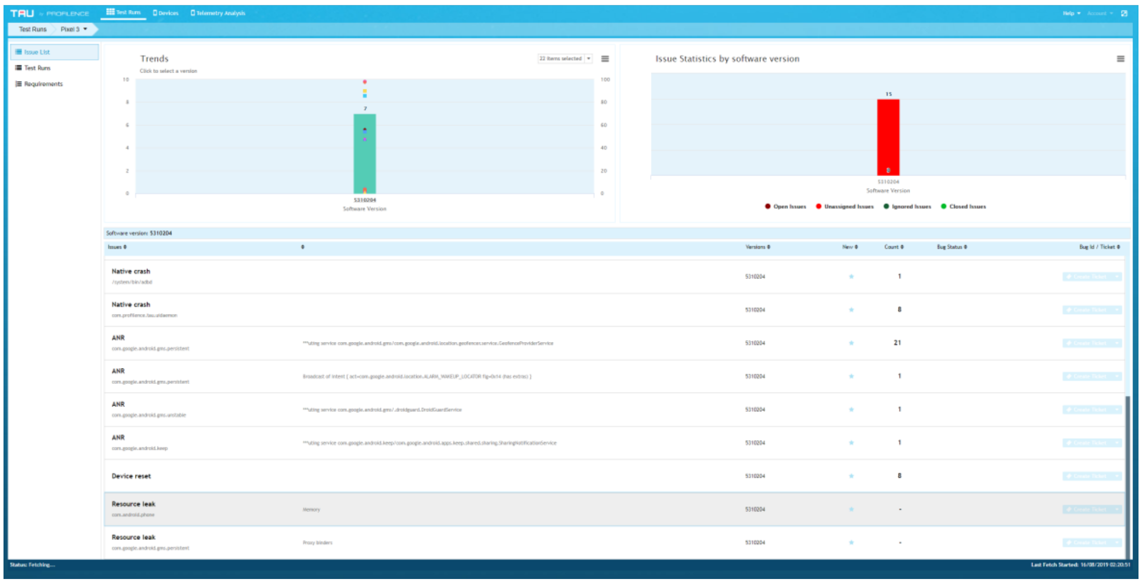


FIGURE 17. The Test run view

The Figure 17 shows a screenshot of a test run analysis view. This view contains a clear view of an ongoing project. The projects may be spread across several test stations and on a remote server. The projects are marked with a solid box and dotted boxes to specify if a project has a categorized project name.

3.2.1 UI development for Issue list

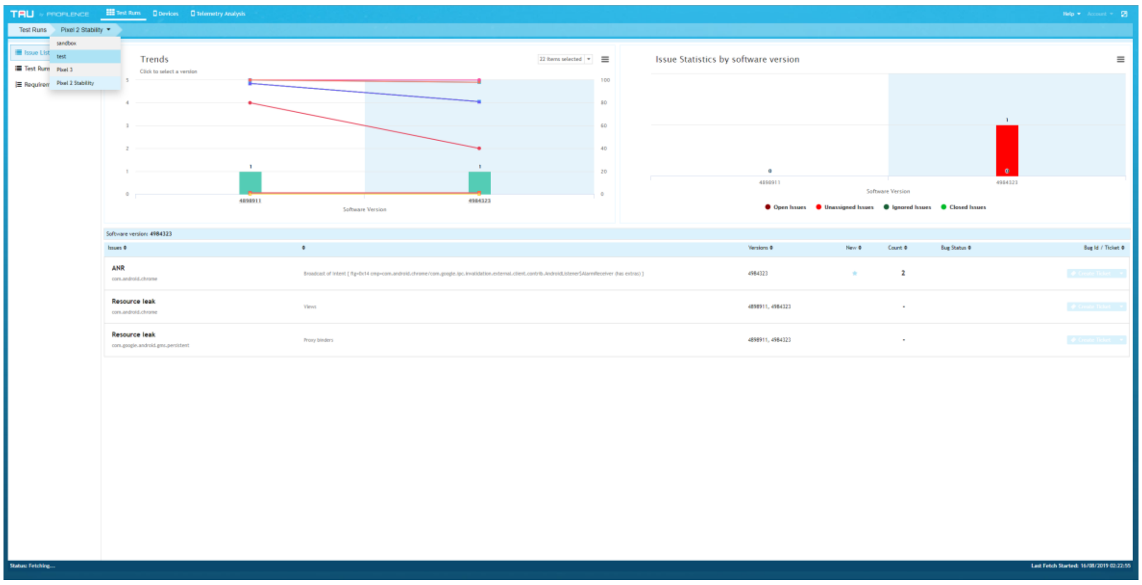


Software version: 5310204

Issues		Versions	New	Count	Bug Status	Bug Id / Ticket
Native crash /system/bin/adbd		5310204	★	1		
Native crash com.profilence.tau.uidaemon		5310204	★	8		
ANR com.google.android.gms.persistent	**uting service com.google.android.gms/com.google.android.location.geofencer.service.GeofenceProviderService	5310204	★	21		
ANR com.google.android.gms.persistent	Broadcast of intent { act=com.google.android.location.ALARM_WAKEUP_LOCATOR flg=0x14 (has extras) }	5310204	★	1		
ANR com.google.android.gms.unstable	**uting service com.google.android.gms/.droidguard.DroidGuardService	5310204	★	1		
ANR com.google.android.keep	**uting service com.google.android.keep/com.google.android.apps.keep.shared.sharing.SharingNotificationService	5310204	★	1		
Device reset		5310204	★	8		
Resource leak com.android.phone	Memory	5310204	★	-		
Resource leak com.google.android.gms.persistent	Proxy binders	5310204	★	-		

FIGURE 18. The Issue list view

The Figure 18 shows mainly two graphs, namely trends and issue statistics by software version. The two graphs are interconnected, upon the selection of a software version on the “Trends” chart, triggers the view to change on the other chart. Also, it generates a table of issues on the bottom area of the page.



Software version: 4984323

Issues		Versions	New	Count	Bug Status	Bug Id / Ticket
ANR com.android.chrome	Broadcast of intent { flg=0x14 cmp=com.android.chrome/com.google.ipc.invalidation.external.client.contrib.AndroidListener$AlarmReceiver (has extras) }	4984323	★	2		
Resource leak com.android.chrome	Views	4898911, 4984323		-		
Resource leak com.google.android.gms.persistent	Proxy binders	4898911, 4984323		-		

FIGURE 19. The Issue list UI

The Figure 19. shows the UI for the issue list view. The user can toggle between the projects from the breadcrumb of the page. The breadcrumb works in a native way like it does on Windows Explorer. The dropdown on the project title makes it easier for the user to navigate to different projects quickly without having to navigate to earlier views. Also, for the ease of access to the user, each element on the page comes with tooltips to help users to access the UI elements easily.

