



**UNIVERSITY
OF OULU**

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Santeri Moberg

**APPLICATION FOR MANAGING TEST
ENVIRONMENTS IN CONTINUOUS
INTEGRATION TESTING PROCESS**

Master's Thesis
Degree Programme in Computer Science and Engineering
May 2021

Moberg S. (2021) Application for Managing Test Environments in Continuous Integration Testing Process. University of Oulu, Degree Programme in Computer Science and Engineering, 69 p.

ABSTRACT

Continuous integration enables the rapid development of software where each code change is automatically tested and integrated frequently to a shared code repository. Nowadays, with increasing complexity of embedded systems, continuous principles have been adopted in the development of embedded software. Using continuous principles in development of embedded software provides its unique challenges as the code changes must be tested in the hardware, to ensure that the software works in production environment. Some of the challenges are limited amount of test environments, long test and build times, and wide variety of different hardware used in the testing. Because of these challenges, an application which can be used to manage test environments and hardware is required.

At the heart of this thesis is a web application called ReserveTool which provides an interface for management and reservation of test environments. The purpose of the thesis work was to improve the existing application as it had limitations to supporting planned new features. After examining the requirements, it was decided that a new application called ReserveTool 2.0 should be developed.

Implementation of the ReserveTool 2.0 consisted of development of a new database and web UI/REST API for management and reservation of test environments. User experiment was carried out to evaluate the usability of the web UI of ReserveTool 2.0. Feedback from the user experiment was positive with a few suggestions that helped in outlining the future development of ReserveTool 2.0.

Keywords: hardware inventory, reservation, REST API, web

Moberg S. (2021) Sovellus testiympäristöjen hallintaan jatkuvan integraation testausprosessissa. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 69 s.

TIIVISTELMÄ

Jatkuva integraatio mahdollistaa ohjelmiston nopean kehityksen, missä jokainen koodimuutos automaattisesti testataan ja integroidaan useasti lyhyellä aikavälillä yhteiseen ohjelmavarastoon. Nykyaikana sulautetut järjestelmät muuttuvat koko ajan entistä monimutkaisemmaksi, jonka johdosta jatkuvia periaatteita on alettu ottaa käyttöön sulautetun ohjelmakoodin kehityksessä. Jatkuvien periaatteiden soveltaminen sulautetussa ohjelmistossa tuo mukanaan omat haasteensa, koska muutokset ohjelmakoodissa täytyy testata sulautetuissa laitteissa, jotta muutoksen toimivuus tuotannossa voidaan taata. Haasteisiin kuuluu testiympäristöjen rajallinen määrä, pitkät testaus- ja käänösajat, sekä kattava kirjo erilaisia testauksessa käytettyjä laitteita. Näiden haasteiden johdosta tarvitaan sovellus, jonka avulla voidaan hallita testiympäristöjä ja laitteita.

Tämän diplomityön keskiössä on web-sovellus nimeltä ReserveTool, joka tarjoaa rajapinnan testiympäristöjen hallintaan ja varaamiseen. Diplomityön tarkoitus oli parantaa nykyistä sovellusta, sillä siinä oli rajoitteita, jotka vaikeuttivat uusien toimintojen kehitystä. Vaatimusten tutkimisen jälkeen, päätettiin että uusi sovellus nimeltä ReserveTool 2.0 tulisi kehittää.

ReserveTool 2.0:an implementaatio koostui uuden relaatiotietokannan, sekä testi ympäristöjen hallitsemiseen ja varaamiseen tarkoitetun web-käyttöliittymän ja REST API:n kehityksestä. ReserveTool 2.0:an web-käyttöliittymän käytettävyys arvioitiin käyttäjäkokeen perusteella. Kokeesta kerätty palaute oli positiivista ja se auttoi kartoittamaan ReserveTool 2.0:n jatkokehitystä.

Avainsanat: inventaario, varaus, REST API, web

TABLE OF CONTENTS

ABSTRACT	
TIIVISTELMÄ	
TABLE OF CONTENTS	
FOREWORD	
LIST OF ABBREVIATIONS AND SYMBOLS	
1. INTRODUCTION.....	8
1.1. Motivation.....	9
1.2. Structure of the Thesis	10
2. SOFTWARE TESTING AND CONTINUOUS INTEGRATION	11
2.1. Software Testing	11
2.1.1. Different Software Testing Levels.....	13
2.1.2. Regression Testing	15
2.1.3. Test-Driven Development (TDD).....	15
2.2. Continuous Integration	15
2.2.1. Testing in CI	17
2.3. CI for Embedded Software	17
2.3.1. Testing in a Target Hardware	18
2.3.2. Simulated Hardware in CI.....	18
2.4. Continuous Delivery	19
2.5. Use of Continuous Principles in the ReserveTool	20
3. REQUIREMENTS	21
3.1. ReserveTool 1.0 and Its Use	21
3.2. Limitations of ReserveTool 1.0	23
3.2.1. Hardware Inventory	23
3.2.2. Hardware Utilization Rate.....	24
3.2.3. No Support for Dynamic Hardware Configurations.....	24
3.2.4. Usability of the System.....	24
3.3. Different Solutions for the Reservation System	25
3.3.1. Target Administration Tool (TAToo)	25
3.3.2. Velocity	25
3.3.3. Choosing the Reservation System.....	27
4. IMPLEMENTATION OF RESERVETOOL 2.0	28
4.1. Technologies Used in the Implementation	28
4.2. Database	29
4.2.1. Hardware Model	29
4.2.2. Configuration Model	32
4.2.3. Other Models	32
4.3. Authentication and Authorization in ReserveTool 2.0	35
4.3.1. User Groups.....	36
4.3.2. Authentication	36
4.3.3. Object-Level Permissions	37
4.4. ReserveTool 2.0 REST API	37
4.4.1. Implementation of ReserveTool 2.0 REST API	38

