OULU UNIVERSITY OF APPLIED SCIENCES

Oleksandr Gudenko

DEVELOPMENT PROCESS OF LEVEL 2 API FOR CI SUPPORT AT NOKIA NETWORKS

DEVELOPMENT PROCESS OF LEVEL 2 API FOR CI SUPPORT AT NOKIA NETWORKS

Oleksandr Gudenko
Bachelor's Thesis
Spring 2022
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Oleksandr Gudenko
Title of bachelor's thesis: Development Process of Level 2 API for CI Support at Nokia Networks
Thesis supervisor: Kari Jyrkkä
Term and year of thesis completion: Spring 2022 Number of pages: 31

This thesis documents the experience and steps taken during the Level 2 API development at Nokia Networks. A Level 2 API is a higher-level API collection that has advanced computing features and, in some cases, can utilise a main Level 1 API to create-read-update-delete, a collection of CRUD requests to achieve the results. The goal of this thesis project is to describe the process and environment during the development of the Level 2 API.

The project follows a development of a new feature set for a configuration database project, named configDB, which is a web application written with Node JS and React JS. The configurations, which configDB manages, are stored in the JSON form, they are collected to create a test suite for radio software under testing. The finished test suite is exported to a Robot Framework script which is used during the CI test run.

The result of the study highlights the challenges with project development while contributing to the larger team effort. Unexpected interruptions, such as bug reports and parallel project support requests, resulted in partial thesis project implementation.

Keywords: JavaScript, NodeJS, Software Development

CONTENTS

VOCABULARY

API	A set of functions and procedures needed for creation of an application that a data from the outside source such as operating system, application, or other service.
back-end	A part of the software that handles communication with storage, database and does other behind-the-scenes functions that make application that it is.
CI – Continues Integration	Is an automation of integration of code contribution from multiple sources into a single software project. The idea is to automatically validate the contribution before adding it to the main project.
CRUD	Create, Read, Update and Delete. A base set of operations for data management.
front-end	A part of the software user sees and interacts with.
GUI – Graphical User Interface	A visual interactive system made for simpler user-software interaction.
HTTP	A protocol which allows for internet communications and resource fetching. The base of internet as we know it.
Meta Data	Is a data that explains the data send and received during machine communications.
PDU	Power distribution unit. Responsible for handling the state of electronic devices.
R&D	Research and development.
Sandbox	Isolated environment for a program or a service to run without being allowed to interact with anything else.

Smoke Test	Could also be called a build verification testing. a test to verify if whole system is working correctly after added changes.
Unit Test	A collection of tests used to validate a specific part of the software system for potential issues during development.
VM	A virtual machine. Is a virtual computer or a software defined computer within physical servers, existing only as a code. [7]

1. INTRODUCTION

The aim of the thesis consists of developing a new set of features for a support tool in a continuous integration of radio hardware and software of existing and future products.

The support tool is a type of a configuration management database, named configDB. The tool is built with JavaScript [1] and is based on a Node JS [2] library. The tool's purpose is building a Robot Framework [3] test suite out of JSON configuration files. ConfigDB allows users to reuse older test suites, test operations and test configurations to assemble new testing scenarios. Front end of the configDB is implemented with React JS [4] and Redux JS [5]. It is used for conveniently interacting with, managing, and creating test suites. The test suites are exported to be used in continues integration of radio unit, later shortened to radio CI, to ensure the quality of developed software.

The main purpose of the thesis is a development of a higher-level API for configDB. The API in question would be responsible for processing test suites and configuration files without the need of a graphical user interface by using a lower-level API just like a client. Such a function will allow the user to dynamically generate a test suite for a specific test run, during the run itself.

Due to the daily tasks being interrupted by higher priority requests, such as bug fix reports or an alternative feature development request, it has been decided to write a diary type thesis for this project. The project was commissioned by Nokia Networks in the city of Oulu, Finland. Nokia is developing multiple solutions that help people around the world act together.

2. THESIS WORKING MODE

2.1. DIARY TYPE THESIS

This chapter will explain and justify the choice for selecting a diary type thesis instead of the traditional one. The project was done in an active R&D environment. The task assigned to me for the feature build could not be given a full attention, as the interruption in a form of bug reports or higher priority feature development request would constantly find a way to my desk. With such an unfocused environment having a diary type thesis explaining the situation by weekly was the most reasonable choice in my opinion.