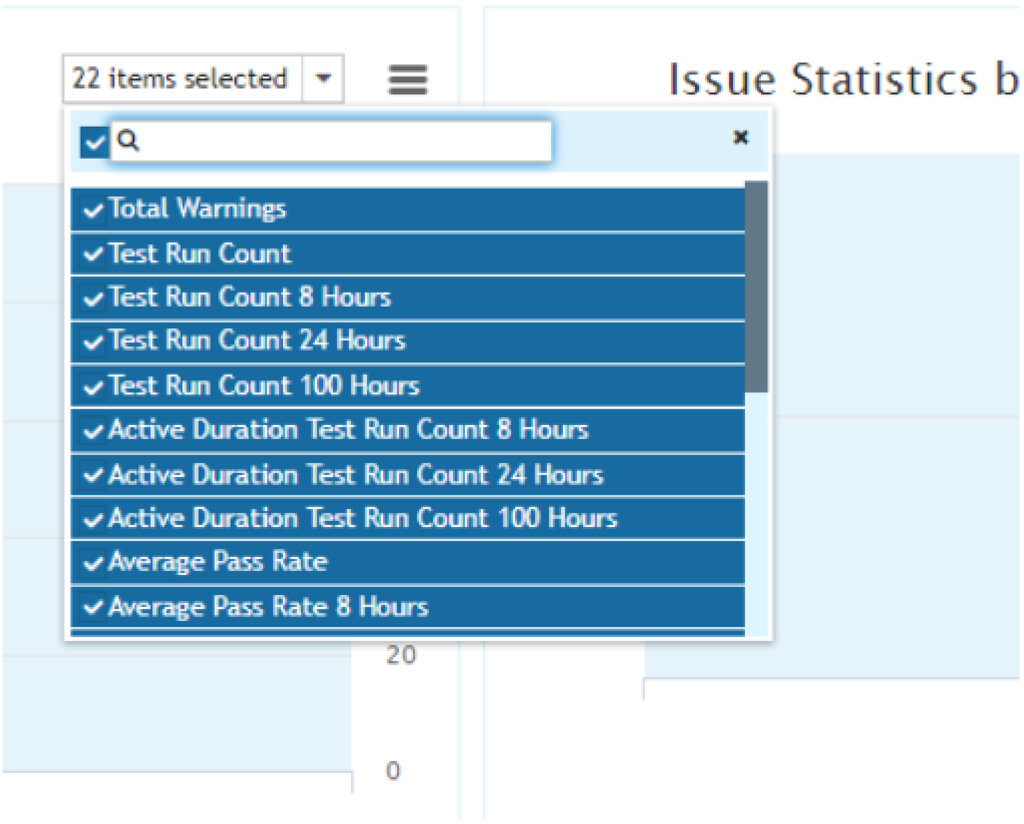


FIGURE 20. Trends chart dropdown combo box

The trends chart on the Figure 18 and Figure 19 shows the trend of device on how one particular device is doing comparatively with other software versions. The dropdown combo box comes with various metrics to compare the software version statistics as shown on Figure 20.

The issue table contains details of the issues generated by the software version. This issue list is like the issue list on the telemetry analytics. The issue list can be further studied by selecting an issue on the table.

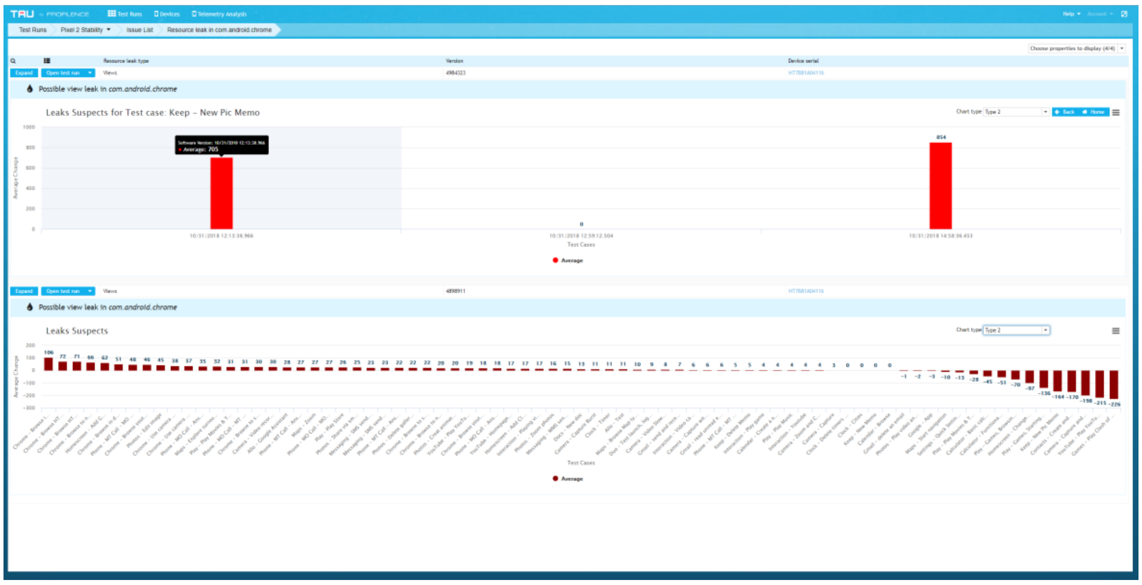

Possible view leak in com.android.chrome
Leaks Suspects for Test case: Keep – New Pic Memo
Average Change
Test Cases
Average
Leaks Suspects

FIGURE 21. Details of a particular issue

The FIGURE 21 above shows details of the memory leak that happened in the selected process, The chart on the image shows the bar chart which clearly shows when the memory leak peaked. The user can further navigate and drill down to the individual bars and get further details. The drill-down view after the user clicks on a data point is shown on FIGURE 22.