4.5.	Web UI of ReserveTool 2.0.....	39
4.5.1.	Navigation	39
4.5.2.	Home	40
4.5.3.	Hardware.....	41
4.5.4.	Configurations	43
4.5.5.	Connection Making View	45
4.5.6.	IP Address and Virtual Machine Views	46
4.5.7.	Orders	47
4.5.8.	Statistics	47
4.6.	Configuration Reservation	49
4.6.1.	Reservation System	49
4.6.2.	Reservation through the Web UI.....	51
4.6.3.	Reservation through the REST API	52
4.7.	Importing the Hardware Inventory	53
4.8.	Tests to Ensure Operability of ReserveTool 2.0	53
5.	EVALUATION AND DISCUSSION	54
5.1.	User Experiment	54
5.1.1.	Survey	55
5.1.2.	Survey Results	56
5.2.	Future Work	57
6.	SUMMARY	59
7.	REFERENCES	60
8.	APPENDICES	63

FOREWORD

This thesis was carried out at Nokia in Oulu. The past year has been quite challenging with the heavy limitations and extraordinary circumstances posed by COVID-19 pandemic. Things were quite different from the usual with the amount of remote working, social distancing, and seeing empty hallways at the site. Nevertheless, I managed to complete this thesis work and my studies at the university are finally coming to an end.

I want to thank everyone in the team at Nokia for guidance and support during this tough year. I would also like to express my gratitude to Aku Visuri who worked as the supervisor for this thesis from the University of Oulu and provided valuable guidance.

Oulu, May 10th, 2021

Santeri Moberg

LIST OF ABBREVIATIONS AND SYMBOLS

API	application programming interface
CCS	cascading style sheets
CI	continuous integration
CPU	central processing unit
CSRF	cross-site request forgery
CSV	comma separated values
DRF	Django REST framework
ER	entity-relation
GUI	graphical user interface
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secure
IP	internet protocol
LDAP	lightweight directory access protocol
NFS	network file system
ORM	object-relational mapper
PSCI	platform software continuous integration
REST	representational state transfer
SQL	structured query language
SaaS	software as a service
TDD	test-drive development
URL	uniform resource locator
VCS	version control system
WMS	warehouse management system
commit	making a set of changes to the code permanent
formset	set of multiple HTML forms

1. INTRODUCTION

Continuous integration (CI) is a software development practice where changes to a shared code repository are frequently integrated. Each integration is verified by automated testing to detect bugs and problems. Utilization of CI ensures that developers always have access to a stable build of the software. Moreover, it results that bugs and problems are found early making them easier to fix, because there is much less backtracking that must be done to find the cause. [1]

This thesis is made as a part of a CI team whose responsibility is to integrate changes to embedded software code to an existing code base. It is crucial that the tools used in the CI environment work smoothly without significant problems so that new builds of the software can frequently be integrated. Using CI in the development of embedded software has its own challenges. Some of the things that must be taken into consideration are limited amount of test environments, build and test times and wide range of different hardware [2]. Testing embedded software at system level, requires software to be tested in hardware, because to find defects that would happen in production, the test environment must be as close as possible to the production environment [1].

Hardware used in the test environments is in their own laboratory which is managed by the CI team. The CI team is responsible for performing maintenance related tasks in the laboratory which might include building target configurations out of the hardware in the laboratory, updating information in the reservation system and replacing broken hardware. To ensure that CI operates smoothly, there is an application called ReserveTool which provides an interface for reserving test environments (target configurations) from the laboratory.

The platform software continuous integration process (PSCI) is the machinery responsible for the continuous integration (section 2.2) and continuous delivery (section 2.4) of the platform software. It consists of various of different CI servers, version control systems and other applications. ReserveTool is used by PSCI for reserving test environments for tests that are performed in hardware.

Platform software makes use of trunk-based development [3] in its branching model. In trunk-based development, developers make commits (changes to the code) frequently to the same branch called a trunk instead of using several development branches which are then later merged to a single branch. In platform software, all the new feature development and new hardware support are done in the trunk. Use of trunk-based development in platform software, places a lot of strain on the CI system and with it, to ReserveTool, as there are large amounts of commits daily and each of them must be tested.

An overview of the PSCI environment and its flow is presented in Figure 1. The platform software consists of four different system components: 1, 2, 3 and 4. There are five different CI servers that operate in the system, one for each system component and one for the release of platform software. The first step in PSCI process is the code review in Gerrit [4], where the code change is reviewed by other developers and verified by a verification CI server which runs a small subset of tests that are supposed to find most obvious errors in the code change. The code change can only be merged with the approval of both verification CI and other developers. After the code change has been merged, the new build of the system component software which has

the code change is tested in its respected system component CI. When the tests have passed in the system component CI, a new system component release is made with a release tool, the binaries of the system component build are saved to a network file system (NFS) and information about the build is saved to a CI database application. System component release is stored in a version control system and it contains release notes about what was made in the code change among with ready-made downloadable packages. Next part in the PSCI is the platform software release CI process which uses new system component builds to combine them to a new build of platform software release which will then be tested at a system level. After the tests have passed, a new release candidate is provided to release management team which then decides which release candidate they promote as a new platform software release.