2.2. WORKING MODE

Due to the ongoing pandemic, at the moment of writing, I am performing all my work duties from the home office as a part of the 2-person developers' group with an additional project supervisor who belongs to a larger team. Due to team's having a multinational local background and a collaboration with multiple sites in other countries beside Finland, most of the communications occur in English language.

My regular tasks consist of developing and supporting configDB, but I also perform a range of secondary tasks. Usually, when new technologies or features related to configDB are being taken into use, I would be testing them to judge their suitability and convenience as well as review the alternatives. Any additional, not planned for the current sprint, feature development requests are managed by me if they are server-side related.

2.3. REQUIRED SKILLS

The primary platform for configDB development is a Linux based operating system. It is expected of the developer to be capable of setting up their own development environment and try to resolve any arising issues by themselves first. But it does not mean that good collaboration skills are not required. Every person has their own responsibility to carry out, but some tasks require two or more people to work together, as well as responding to support requests professionally and timely.

Due to configDB being a web application primarily developed by just two people, a new member should be preferably a full stack developer who are capable of learning new technology as they are expected to work with a whole project. The most important programming languages are JavaScript, Python, and python-based Robot Framework.

JavaScript is the foundation of configDB development. The libraries chosen for the application are Node JS and Express JS as the core in the back end while React JS and Redux JS are our choices for front-end client implementation.

Python is a choice for automation of my team. Every test is either created with Python or used to create libraries for Robot Framework based tests.

2.4. TEAM'S WORKING ENVIRONMENT

My team holds a sync meeting every Monday, Wednesday, and Friday, where we are reporting on our progress and challenges that we are facing at the time. Additionally, every feature build has an opening meeting and closing meeting. The team discusses the sprint and collects general feedback for later improvement.

Nokia is using Jira to host their agile style board which incorporates multiple aspects from popular boards, such as Scrum, Kanban, and Cadence, to keep track of the tasks during feature builds.

Documentation for developed tools and guides for setting up technologies in use are hosted by Confluence. Microsoft Office is used extensively for communications and presentations as it is easy to use and quick to learn.

Employees have a choice of a personal development environment by selecting between a virtualized Linux environment or a Windows 10 installation curated by the Nokia software team.

3. SUPPORT TOOL HIGH LEVEL API

The objective of the thesis was to develop a higher-level API which would allow communications with a server previously limited only to a client and accessed via a graphical user interface. This would allow a radio base station tester to create a script that will generate a test suite with the configDB API during the test run. Currently, users must interact with the configDB client to create a test suite that will fit the radio-software combination. Different but rather similar types of tests will make users recreate the test suites with different parameters. This leads to the increased number of similar records polluting the database. The GUI usability convenience is lost due to database pollution, as looking for the correct record becomes increasingly more difficult.

Before the level 2 API, the record processing was split about equally between the client and server, which was enough, but with an active growth of configDB, a slowdown has been noticed in the application's client. A new API will allow to shift the processing load from the client, more to the server. Such action will help guarantee the same level of performance and hold a consistent experience on a bigger variety of devices, as the server will perform most of the processing, while the client will display the results. An expected outcome of Level 2 API development can be viewed in the figure 1 under the list of requirements.

List of requirements:

1. Improve responsiveness of the user interface
2. Offload a client by moving processing to the server
3. Allow for a more convenient interaction with automated CI pipelines
4. Have an easier scalability of the application
5. Improve user's experience