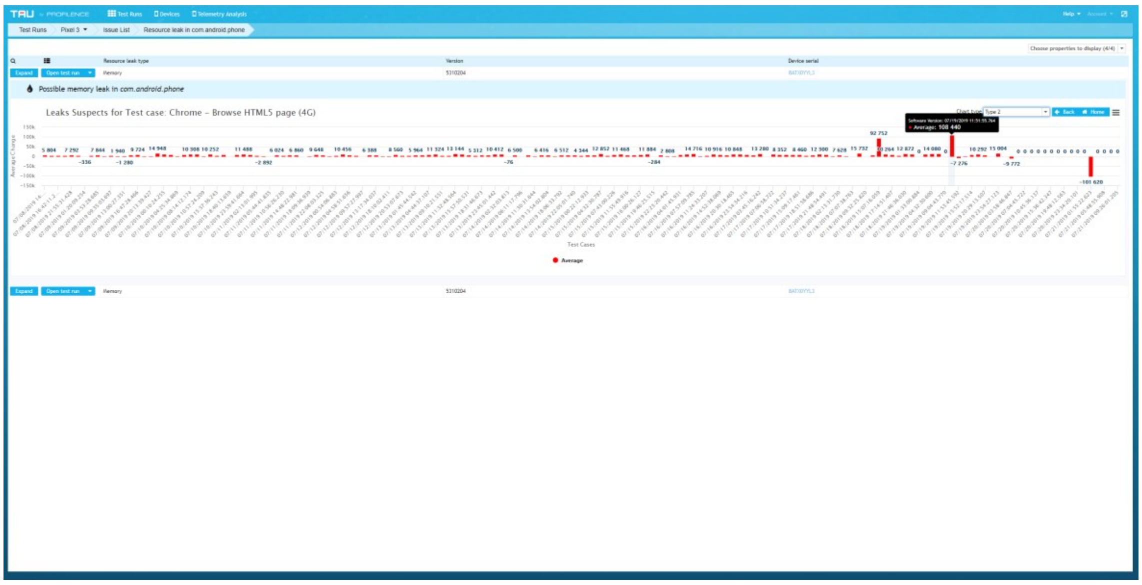
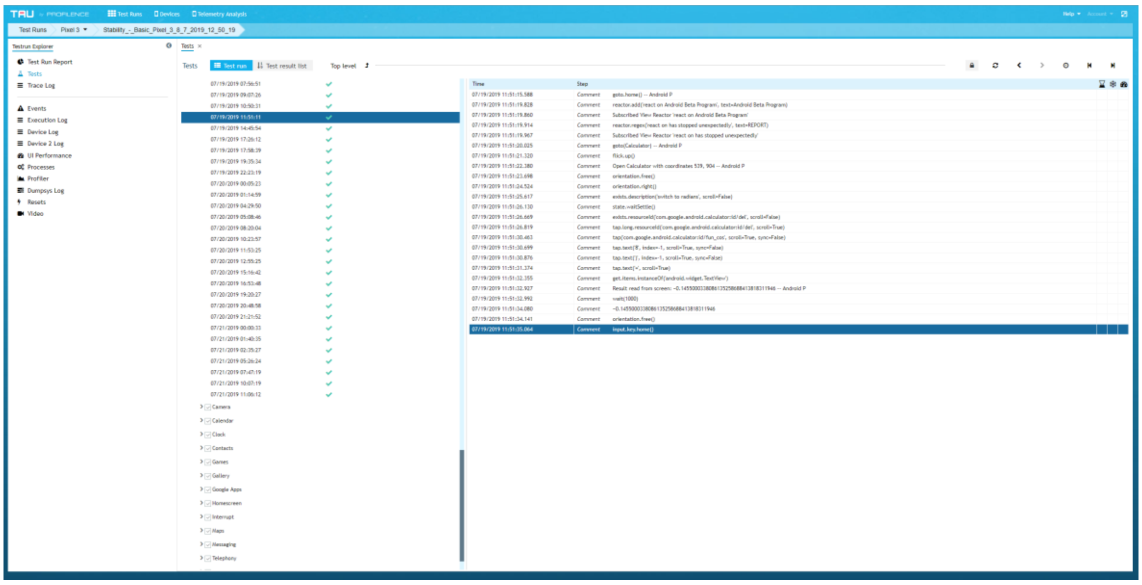

Possible memory leak in com.android.phone
Leaks Suspects for Test case: Chrome – Browse HTML5 page (4G)
Average Change
Test Cases
Average

FIGURE 22. Details of an issue



```
Time                     Step
07/19/2019 11:51:15.588  Comment  goto.home() -- Android P
07/19/2019 11:51:19.828  Comment  reactor.add('react on Android Beta Program', text=Android Beta Program)
07/19/2019 11:51:19.860  Comment  Subscribed View Reactor 'react on Android Beta Program'
07/19/2019 11:51:19.914  Comment  reactor.regex('react on has stopped unexpectedly', text=REPORT)
07/19/2019 11:51:19.967  Comment  Subscribed View Reactor 'react on has stopped unexpectedly'
07/19/2019 11:51:20.025  Comment  goto(Calculator) -- Android P
07/19/2019 11:51:21.320  Comment  flick.up()
07/19/2019 11:51:22.380  Comment  Open Calculator with coordinates 539, 904 -- Android P
07/19/2019 11:51:23.698  Comment  orientation.free()
07/19/2019 11:51:24.524  Comment  orientation.right()
07/19/2019 11:51:25.617  Comment  exists.description('switch to radians', scroll=False)
07/19/2019 11:51:26.130  Comment  state.waitSettle()
07/19/2019 11:51:26.669  Comment  exists.resourceId('com.google.android.calculator:id/del', scroll=False)
07/19/2019 11:51:26.819  Comment  tap.long.resourceId('com.google.android.calculator:id/del', scroll=True)
07/19/2019 11:51:30.463  Comment  tap('com.google.android.calculator:id/fun_cos', scroll=True, sync=False)
07/19/2019 11:51:30.699  Comment  tap.text('8', index=-1, scroll=True, sync=False)
07/19/2019 11:51:30.876  Comment  tap.text(')', index=-1, scroll=True, sync=False)
07/19/2019 11:51:31.374  Comment  tap.text('=', scroll=True)
07/19/2019 11:51:32.355  Comment  get.items.instanceOf('android.widget.TextView')
07/19/2019 11:51:32.927  Comment  Result read from screen: −0.1455000338086135258688413818311946 -- Android P
07/19/2019 11:51:32.992  Comment  wait(1000)
07/19/2019 11:51:34.080  Comment  −0.1455000338086135258688413818311946
07/19/2019 11:51:34.141  Comment  orientation.free()
07/19/2019 11:51:35.064  Comment  input.key.home()
```

FIGURE 23. Logs of particular timestamp of an issue

The Figure 23 above shows the UI for the logs of crash issues. The issues are highlighted to show the user which issue is being observed.