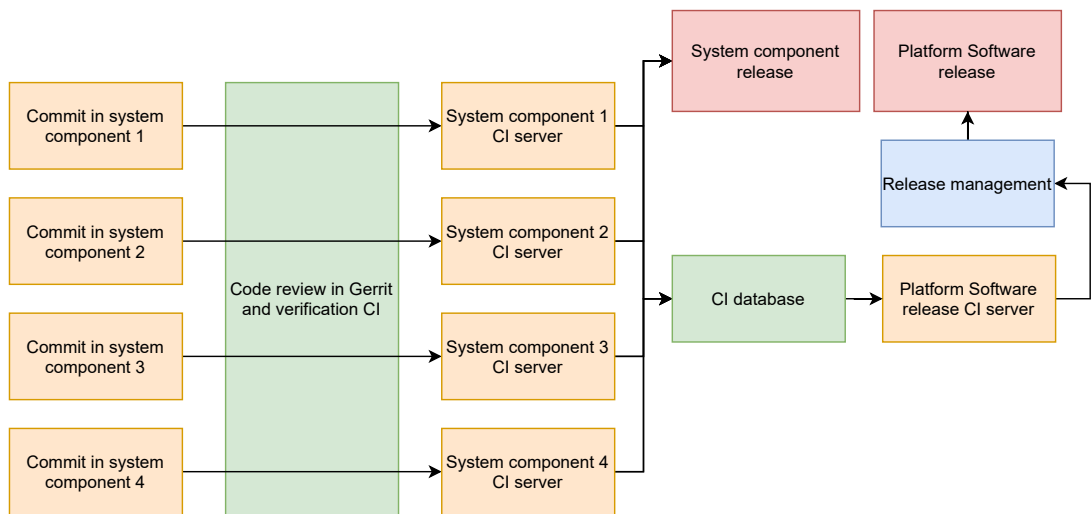


Figure 1. Description how commits proceed in the PSCI.

1.1. Motivation

ReserveTool is a web-based application which responsibilities include cataloguing hardware in the laboratory, providing an interface for the reservation of test environments, and statistical data about the usage of the hardware. The goal of this thesis work is to improve the usability and reliability of ReserveTool, the utilization rate of the hardware in ReserveTool and provide support for new features such as dynamic target configurations, an accurate hardware inventory and implicit reservation using pools and labels.

During this thesis, requirements for the new reservation application were examined and it was determined that a new application called ReserveTool 2.0 should be developed. The database of ReserveTool 2.0 was designed in a way that information about each individual hardware used in the target configurations is stored in the database among with the connections between different hardware. Both web UI and REST API were designed to provide functionality for managing database objects and reserving configurations. Usability of the web UI was evaluated with a user experiment.

1.2. Structure of the Thesis

Chapter 2 talks about what is software testing and its purpose, what is continuous integration, and how continuous integration principles work in embedded software environment.

Chapter 3 compares different options for the reservation system that were available and tells the reason why developing ReserveTool 2.0 was the chosen approach to fix the limitations of ReserveTool 1.0. The chapter also gives an overview about the usage of ReserveTool 1.0 and its limitations.

Chapter 4 describes the implementation of ReserveTool 2.0.

Chapter 5 Reviews what was achieved during the thesis, describes a user experiment and its results. Finally, the future development of ReserveTool 2.0 is discussed.

Chapter 6 Tells the summary of the thesis.

2. SOFTWARE TESTING AND CONTINUOUS INTEGRATION

To ensure that software does only what it is designed to do, it must be tested. It is estimated that about half of the time and expenses used in software development are spent in testing the developed software or system [5]. Consortium for information & software quality (CISQ) estimated in their report on the cost of poor software quality in the US [6], that in 2020 it amounted to \$2.08 trillion. The largest contributor to such a huge number was operational software failures which constituted to \$1.56 trillion.

There is a universal rule in testing which states that it is impossible to find all the errors and defects from the software with testing as there is never enough time and resources to do that [7 p.3]. Because, to find all the possible errors and defects, every possible path the software could take would have to be tested with every possible input there is [5 p.9-14]. For this reason, different strategies and choices about the software testing must be made.

CI is a development practice where changes to a existing code base are integrated frequently and verified by automated build and tests [1]. With the use of CI, problems caused by a separate integration phase in the software development can be avoided.

This chapter first investigates software testing, different software testing strategies and the four distinct software testing levels. After that, topics under CI such as how CI operates, what are its benefits, how testing is performed in CI and how does CI work in embedded system environment are explored. At the end of the chapter, a summary about how the explored topics are related to ReserveTool is given.

2.1. Software Testing

Purpose of the software testing is not to make software look perfect or completely free of errors. It is to find as many errors and bugs as possible with a finite number of resources [5 p.5-9]. This results in a trade-off between the quality of the software testing and costs in resources such as time and money. Software testing can be divided into two different testing strategies: black box testing and white box testing [5 p.9].

In black box testing, the program is viewed as a black box where its internal structure and behaviour are not known. Only the input and output of the program are examined and tests are designed based on the specification of the program. Because it is impossible to exhaustively test the program with all the possible inputs, the goal should be to maximize the number of errors found with finite amount of test cases [5 p.10-11]. According to Jorgensen [8 p.7], specification-based testing (black box testing) offers two main advantages for test cases:

1. Test cases are independent of the implementation of the program which makes them to withstand changes to the implementation.
2. Test cases can be developed parallel with the implementation reducing the duration of development intervals.

Jorgensen also points out few negative things about black box testing which are that it frequently suffers from significant redundancy found in test cases and from the possibility that there are gaps of untested software. Moreover, with only the

specification it is hard to create tests for unspecified behaviour as the tester must rely on their imagination. Picture illustrating black-box testing is shown in Figure 2.