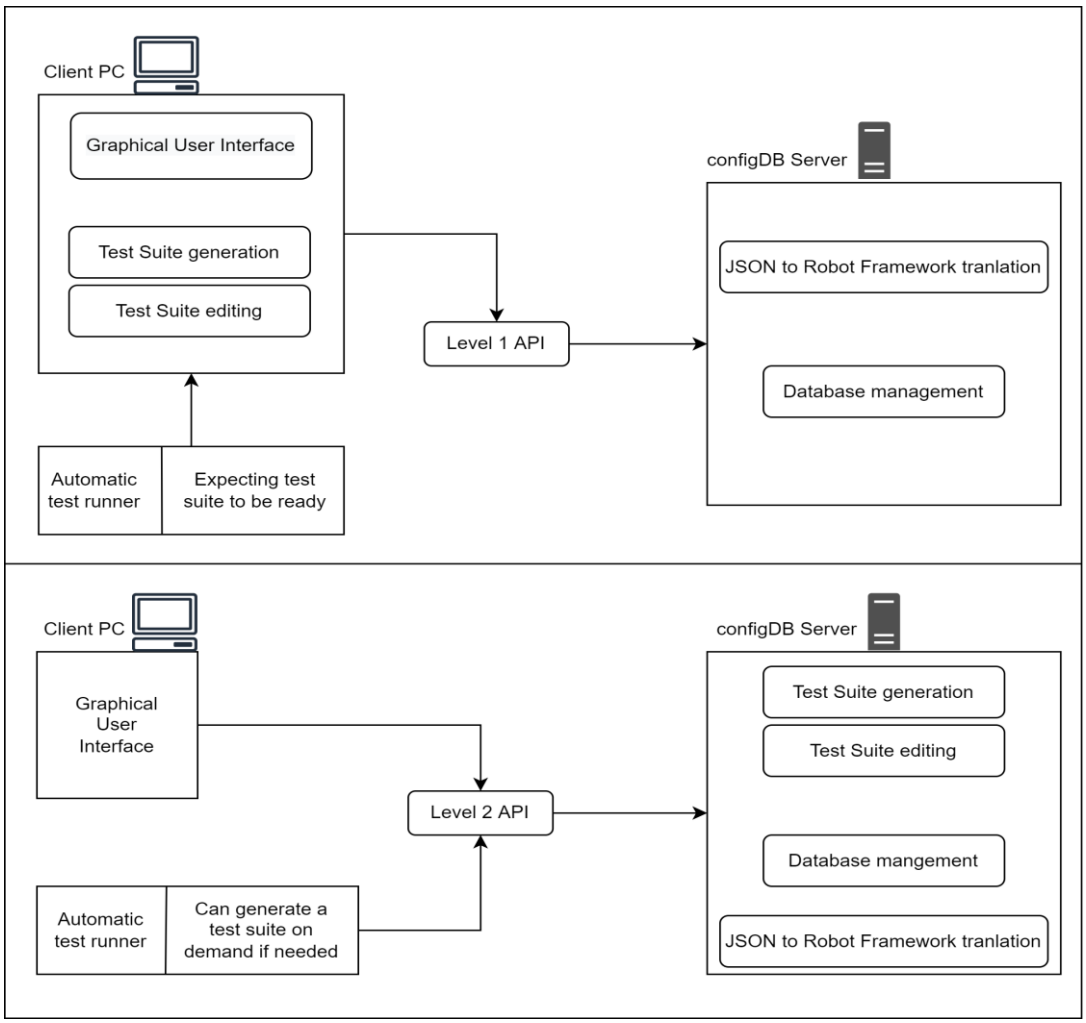

Figure 1: ConfigDB workflow before and after Level 2 API implementation

4 DIARY ENTRIES

This diary covers a total of 8 weeks of work, from 01.03.2021 to 23.05.2021. All reports will be done in a weekly manner.

WEEK 1 (1 MARCH 2021 – 5 MARCH 2021)

The week started with a closing meeting of the ongoing feature build. The team held a discussion meeting with every member providing feedback about their experience during the run to the team leader. All comments were reviewed and used to improve the following sprints. The team spent the rest of the day finalising all remaining task and preparing for the next feature build opening. The day finished with a start to a level 2 API design phase.

On the following day, a new feature build kick-off meeting was held. My team went over the tasks for the next sprint. After the meeting, the required documentation for a level 2 API, initial version, was finalised. Current plans for a Level 2 API were to create a collection of 4+1 API endpoints; tags management API, record usage validation API, test suite management API which includes: create, remove, and edit. The last and an extra feature, which had been planned before as an independent project, has been added to the level 2 API collection and it is a version control API.