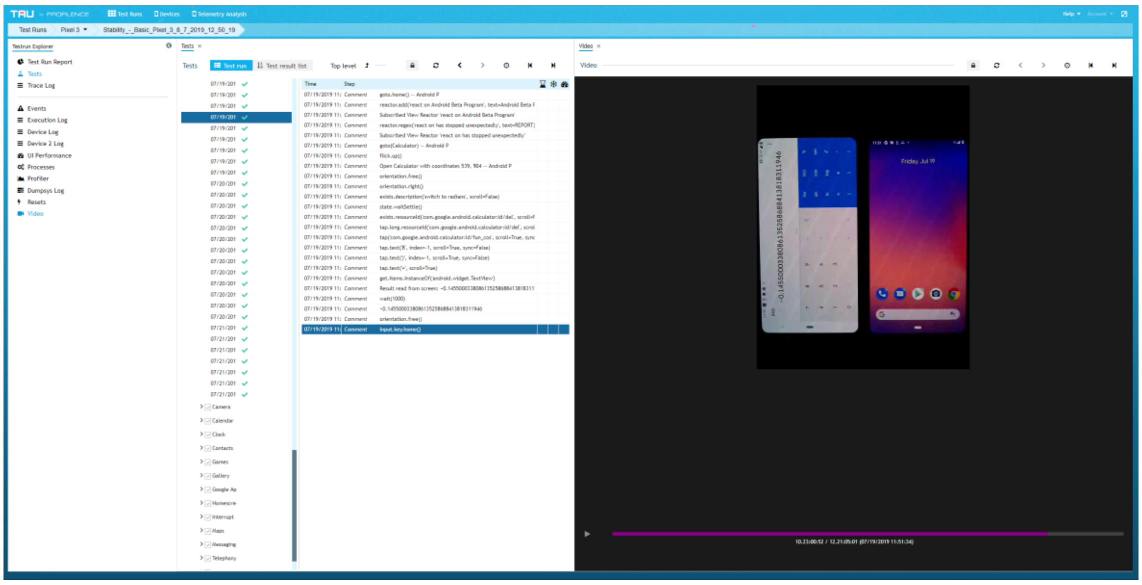
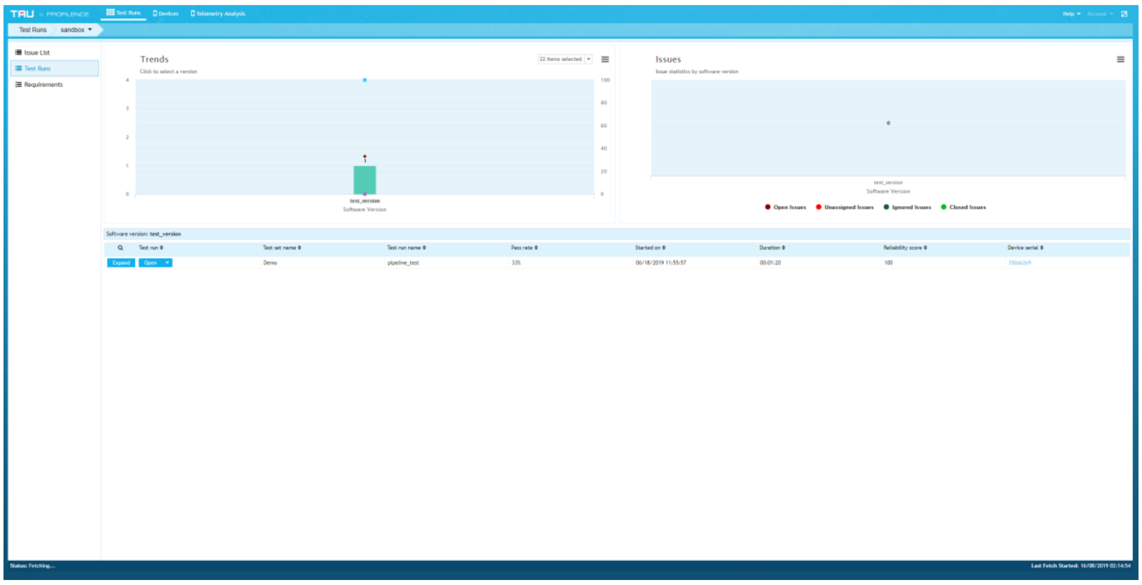


FIGURE 24. Log and video recording pointed to the particular

The Figure 24 above shows a side by side view of the recorded video and the crash log list. The user can pinpoint the crash log and see what occurred during that crash log timestamp in the video on the right. The video and the crash logs are synchronized together. Clicking on the time frame of the video or the crash log row will automatically update the timestamp of the other view.

3.2.2 UI development for test runs

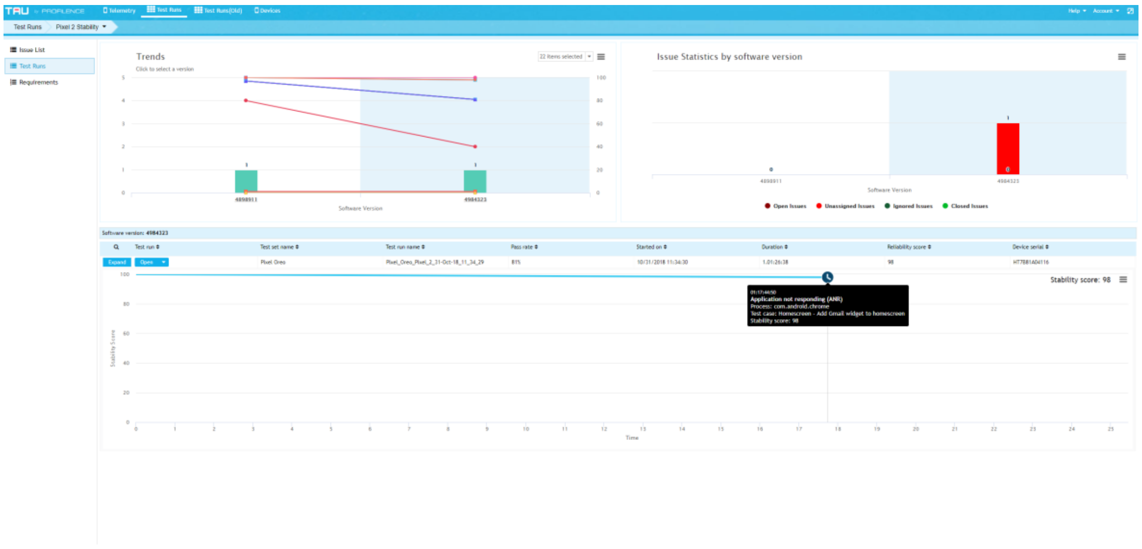
The test run view contains the test runs of the devices selected, Test runs are listed so that the user can get a quick glimpse of the stability score. This score is the rating of device test run based on how stable or unstable the device was during the test run. For instance, a device can have a score of 100 at an 80-hour period of time if there were no issues during that period of hours but as time progresses, the stability score decreases as new issues are detected.