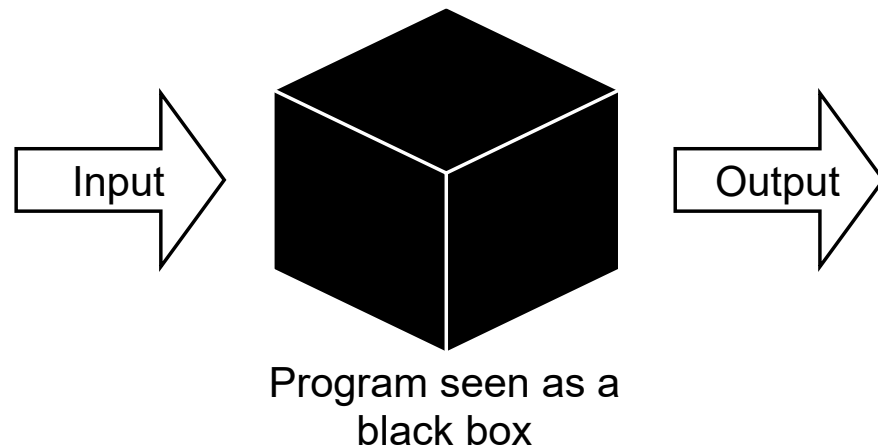


Figure 2. In black box testing, the program is thought as a black box where its internal structure and behaviour are not known. Tests are made and measured using the specification of what the program should output given the specific inputs.

White box testing allows the test designer to look at the program's internal structure and logic when designing tests for the program. Access to the internal structure of the program allows the test designer to see all the different paths the program can take and test them. Even if all those paths could be tested which is highly unlikely, it still would not guarantee that all the errors would be found, because some errors in the same path could happen with specific inputs. Picture illustrating white box testing is presented in Figure 3. White box testing also has the risk of making tests depend too much on the program's code and ignoring the specification of what the program should do. [5 p.11-13]

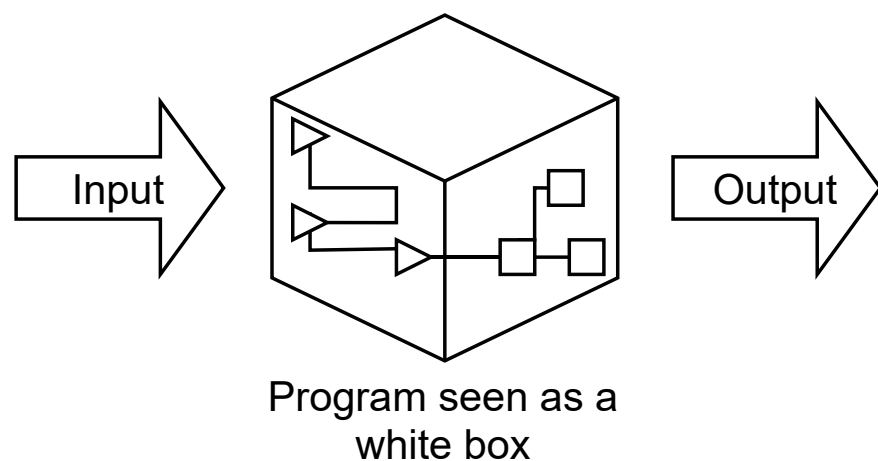


Figure 3. In white box testing, program's internal structure and behaviour are known which allows the tester to create tests for all the paths the program can take.

Generally, it is effective to make use of principles inside both black box and white box strategies to form a reasonable compromise between resources used for testing and errors found [5 p.90]. The resulting combination from the white box and black box testing can be called a grey box testing, where the tester has partial knowledge of the internal workings of the program but not the full source code of the program [9]. Partial knowledge of the internal structure allows a tester to design intelligent test scenarios that touch on things like exception handling, communication protocols and data types [10].

2.1.1. Different Software Testing Levels

Software testing can be divided into four different levels: unit testing, integration testing, system testing and acceptance testing [11 p.16]. The hierarchical relationship between different software testing levels is shown in Figure 4.

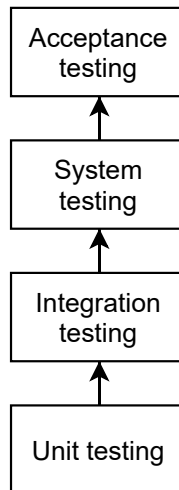


Figure 4. Software testing levels can be arranged in hierarchical order from the highest (acceptance) to the lowest (unit) level of testing.

Unit testing

Unit testing also known as the module testing is the lowest level of testing where the testing is focused on the small building blocks of the program rather than the program which makes the pinpointing of faults easy, as the errors found by the tests are contained in their relative units or modules [5 p.91]. According to Jorgensen [8 p. 78], there is not a single rule that states what constitutes as the unit in the unit testing. It can be a function, a class, a single procedure etc., and it depends on several factors such as the type of software and the programming language the software is implemented in. It is probably best defined by the organization implementing the software. Unit testing utilizes white box testing techniques [11 p.158].

Integration testing

Integration testing is the next level above the unit testing. In integration testing, individual units or modules are systematically put together while tests are performed on the interfaces between the units [11 p.158-159]. According to Naik et al., [11 p.163] integration testing includes both the black box and white box testing approaches.

There is some discussion about whether the integration testing should be considered as its own software testing level. Already back in 2004 Myers et al. [5 p.129] did not consider integration testing as its own testing level by stating that integration testing is part of the unit testing (module testing) when incremental module testing strategy is used. The close relationship between integration and unit testing is further emphasized by a study made by Trautsch et al. [12] which found that there was no significant difference between the defects found with integration testing and unit testing as the unit tests were also capable of detecting interface defects.