Once the first version of documentation was developed a feedback meeting was organised. After discussing the documentation, the team wanted the record validation API to be build first. The idea behind this endpoint is to validate if a record is referenced anywhere else in the database to ensure that editing a record will not break anything accidentality and allow a safer environment for the users. The response of the endpoint will tell the user if modifying a record through configDB is deemed safe, or if the record is referenced in the active test suites and could potentially break them and by extension the radio CI.

By the end of the week, the first version of the validation usage endpoint was created, it was renamed to a “relationship API” due to a name conflict. Another endpoint belonging to the level 2 API, which is responsible for the validation of structural integrity of the whole test suite, has been assigned the original “validation API” name.

The relationship API was quickly developed due to a powerful collection of database query methods offered by MongoDB. This simplified the amount of additional processing needed and therefore speeded up the development time.

Before the week ended, the feature implementation request ticket for configDB had been submitted. The request was to add support for a Robot Framework tags system to configuration files during a test suite export. As this request has a higher priority than level 2 API development, it was decided that I start working on this issue starting on Monday next week.

This week’s focus was dedicated to 1st and 5th requirements listed in chapter 3. The team decided to prioritise the safety and stability of the editing environment. The relationship API will offer a better piece of mind when handling so many moving parts as it is responsible for verifying the links between records and can help to warn users of potential issues arising after the edit.

WEEK 2 (8 MARCH 2021 – 12 MARCH 2021)

Once the morning meeting report ended, the work on tag system support started. After assessing the state of configDB, we estimated that the duration of tag system implementation would take around 5 hours of work due to configDB built on a reliable and flexible foundation which considers the possible growth in direction we are heading. The first version of the feature was developed on Monday and was set for the BETA stream rollout with a closed ticket.

While the documentation for a level 2 API was under multiple reviews, a parallel project, of development of a version control system for the test suite records that are stored in MongoDB, was split between me and my co-worker. The task has a front end and back-end development, where developing the API at the back end would be my responsibility and afterwards assisting with integration to the graphical user interface. This feature development project has been incorporated into the level 2 API as a version control API, mentioned last week.

In the middle of the week a bug report for configDB was submitted. An issue occurred in the configDB API. After multiple Git branches were merged into the master branch, of which some were relatively old, the API no longer zipped and delivered correct files to the users. The issue took us two days to resolve. Once the issue was resolved, I continued development of a version control system API for the configuration database.

At this point in time, a basic design of a version control prototype for the test suite records has been made. Now, before proceeding with the test suite export, the API verifies if a specific version of the suite is requested by the user and if not, it is looking for a default version stated in records metadata. After the version has been selected it proceeds to collect all necessary configuration records from MongoDB and create a JSON file containing all the test information. The JSON file is used as a blueprint to create a Robot Framework executable test and package all reassures to a singular zip file before sending it to the user.

During Friday, a couple of minor quality of life improvements were made for configDB developers. One of such improvements is a remake of node scripts. For example, instead of running two different scripts to test an API, it was reworked to execute one. Another improvement is optimization of a client and server development environment by making independent scripts work better together for a seamless code reflection in the browser window. Also, a script for automatic configDB BETA development branch deployment was made. The machine hosting the BETA stream of the application was optimised for this purpose.

Next week will begin with review and addressing of the rest of Level 2 API documentation feedback.

This week was focused on the start of the version control API documentation and basic implementation of the prototype and addressed requirements 3,4 and 5. The version control project was added retroactively to the level 2 API, but it was something we all kept close look on and after a careful judgement, it was decided that it fits the level 2 API requirements and would bring the most benefits in our current environment.

WEEK 3 (15 MARCH – 19 MARCH 2021)

This week, the designs for the version control and the rest of the level 2 API endpoints were approved after addressing the feedback given during review meetings. While the development of the version control system could not continue due to a new higher priority request that have been sent my way, my co-worker started GUI implementation.

The higher priority task was about implementation of another Robot Framework feature, which was missing in our JSON-to-robot-framework exporter in configDB. The feature is a “suite tear-down,” which is a set of instructions executed when a Robot Framework test suite stops, for example due to errors, a test failure or simply when it ends. Implementation of the suite tear-down feature took two days. It was taken into use quickly and the main focus returned to working on the version control API.