Software version: test_version

Test run	Test set name	Test run name	Pass rate	Started on	Duration	Reliability score	Device serial
Expand Open	Demo	pipeline_test	33%	06/18/2019 11:55:57	00:01:20	100	350ab2e9

FIGURE 25. The Test run view



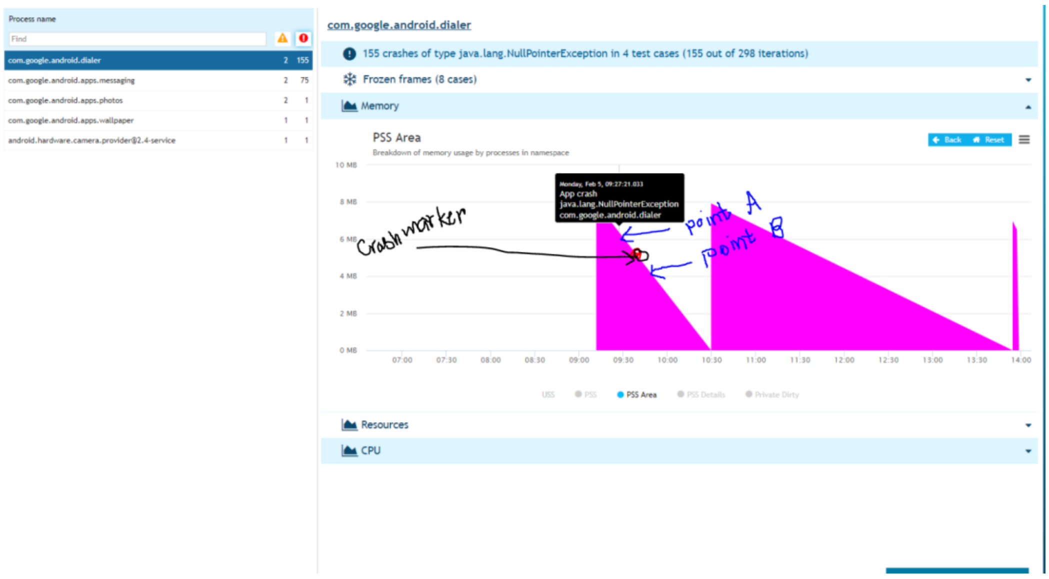
Software version: 4984323

Test run	Test set name	Test run name	Pass rate	Started on	Duration	Reliability score	Device serial
Expand Open	Pixel Oreo	Pixel_Oreo_Pixel_2_31-Oct-18_11_34_29	81%	10/31/2018 11:34:30	1.01:26:38	98	HT7881A04116

01:17:44:50
Application not responding (ANR)
Process: com.android.chrome
Test case: Homescreen - Add Gmail widget to homescreen
Stability score: 98

FIGURE 26. The Stability score view in test run view

The Figure 26 shows the test run stability score a glimpse of the test run shown on the FIGURE 25 above. The stability score can also be further inspected with a tooltip. The tooltip also allows the user to navigate deeper into the reason which shows why the stability score dropped showing what changed during those time periods.



Process name		
com.google.android.dialer	2	155
com.google.android.apps.messaging	2	75
com.google.android.apps.photos	2	1
com.google.android.apps.wallpaper	1	1
android.hardware.camera.provider@2.4-service	1	1

com.google.android.dialer

155 crashes of type java.lang.NullPointerException in 4 test cases (155 out of 298 iterations)

Frozen frames (8 cases)

Memory

PSS Area

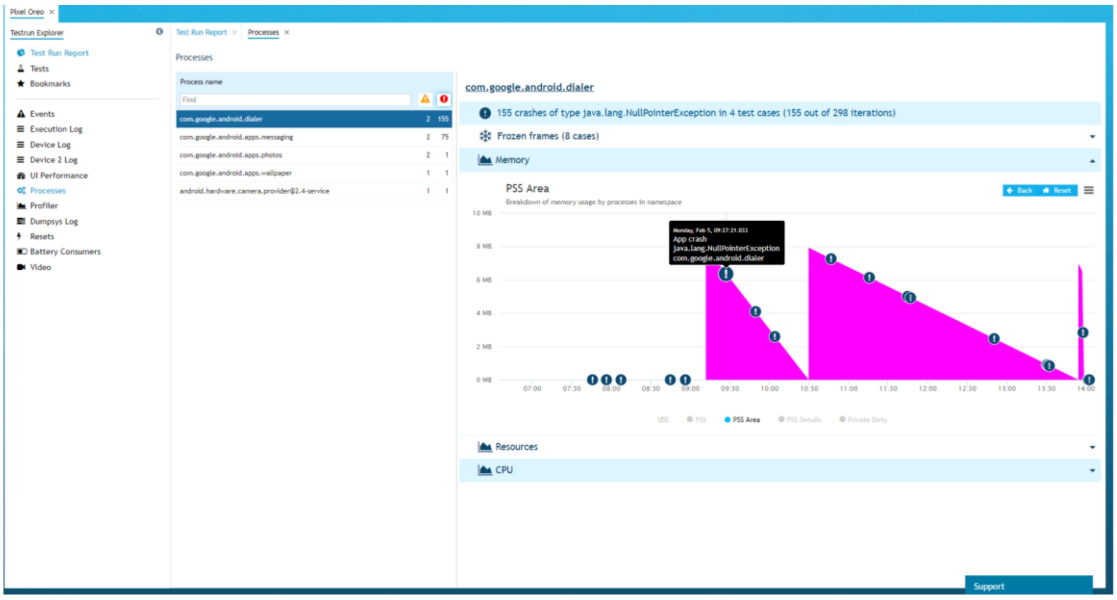
Breakdown of memory usage by processes in namespace

Monday, Feb 5, 09:27:21.033
App crash
java.lang.NullPointerException
com.google.android.dialer

Resources

CPU

FIGURE 27. The Mockup for approximation of crash point



Processes

Process name		
com.google.android.dialer	2	155
com.google.android.apps.messaging	2	75
com.google.android.apps.photos	2	1
com.google.android.apps.wallpaper	1	1
android.hardware.camera.provider@2.4-service	1	1

com.google.android.dialer

155 crashes of type java.lang.NullPointerException in 4 test cases (155 out of 298 iterations)

Frozen frames (8 cases)