System testing

System testing is the level above integration testing. Purpose of system testing is to compare the system with its original objectives [5 p.130]. Jorgensen [8 p.253] states that system level testing is the closest level to everyday experience where we evaluate products based on our expectations and not based on some specification or standard. System testing includes wide variety of testing such as functionality testing, security testing, robustness testing, load testing, stability testing, stress testing, performance testing and reliability testing [11 p.17].

Myers [5 p.143] emphasizes that system testing should not be performed by programmers who coded the program or by the organization that is responsible for the development of the program. This stems from two reasons: the system testing should be performed from the viewpoint of the end user and development organization is limited by psychological ties to the program which limits their ability demonstrate that program does not meet its objectives.

In a survey conducted by Kassab et al. [13], it was found that system level testing was the most common level of testing used in the projects conducted by industry professionals with functionality testing being the most prevalent aspect of system testing.

Acceptance testing

Acceptance testing is the final level above the system testing. It is usually performed by the customer or by the end user with the purpose of comparing the developed program with its initial requirements. If there is a contract between the customer and the company developing the program, acceptance tests are done by comparing the program's operation with the original contract. [5 p.144]

Naik et al. [11 p.451] divides acceptance testing into two categories: user acceptance testing and business acceptance testing. User acceptance testing is carried out by the customer to ensure that system satisfies requirements that were set by contract. On the other hand, business acceptance testing is performed by the development organization to ensure that the system will pass the user acceptance testing.

2.1.2. Regression Testing

In regression testing, the same tests are always ran to the software after it has been changed. This ensures that the newly introduced or changed parts of the software do not interfere with the existing ones and the software works as it should [14, 15, 5 p.18]. Regression testing is commonly performed in CI to confirm that the change was integrated properly [16]. In the PSCI, all the testing that is done can be considered regression testing.

According to Wong et al. [14], regression testing becomes more expensive when the time goes on and software evolves, as old tests are rarely discarded, and new ones are introduced. These costs can be reduced with different techniques for regression test selection such as modification-based test selection where only tests where new and old versions of the software produce different outputs are performed. Elbaum et al. [17] also introduced other techniques that can improve test selection such as continuous regression test selection and continuous test suite prioritization.

2.1.3. Test-Driven Development (TDD)

In traditional software development, tests for implemented features are written after the code has been written. Test-driven development does the opposite of that and tests for the feature are written prior to coding it. As the name of the practice suggests, doing tests prior to coding the program drives the design of the program. Having tests that can be run during the development progress helps to produce a cleaner design, because the code can be easily refactored, as the programmer has the confidence of knowing the program works, if the tests pass. [18]

2.2. Continuous Integration

At the beginning of Martin Fowler's famous article about CI [1], Fowler describes the traditional integration of software with a story about a company which had a software project that had been integrating for several months with no estimation about how long the integration would take. Fowler then points out that the integration being long and unpredictable process was a common story between software projects back then, and this lengthy process can be avoided with the use of CI.

According to Fowler, in CI, members of a development team integrate their changes to a shared code repository frequently, usually multiple times each day. Each integration is verified by an automated build which includes testing to find integration problems as soon as possible. Figure 5 shows how a typical CI might operate:

1. A developer commits changes to a shared code repository.
2. A CI server polls the code repository and notices the change.
3. The CI server does automated build and testing of the change.
4. After the tests are done, feedback about the result of the build is sent to the developer.

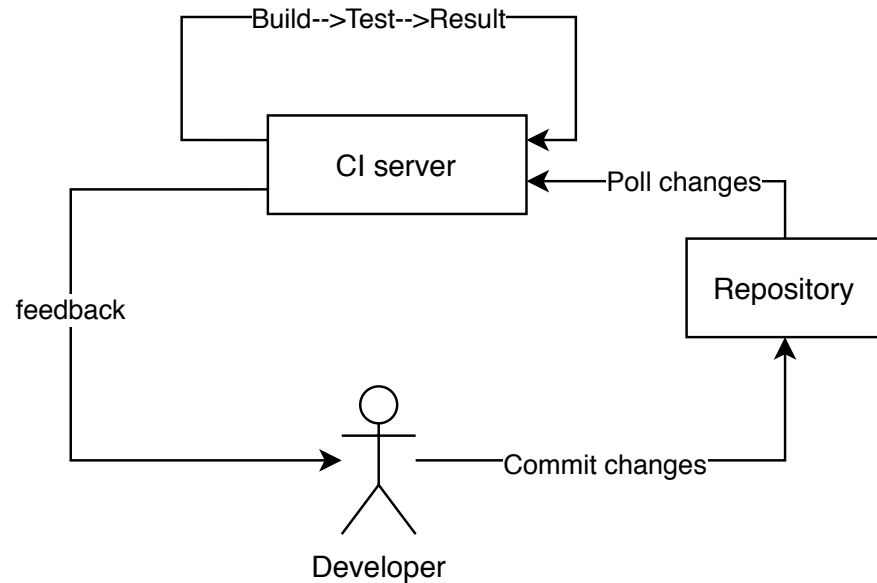


Figure 5. Description of how a typical CI operates.