For the next two days the original configDB code was retouched to add a version control support. It has become possible to choose which version of the suite to export and in case if none is selected by the user, the API exports a default version of the suite which was previously approved by the owner, the default is version one. This was accomplished by verifying if there is any request for a specific version before starting the export process. After a version is selected, the API collects all necessary files from the database to assemble a Robot Framework test suite from collected JSON files, then it creates a zip package and sends it to the user.

By the end of the week an alpha version of the version control API was developed, and a close collaboration with a co-worker started in order to test new code in the GUI.

This week begun an active development of the version control API as a major sub project that targets all requirements. The alpha version of a “version control

API” was implemented and is being tested together with the GUI. The back-end development progress is going well and could even be considered ahead of the schedule.

WEEK 4 (22 MARCH 2021 – 26 MARCH 2021)

This week my attention was primarily dedicated to the version control API development, but it was disturbed by the very due update for my virtual machine, later called a VM.

The VM where all my work being done, such as developing configDB, had only 8 gigabytes of dedicated random-access memory, RAM. The project and therefore hardware requirements evolved rapidly during the last few months, and it outgrew the capabilities of allocated resources for my VM.

It was no longer possible to continue using this VM as it was for configDB development. It was running out of memory and crashing, after about every 30 minutes to an hour of active work. This issue was swiftly reported to my supervisor. After analysing the situation, he moved my VM to another server rack which had more available resources for me to use. After receiving an upgraded VM with a double the amount of previously available memory, 16GB of RAM instead of 8GB, implementation of version control API could resume.

The version control API was finished last week and only received small adjustments requests, which were quickly addressed early this week. It was time to integrate the API with a graphical user interface, GUI, previously worked on by my co-worker. Initially, my involvement with GUI implementation was minimal. My primary task would have been to assist by explaining how the Level 2 API was functioning. In early stages of integration, it was noted that the amount of work to be done on the front end was severely disproportionate to the work left to be done on the API and it would require significantly more effort to make it work. After discussing the case with supervisors, it was agreed on a closer collaboration between me and my co-worker, that my task would consist of assisting with a further client integration and later development of missing functions.

The GUI was implemented with React JS and Redux JS. The way configDB GUI handles records are complicated. It takes the original record as a blueprint which

has references to other records in the configDB database, hosted in MongoDB. It fetches each referenced record with the help of the base API and then handles them individually with a proper reference to the test suite. It is possible to edit all these records individually through one main test suite editor as if the user were editing a single file. The independence of the records was implemented to improve each record's reusability. Instead of creating a new record with the same content for the new test suite, the original is being referenced in multiple test suites instead. For example, when tests are using the same power distribution unit, or the PDU. It sounds inefficient to write the same configuration and operational code to activate it in every case the PDU is used. The user's time would be better spent on running the tests and handling the results.

The test suite export API has been upgraded with a version control API and the changes needed to be accommodated in the GUI. First, the JSON structure of the test suite was modified to support the version control API. Secondly, the old JSON structure was replaced, see Figure 2, with a version control supported added in the front-end. Lastly, configDB functional was restored to the previously offered feature level which was impacted by a version control and a level 2 API changes.