Memory

PSS Area

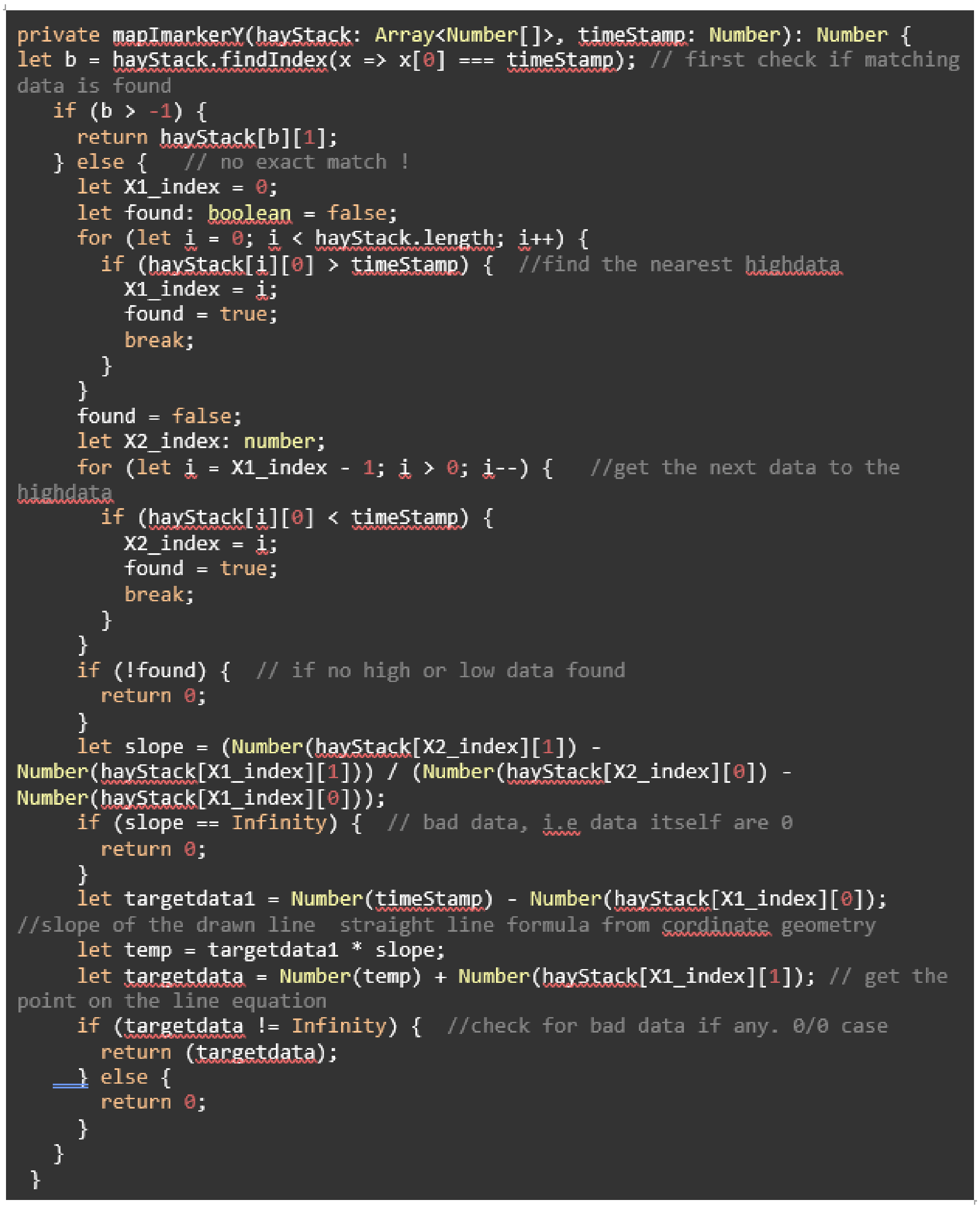
Breakdown of memory usage by processes in namespace

Monday, Feb 5, 09:27:21.033
App crash
java.lang.NullPointerException
com.google.android.dialer

Resources

CPU

FIGURE 28. The Approximation of crash point

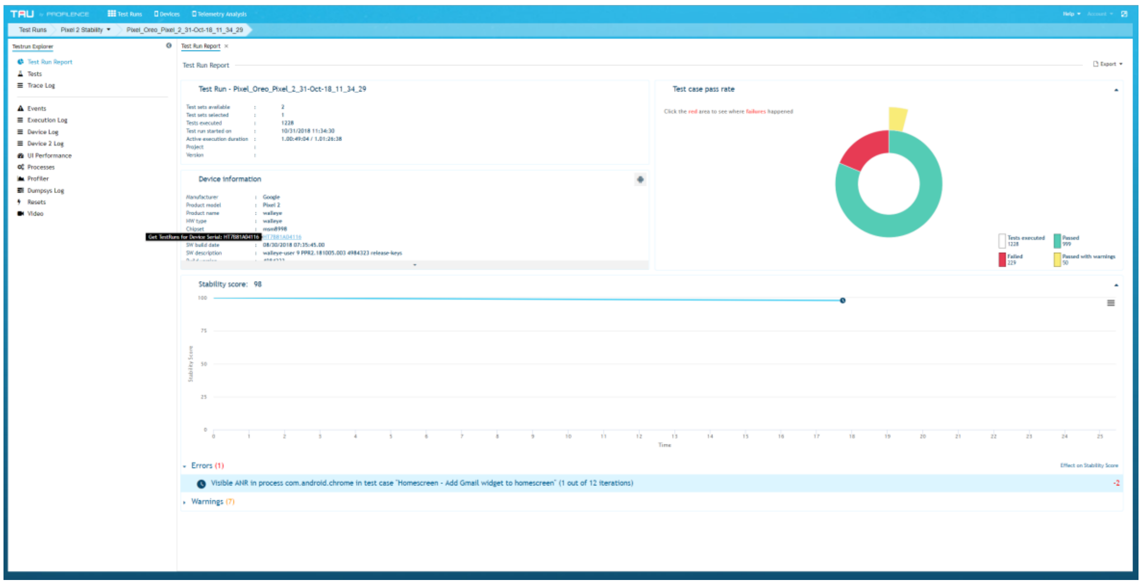


```
private mapImarkerY(hayStack: Array<Number[]>, timeStamp: Number): Number {
let b = hayStack.findIndex(x => x[0] === timeStamp); // first check if matching
data is found
   if (b > -1) {
     return hayStack[b][1];
   } else {   // no exact match !
     let X1_index = 0;
     let found: boolean = false;
     for (let i = 0; i < hayStack.length; i++) {
       if (hayStack[i][0] > timeStamp) {  //find the nearest highdata
         X1_index = i;
         found = true;
         break;
       }
     }
     found = false;
     let X2_index: number;
     for (let i = X1_index - 1; i > 0; i--) {   //get the next data to the
highdata
       if (hayStack[i][0] < timeStamp) {
         X2_index = i;
         found = true;
         break;
       }
     }
     if (!found) {  // if no high or low data found
       return 0;
     }
     let slope = (Number(hayStack[X2_index][1]) -
Number(hayStack[X1_index][1])) / (Number(hayStack[X2_index][0]) -
Number(hayStack[X1_index][0]));
     if (slope == Infinity) {  // bad data, i.e data itself are 0
       return 0;
     }
     let targetdata1 = Number(timeStamp) - Number(hayStack[X1_index][0]);
//slope of the drawn line  straight line formula from cordinate geometry
     let temp = targetdata1 * slope;
     let targetdata = Number(temp) + Number(hayStack[X1_index][1]); // get the
point on the line equation
     if (targetdata != Infinity) {  //check for bad data if any. 0/0 case
       return (targetdata);
     } else {
       return 0;
     }
   }
 }
```