There are several benefits that usage of CI introduces:

- **Reduced risks:** According to Fowler [1], without CI there is no guarantee on how long the integration process of the development cycle might take. With CI, this unpredictability can be eliminated as there is always a working version of the software and there is knowledge about what parts are working and what are not.
- **Rapid feedback:** Through CI, feedback about the state of the software project can be learned several times a day. This can reduce the time spent finding and fixing bugs or problems. [19]
- **Frequent deployment:** With CI, deployable software can be generated at any given time, because software can be proven to work after it has passed the tests [20 p.56].
- **Reduces repetitive processes:** Using automated CI, various of repetitive processes such as code compilation, testing, database integration can be automated which frees people to do higher value work [19 p.30].
- **Greater product confidence:** When, utilizing CI practices, developers have greater confidence to make changes, as they know that every build of the software is verified by testing, and with frequent interactions, CI system can inform them quickly if something goes wrong in their code change [19 p.32].

In their study about how adopting CI impacted different software projects, Zhao et al. [21] observed increase in quality and amount of automated testing as well as increase in amount of commits which was to be expected with the "integrate frequently" guideline of the CI. In a similar study by Vasilescu et al. [22], it was noticed that after adopting CI, there were fewer pull requests rejected and an increase in amount of bugs found by the core developers of the software project which overall increased the quality of the

software. Study on CI usage in open source projects by Hilton et al. [23] observed that projects that use CI release more than twice as often as those that do not.

2.2.1. Testing in CI

To ensure that broken code does not get committed to the CI, developers should run a private build of the software locally which emulates the integration build [19 p.42]. This provides a safety net for the CI and reduces the number of broken builds.

Building and testing a change in CI should be reasonably fast so that it does not slow down development significantly. In Kent Beck's famous book about extreme programming [24 p.110] he stated build time of ten minutes to be ideal. According to Duvall et al. [19 p.91], most of the build time in CI is taken by the automated testing. In a study about the long duration of CI builds Ghaleb et al. [25] found several factors that often constitute to the long build times in CI:

- Triggering builds during the day or on weekdays. This is the most likely due to CI system having a limited number of resources and there being several simultaneous builds.
- Builds are not marked as finished even though all the required jobs are finished.
- Rerunning failed jobs under the build to keep stable build status.

As minimizing build time is important, new techniques for reducing build time are developed. One example of this is SmartBuildSkip framework developed by Jin et al. [26] which uses machine learning to determine which build should be skipped as their result would be success. Once a build fails, all the subsequent builds are executed until the build result is success.

A recent survey by Garousi et al. [27] indicates that test automation is the aspect of software testing which proposes most challenges to industry professionals. According to Fowler M. [1], tests performed in CI should be automated tests that check the functionality of the code after a change. Testing in CI can be considered being mostly regression testing as the same set of tests or a subset of tests are performed to each a new build of the software. This also results in that bugs and problems are found more easily, as the developer has confidence that the build, they started their work on, was stable and had passed the tests.

2.3. CI for Embedded Software

In the early days of CI back in the year 2007, Karlesky et al. [28] found that embedded software industry's general perception was that implementing the CI and practices of TDD to embedded software development is too difficult due to software being so close to the hardware and limited amount of test environments. In their article, they proved that implementing CI in embedded software development is possible and provides benefits such as reduced integration time and making it easy to pinpoint the source of unexpected behavior in the system.

Nevertheless, there are certain challenges due to the nature of embedded software that it poses to the CI such as expensive [29] test environments, limited amount of test environments and long build and test times for tightly coupled systems [2].

2.3.1. Testing in a Target Hardware

Fowler M. [1] emphasized that to find problems and bugs that would appear in production, the software must be built and tested in a target environment that is as close as possible to the production environment. This is because if the test environment differs from the production environment, tests done in each environment might produce different results which makes the selection of test environment very important. Rapid feedback is also one of the essential things at the heart of CI which ensures that problems and failures do not propagate and cause other failures [19 p.203-204].

According to Broekman [7 p.199-207], in the prototyping stage of embedded software development there are five different test levels:

- software unit tests
- software integration tests
- hardware/software integration tests
- system integration tests
- environmental tests

Only the first two levels can be performed without the target hardware. In the hardware/software integration tests, software is loaded in the hardware memory. In a system integration test, all the hardware parts of the embedded system are brought together, as the purpose is to verify the correct operation of the complete embedded system. Environmental tests are used to test how the embedded system influences its environment and how the embedded system reacts its surrounding conditions like the temperature, humidity, shock, and vibrations. After the prototyping stage, comes the pre-production stage where real target hardware is also used.

2.3.2. Simulated Hardware in CI

Hardware can be simulated using a virtual platform. The virtual platform can run the unmodified binaries of the software that would be ran on a real system and perform well when simulating hardware – software interface [30 p.2]. Broekman et al. [7 p.193-195] introduces a simulation stage as the first stage of development in embedded systems. In the simulation stage testing is done using simulated hardware. As the development progresses the simulated hardware is gradually replaced by real hardware. Engblom [31] listed several advantages that simulated hardware provides for CI in embedded system environment:

- The state can be saved. By saving the state virtual hardware significant amount of time can be saved. For example, state where the system has already been

booted could be saved and it could be reused as a starting point for the tests. Normal hardware would have to be booted to remove the effects of previous tests. [31].

- It has been seen that limited amount of test environments is one of the challenges in CI of embedded software [2]. With virtual hardware, this is not a problem, since it can be simulated on any server that is capable of running the simulated hardware. Also, when developing software for a new variant of hardware, their amount can be very limited or they might not even have been manufactured yet. Using a virtual platform can start software development for the new product earlier [30 p.5].
- There is more control when using simulated hardware; a simulator program will not go unresponsive if there is a failure during the testing [31].
- All the different hardware configurations can be tested because it does not have the physical limitations of the normal hardware [31].

However, there are some properties that cannot be tested properly on simulated hardware like internal buses, clocks, pipelines, and caches. [30 p.2] For that reason, system level tests need to be run on real hardware. After all, as said by Mårtensson et al. [2] it cannot be ensured that same tests will pass for a build that runs on real hardware than for a build that runs on simulated environment.