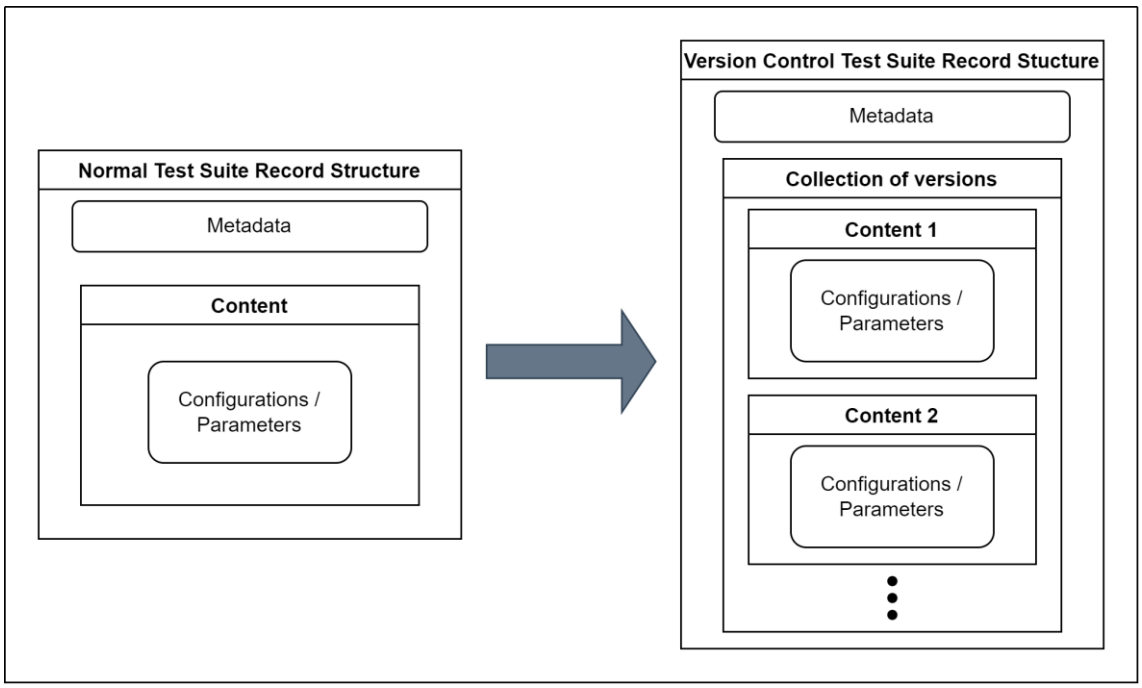

Figure 2: ConfigDB database record structure changes brought by version control feature

It was time to implement additional features that were introduced with the version control. The first feature to be focused on was a default version selector. An example of this: selecting a version 2 to be a standard exportable version in case no specific version is requested.

Once it was complete, our supervisor notified us that a script responsible for transforming all the current database records to the version control type records we have just integrated into configDB was ready for testing. After verifying the results of the script, the conclusion was drawn that the script worked as expected. It automatically modified the structure of all existing records in the database to the new standard without manual interfering.
The second component, which was made, is a version selector in the suite editor for switching between versions during editing. The implementation took two days due to an added complexity of managing multiple versions and records linked to them.

The configDB editor needs to know which version is a default one, which version is being edited and how much it has been changed from the original after the editor session has ended. Once the version selector is used, the selected version

information is loaded to an editing sandbox. For example, while a user interacts with one version, they cannot make any modifications to other versions. Each time a user decides to switch between versions, they need to save editing progress if they have made any changes. This way we can guarantee that modifications are not lost between version switching and will not be applied to the wrong version.

The last day of the week was dedicated for Git branches clean-up and preparation of tasks for the next feature build.

The development of version control API integration is providing a support for majority weaknesses in the configDB user experience. Successful integration of front-end and back-end code allowed for timely implementation of front-end GUI features for record version switching. Additionally implemented upgrades in a user interface started preparations for the API record transfer from the version 1.0 to the version 2.0 in the database.

WEEK 5 (29 MARCH 2021 – 5 MARCH 2021)

This week had a feature build change. The team held a retrospective meeting where it was discussed the team's achieved progress and what kinds of good and bad experiences we encountered. After analysing how everything went, another meeting was held It was an opening meeting for an upcoming sprint.

In the beginning of the week, a new ability to create new versions for the test suite was added. This means that configDB either creates an empty content or clones the content of another version to the new version. The user can either create a whole new test suite or build upon any available version without interfering with that version because once the new version is created from the base version, it does not have any active link and is treated independently.

This was the last piece needed to set the version control in alpha, but when all pieces were assembled, it was noted that something was functioning incorrectly. After running a test for the GUI feature set, it was discovered that some parts of the test suite were being modified when they were not intent to be.

Upon a further investigation it was discovered a flaw in the way of how the GUI implementation creates a copy of the original records. The project had been developed under the assumption that the code build as a foundation took an independent copy of the object, which means that the original record would never be modified, but due to a bug in the cloning logic all operations were performed directly on the original object. Instead of making a new copy of the record a reference was created to it instead.