FIGURE 29. Source code for approximation of crash point

The Figure 28 shows an area chart where the approximation of the crash point is carried out. Approximation of crash point is vital to figure out when exactly the crash occurred in the test run. The mockup designed for this UI is also presented in the FIGURE 27 above. Source code (Figure 29.) shows a code snippet of the approximation of the crash point.

The FIGURE 30 below shows a screenshot of a test run view that has an easy access link to the project's view.



Test Run - Pixel_Oreo_Pixel_2_31-Oct-18_11_34_29

Test sets available	:	2
Test sets selected	:	1
Tests executed	:	1228
Test run started on	:	10/31/2018 11:34:30
Active execution duration	:	1.00:49:04 / 1.01:26:38
Project	:	
Version	:	

Device information

Manufacturer	:	Google
Product model	:	Pixel 2
Product name	:	walleye
HW type	:	walleye
Chipset	:	msm8998
Get TestRuns for Device Serial: HT7B81A04116	:	HT7B81A04116
SW build date	:	08/30/2018 07:35:45.00
SW description	:	walleye-user 9 PPR2.181005.003 4984323 release-keys

Click the red area to see where failures happened

Tests executed 1228	🟩 Passed 999
🟥 Failed 229	🟨 Passed with warnings 50

Errors (1)

Visible ANR in process com.android.chrome in test case "Homescreen - Add Gmail widget to homescreen" (1 out of 12 iterations) -2

Warnings (7)

FIGURE 30. The Detailed Test run report

3.2.3 UI development for requirements

The Requirements tab contains the list of requirements for the test run. The requirements are the test that a device must go through to make sure that a specific feature is tested well. In this view, the result of such test is displayed. For easy access to the data, the results are color-coded and displayed along with the icons.

There is a huge amount of data for the requirements view and it needs a lot of processing before data is presented hence data is received as Protobuf which makes data retrieval and processing a bit faster.

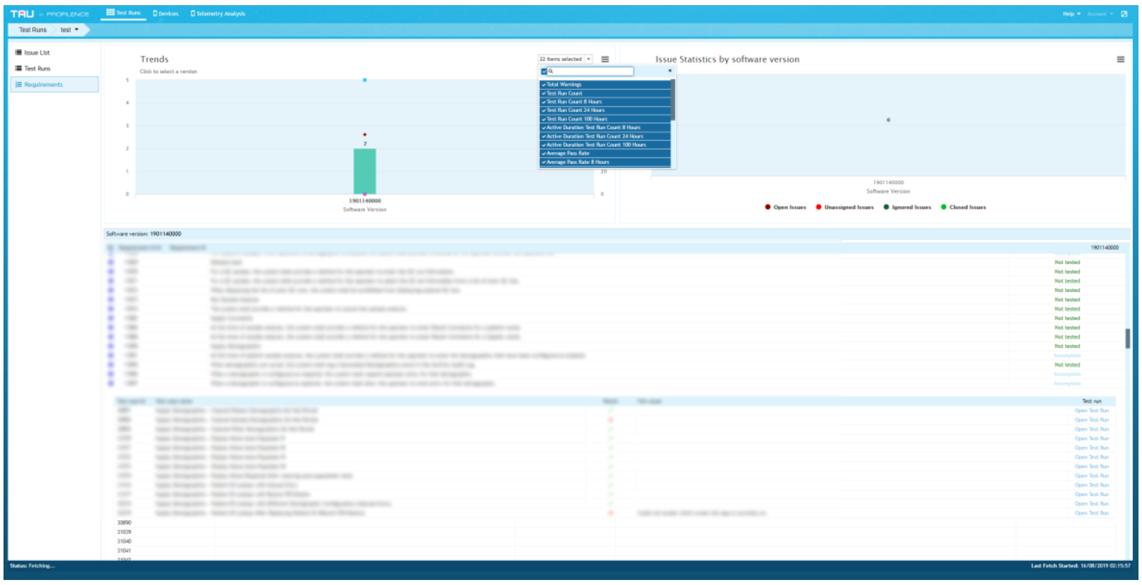


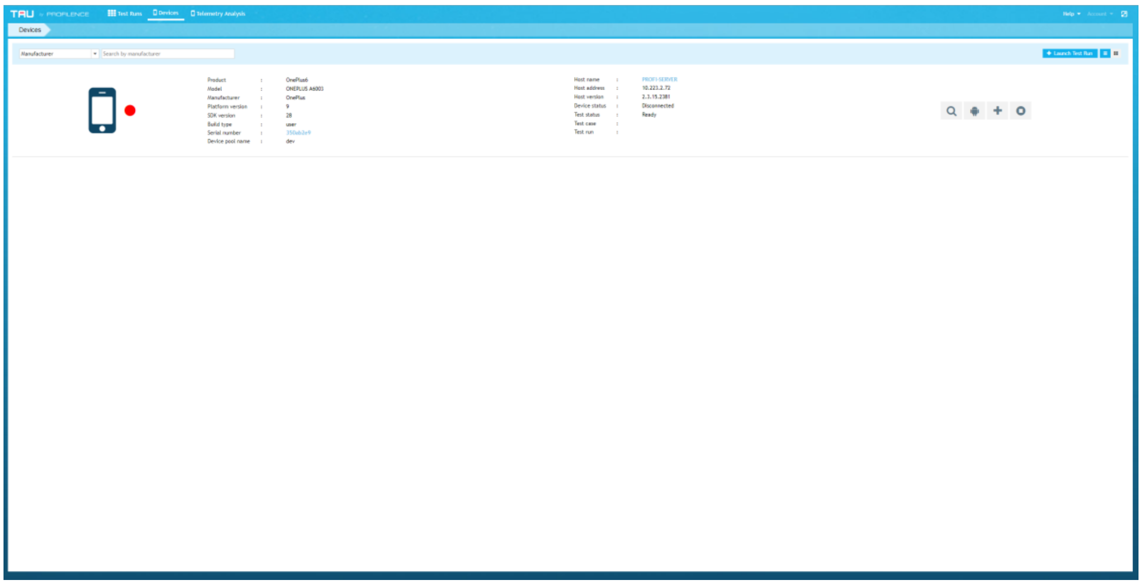
FIGURE 31. The Requirement view

The Figure 31 shows a result for the requirement list. The requirements are marked as “Not tested”, “Incomplete”, “passed “and “failed”. The user can see specifically how a requirement passed, failed or incomplete. If the requirement is still incomplete, then it means further tests need to be performed.