2.4. Continuous Delivery

According to Humble et al. [20 p.105], just having a CI is not enough, because CI mostly focuses on development teams it leads to too long feedback cycle between development team and operations team who are responsible for releasing the software. This can be fixed by further automating the release process by introducing continuous delivery (CD).

CD is a software engineering approach which can deliver software changes into production or into the hands of the users in a quick, safe and sustainable way [32]. In essence, operations team can deploy a build of the software into production with a push of a button [20 p.106]. This is made possible with the use of CI which ensures that the code is in a state where it can always be deployed. The process of CD can be visualized with a deployment pipeline which shows the steps that code change takes on its way to production. As the code change progresses through the pipeline, confidence that it will work in production increases as the environment becomes more production like with each further stage [20 p.108]. A typical CD pipeline is presented in Figure 6.

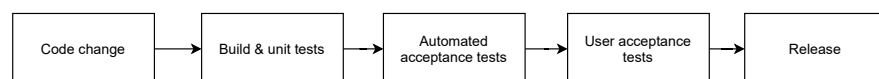


Figure 6. Description of a typical CD pipeline. All the steps in the pipeline are automated except the final step of releasing the software which is done manually.

The pipeline starts when a code change is made to a VCS by CI, triggering a new instance of the pipeline. A new build of the software is made and unit tests are performed. After that automated acceptance testing follows. Once that stage is done, it is followed by user acceptance testing. All the aforementioned stages are automated up until the final stage which is the release stage where operations team can choose to make a release when they find a release candidate, they deem suitable. Objective of the pipeline is to eliminate unfit release candidates as soon as possible in the pipeline so that there is rapid feedback on the root cause of the failure to the team [20 p.108].

Continuous integration is often combined with continuous delivery in an abbreviation called CI/CD. Although CD in the abbreviation can also refer to continuous deployment. It is different from the continuous delivery in a way that continuous deployment goes one step further where new software is automatically deployed to the customers [33].

2.5. Use of Continuous Principles in the ReserveTool

ReserveTool is an essential part of the continuous integration and delivery of the platform software. It provides the test environments (physical hardware or virtual hardware) to automated CI and to developers who run the private builds of the software before committing code changes to CI. It is important that the ReserveTool ensures and enables the continuous development of the platform software.

3. REQUIREMENTS

Purpose of this chapter is to give the reader an overview how ReserveTool 1.0 is used in the PSCI environment, what are the problems the system in place has and what are the possible solutions. This is done by first examining ReserveTool 1.0 and comparing its features with different reservation system candidates used in the company, including ReserveTool 2.0. Finally, a reasoning why the development of ReserveTool 2.0 was the chosen approach, is presented.

3.1. ReserveTool 1.0 and Its Use

ReserveTool 1.0 is the current reservation system that is used in the PSCI environment by two system component CIs, platform software release CI and by the developers of previously mentioned two system components. The users of ReserveTool 1.0 can be divided in three different groups:

- **Admins** are in the charge of maintaining the ReserveTool. They have physical access to the laboratory where the different hardware is stored. Their typical tasks include making physical connections between hardware, replacing broken hardware, building new target configurations, and registering them to ReserveTool.
- **Automated CI servers** use ReserveTool's REST API for finding suitable target configurations for tests that need to be performed in hardware. This user group includes two system component CI servers and the platform software release CI server. CI servers use a Python client that ReserveTool provides to communicate with ReserveTool's REST API. Figure 7 presents a flowchart on how a typical CI job reserves a target configuration for testing a new software build. Time out for the test job on reserving suitable target is usually set at 15 minutes.
- **Developers** reserve target configurations to test their changes before they are committed to the system component CI. Developers are also responsible for providing new test cases when they are needed which could be due to a new feature or hardware being introduced. These new test cases must be tested prior to their use in the CI. Typical use case for the developer involves reserving a configuration, powering it on and running tests on it. Developers mainly use ReserveTool through its web UI.

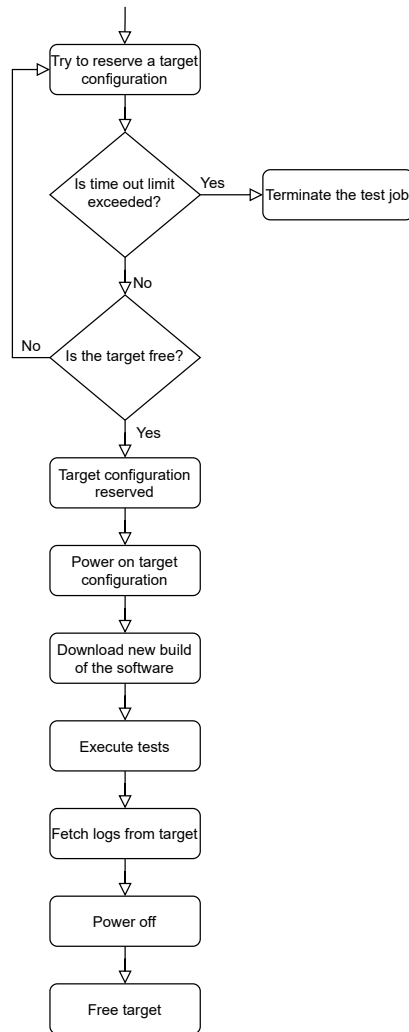


Figure 7. Flowchart detailing how a continuous integration test job uses target configurations for testing the new builds of software in the PSCI environment.