The issue was solved by using a Ramda [6] library for JavaScript, which provided a good solution for a deep copy of the JSON file. Later a version control version of configDB has been deployed to the BETA server.

On Thursday, which was the last day of the week due to Easter holidays, a bug report was submitted for a configDB application. The issue occurred with the summertime savings switch. The time was moved 1 hour forward and the difference in time formats between the server and the client resulted in a 4 hours' time difference in logs.
It was decided to fix the issue by adding a time zone to send timestamp so that our server would not need to guess the time zone by its own location, which was causing the issue. It was an easy task, and the patch was applied during the day.

A great progress has been made during the version control API development which has become the main subproject of level 2 API. The first BETA release of the application was made and feedback from the team was gathered.

WEEK 6 (6 APRIL 2021 – 9 APRIL 2021)

The week started on Tuesday, due to Easter Monday. Early in the day it was discovered that last week's issue was only partly resolved. The fix we applied successfully worked in our development machines, but the patch did not work for our production machines. The issue persisted due to the differences of a local time configuration in development and production machines. The machines used for developing the patch are displaying time in Helsinki Local Time (EETC), but our production hosting machine has Coordinated Universal Time (UTC). The confusion occurred when the EETC timestamp was sent by the client, but the server misunderstood the data as a UTC time and the server added an additional local time difference on top, so UTC+2 was changed to UTC+4. As soon as the issue was clearly identified, a proper patch was applied quickly.

The next step was to update older CI tests for the front-end GUI, as it needed to test for newly added elements. Previously, a front-end CI test was developed with the Selenium library for the Robot Framework by me. So now with the Level 2 API partly implemented, the tests needed to be updated and the task was assigned to me.

The new CI test flow would start when a test suite is created through the GUI. The CI test firstly confirms that the version description is created together with other metadata, such as a name and a record type. Then the CI test verifies if it is possible to create test cases, followed by testing the ability to create a new test suite version from a version 1 or as an empty suite. The last new test was to verify if switching between versions was working correctly.

An update of configDB CI tests to include tests for Level 2 API features was developed so far. This will help to guarantee a stable user experience along the development of the rest of the missing features.

WEEK 7 (12 APRIL 2021 – 15 APRIL 2021)

This was a less eventful week due to no new bug reports or assist requests were made. With the version control being tested in the beta release, the next requested level 2 API endpoint was a tag management API for software packages. The goal of the endpoint is to set a new tag to a software version which is either being tested, has been avoided or finished testing either successfully or unsuccessfully.

Four supported tags are:

1. Pass
2. Fail
3. Triggered
4. Avoid

Out of all above mentioned tags, “Avoid” is the only tag that needs to be set manually by a human, while the rest are made for automation. Software packages are in a “Triggered” state while they are under testing and can only be resolved to “Pass” or “Fail”, which is directly reflected in a Tag system view on the configDB. While “Avoid” is used only in cases where it has been determined by the user that the software package is potentially dangerous for installation or is out of order and simply would not install. After successfully pitching the idea of how the endpoint should function, the implementation process started.

First, the “set one tag” functionality was implemented, and it did not take much longer to add a possibility to send a custom array of tags after the review was deprecated. As changing the tag reflected the package’s state change and it was decided that the package can only have one state at the time, therefore contradicting the whole API purpose.

After adding the API endpoint, an issue with the main API had been encountered. Debugging took the rest of the week. It was found when our main API was retrieving the records that it was being verified for a content validity, but the

verification process contained a weak verification statement which allowed empty records to bypass it.

This week’s focus directed to the requirement 2, which is automation. The API has been designed, reviewed, and approved.

WEEK 8 (19 APRIL 2021 – 23 APRIL 2021)

During the final week of the thesis the tags API with automation in mind was successfully finalised. The results are as follows:

1. When a new software package is taken into testing, the status of the package is changed to “Triggered” on the configDB platform.
2. If the test is finished successfully, the “Triggered” tag is replaced by a “Pass” tag. In case the run fails, a “Fail” tag is assigned instead. If the test encounters an unexpected exit, the “Triggered” tag will be left untouched. Users need to manually assign either the “Fail” or “Avoid” tag.
3. If a user, after reviewing the package, identifies that the package is dangerous or simply unnecessary to test, they can add a tag “Avoid” which will make the runner skip the package and not retest it unless the user manually does.