3.3 UI development for Devices

Devices view is a dashboard for the user to see how many devices are currently undergoing a test run or are ready to start a test run. From this view, a user can start or stop a test run. A brief information to a device is also available to distinguish a device for one another.

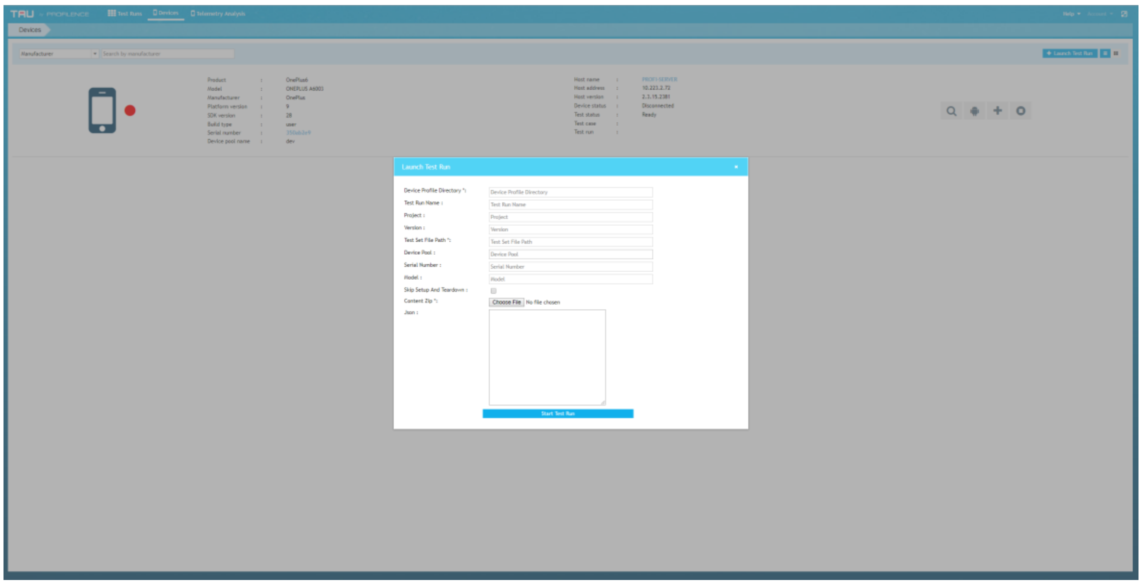
The FIGURE 32 below shows the dashboard view for available devices. The dashboard is currently showing that there are only one device undergoing tests and it shows all the necessary details of the device.



Product	:	OnePlus6
Model	:	ONEPLUS A6003
Manufacturer	:	OnePlus
Platform version	:	9
SDK version	:	28
Build type	:	user
Serial number	:	350ab2e9
Device pool name	:	dev

Host name	:	PROFI-SERVER
Host address	:	10.223.2.72
Host version	:	2.3.15.2381
Device status	:	Disconnected
Test status	:	Ready
Test case	:	
Test run	:	

FIGURE 32. The Devices UI



Device Profile Directory *:	Device Profile Directory
Test Run Name :	Test Run Name
Project :	Project
Version :	Version
Test Set File Path *:	Test Set File Path
Device Pool :	Device Pool
Serial Number :	Serial Number
Model :	Model
Skip Setup And Teardown :	☐
Content Zip *:	Choose File No file chosen
Json :	

FIGURE 33 Test run starter UI

The Figure 33 shows a UI for starting a test run, A user can simply click the easily accessible buttons to start a test run which contains pre-complete the form with necessary information already filled in.

4 CONCLUSION

During this development, different technology and tools were evaluated and used to reach the aim of the thesis project. The UI of the web application was developed and described in the chapter 3, which describes all the UI parts of the application.

In the chapter 2, different tools and libraries used were defined and reasons why these libraries were used were also elaborated. The web application for automated comparative report generation from test automation results was created for Profilence Oy.

New web technologies, such as Angular and RxJS were used for development. The reports generated by the application were analytical and supplied a dashboard view to the report reviewer. The users can now view the reports without having to go through millions of rows of data and just simply view the graphs and get an overview of the test results.

Switching to the developed web application from this project not only saves a lot of time for test automation but also reduces the stress to go through the data. Automated report generation also allows the developers to worry only about development and not about the complexities of having to set up the test environment every time for new test cases.

Getting an overview of the test run makes it easier to track and fix the new and existing bugs. By completion of the web application, Profilence Oy has already deployed the web application to the clients. The clients’ process to deploy a latest version with minimized bugs has been utterly improved due to the new web application. The client is now able to get the report in a presentable format in which stakeholders who are unaware of the logs and their meaning can now understand the result by the help of graphs and make decisions based on these results.

REFERENCES

1. Buffers, G. P. (2019). *Protocol Buffers*. Date of retrieval 5 July 2019, from developers.google.com: https://developers.google.com/protocol-buffers/

2. Extensions-Reactive. (2017). *About the Reactive Extensions*. Date of retrieval 15 July 2019, from Reactive Extensions: https://github.com/Reactive-Extensions/RxJS

3. guru99.com. (2018). *Postman Tutorial*. Date of retrieval 12 August 2019, from Guru 99: https://www.guru99.com/postman-tutorial.html

4. Hubbard, H. D. (1939). Graphic Presentation. In H. D. Hubbard, *Graphic Presentation.*

5. Inc, G. (2019). *Angular - Architecture Overview*. Date of retrieval 14 August 2019, from Angular: https://angular.io/guide/architecture

6. Microsoft. (2019). *Visual Studio Code UI*. Date of retrieval July 2019, from https://code.visualstudio.com/docs/getstarted/userinterface

7. Pierre, J. S. (2018, August 22). *The Firebase Blog: Exporting Crashlytics data to BigQuery*. Date of retrieval 15 August 2019, from firebase.googleblog.com: https://firebase.googleblog.com/2018/08/exporting-crashlytics-data-to-bigquery.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+FirebaseBlog+%28The+Firebase+Blog%29