Software packages are displayed with tags as the result of automated CI runs and can be viewed by anybody interested in the results.

The finished development of the tags API for the test result verdicts on configDB. This API endpoint allows both automated CI runners and humans to place verdicts on the software package’s state and can be updated in real time, which fulfils the second requirement for the Level 2 API.

5 CONCLUSION

The Level 2 API project begun as a design of a set of complex API endpoints build on top of basic CRUD requests offered by configDB. The goals were to ease the processing load off a configDB client and better scale the project by utilizing the server's resources instead, which can be expanded when so is required. This would offer a better consistency in user experience and automation integration as the users would be able to assemble a required test suite on the fly by directly communicating with the API during the test, instead of relying on the GUI to prepare the suite beforehand while the configDB client will also take advantage of Level 2 API by no longer processing records locally.

Unfortunately, a parallel project has been pushed forward and it has been decided to merge the two of which the latter received a higher priority. A test suite version control API is responsible for test suite record version management. Its main purpose is to provide an ability of safely adjusting and developing a test suite record by keeping independent multiple versions of the suite in one database entry and this provides an ability to switch between them quickly in the client's editor. This also helps with database record management as the same test suite, which was slightly modified, is not appearing multiple times, and therefore not wasting space for storing and processing power for retrieving the correct one

In the end it has become a Level 2 API + test suite version control project of which version control has been fully implemented, while the Level 2 API development passed initial stages. This was not an expected outcome in the project planning phases and was not something me and my supervisor could foresee.

Eight weeks is a short time frame to design, develop, integrate, and test a new API of such a large scope. Originally, we did not expect a major refactor of our GUI test suite editor. Some changes were expected due to added features of a Level 2 API as well, but not the performed ground-up rewriting of some of the core editor components due to the version control inclusion. Additionally, support

tickets were interfering with the time and workflow of the project. Luckily, the test suite version control covers most of the quality-of-life improvements to configDB users and offers a much-needed reliability improvement to record management.

Out of all originally proposed Level 2 API endpoints only two where successfully implemented. They are a tags API, a radio software package tagging, and a relationship API, which was responsible for verifying the references of records throughout the database. Granted, these two are the simpler in design compared to the rest and do not affect client processing in a major way, but they lay a foundation for the future functional beside the Level 2 API. The three unimplemented endpoints are responsible for direct test suite management, such as generation of the new suite and reconfiguration of an existing one with are optimisation for automation.

The project has not yet been released to every site, but it is going through real world application testing at the local team at the Oulu site after passing every unit test developed for it. The feedback is positive, and it is expected to have a production release in the coming weeks. After the release, the project will be available for use for all configDB users at every Nokia Networks' site.

The project was able to alleviate the load spikes in the configDB client and made it more responsive. But as it has been stated above, the Level 2 API was not fully implemented. The duration of the thesis was not long enough time frame to complete all proposed features while developing a version control solution, which my supervisor at Nokia considers an add-on but non the less a part of Level 2 API. Finishing remaining three endpoints will further lower the client load substantially, in comparison we expect to lower the load from about 70% to 10% after Level 2 API full integration. Such a massive boost in performance will result in page load time as low as a quarter of a second compared to current load times which are taking about one to two seconds, even on devices with exceptionally low computing power. The next step after the Level 2 API development would be looking into web sockets and how they can further shift the rest of the local processing to the server side.

REFERENCES

1. JavaScript. Search date 07.06.2021. https://www.javascript.com/

2. Node JS. Search date 07.06.2021. https://nodejs.org/en/

3. Robot Framework. Search date 26.05.2021. https://robotframework.org/

4. React JS. Search date 08.06.2021. https://reactjs.org/

5. Redux JS. Search date 08.06.2021. https://redux.js.org/

6. Ramda. Search date 2.06.2021. https://ramdajs.com/

7. Virtual machines: virtual computers within computers. Search date 6.02.2022. https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/