A use case diagram which displays the use cases of different user groups in ReserveTool 1.0 is presented in Figure 8. Developers and automated CI servers are the end users of ReserveTool 1.0 who reserve and queue target configurations (test environments) to perform tests in hardware. Use cases for admin users consist of management of target configurations, pools and hardware inventory, and browsing of statistical data about the utilization of target configurations.

Target configurations in the ReserveTool 1.0 are divided into different pools which are groupings for configurations that determine the users who can reserve the configurations belonging to that pool. In ReserveTool, different CI servers and developer teams have their own pools of configurations from where to reserve. The configurations are either behind a virtual machine or a unique IP address depending on if the configuration is meant for the use of CI or developers. Configurations used by the CI are behind virtual machines because it makes configuring the targets which to reserve easier from the side of a CI server. Configurations that are meant for developers

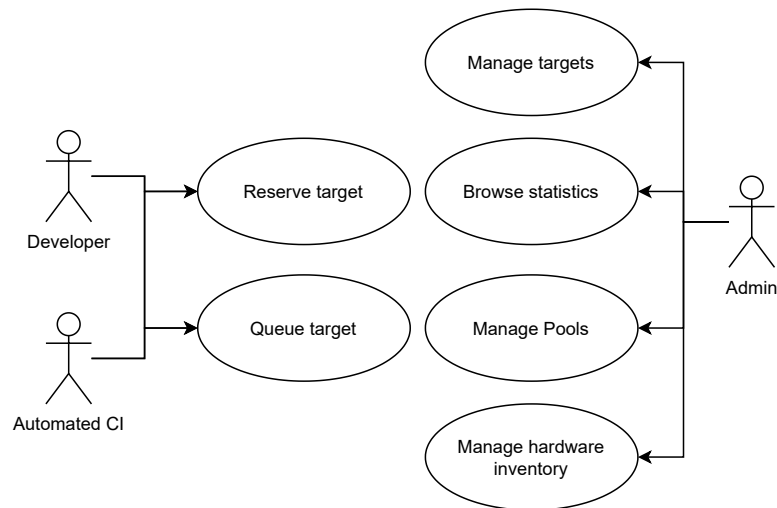


Figure 8. The use cases for ReserveTool 1.0.

are behind IP addresses, because it is better for the developers to have straight access to the configuration as it gives them more control over the hardware.

3.2. Limitations of ReserveTool 1.0

ReserveTool 1.0 has a wide variety of limitations that hinder its development and the performance of the PSCI environment. The main limitations being the poor information about the hardware inventory, the hardware utilization rate and lack of support for dynamic hardware configurations. These limitations are examined more thoroughly in next subsections.

3.2.1. Hardware Inventory

ReserveTool 1.0 does not have a proper hardware inventory. This poses a problem to the maintainers of the laboratory because they do not have accurate and up to date information about what hardware the laboratory has. This problem is amplified even more due to old hardware breaking and new hardware coming in. Currently, ReserveTool 1.0 relies on a script that reads the hardware information from the target configurations that are registered in the system. The solution does not give accurate information for a few reasons:

- The script can read information only from a target configuration that is powered on.
- Some hardware types such as power switches for example, do not support reading information by a script.
- Hardware currently not used in any target configuration cannot be reached by the script.

These limitations indicate that non-automated approach could give much more accurate information but would require to be manually updated by the maintainers to keep it up to date.

3.2.2. Hardware Utilization Rate

Each day there are multiple code changes that go through the PSCI and they need to be tested at the system level (in hardware). Because there is limited amount of test environments (target configurations), all the tests cannot be run parallel, and queues have to be made. If the utilization rate of the hardware is not 100%, there is always some savings to be had by increasing it. Utilization of the hardware in the ReserveTool 1.0 can be measured by looking at the reservation time of different target configurations.

The utilization data that can be measured in ReserveTool 1.0 has one problem; it does not know the amount of time the reserved target configuration was powered on during the reservation. This is especially problematic for the measuring utilization of target configurations allocated to developers as they tend to be reserved for longer periods of time than the target configurations reserved by CI servers and are not necessarily powered on the whole time of the reservation. Utilization data would be improved greatly by considering the amount of time the configuration is powered on.

3.2.3. No Support for Dynamic Hardware Configurations

The database of ReserveTool 1.0 has models only for the target configurations, and not for the individual hardware of which the target configurations are built from. Because of that there is no support to dynamically create target configurations from different hardware, as it is not possible to save the information about physical connections between different hardware. For example, when considering a fiber switch which is hardware that handles fiber connections to multiple hardware, information about which port of the fiber switch has which hardware connected, could be used to create dynamic hardware configurations in ReserveTool without having to move them physically.

3.2.4. Usability of the System

Over the lifetime of ReserveTool 1.0, several small usability limitations have emerged. These limitations are listed below:

- There is no option to extend or repeat a reservation.
- There is no option to schedule a reservation ahead of time.
- User cannot reserve a target just by stating pool and labels, they must explicitly select a target configuration.
- There is no permission handling so regular users can accidentally break something.

- Some parts of the web UI feel slow, especially views which list large amount of target configurations.

3.3. Different Solutions for the Reservation System

There are two other reservation systems in the company use that could have been chosen as the reservation system for the PSCI environment: Velocity [34] and TAToo. Therefore, there was a choice between one of them or development of ReserveTool 2.0. Each of these options provided a different set of features and limitations. Table 1 shows a comparison between the features offered by ReserveTool 1.0, planned ReserveTool 2.0, TAToo and Velocity. ReserveTool 1.0 is shown in the Table 1 to give perspective on what features the current reservation system offers and what features it lacks.

3.3.1. Target Administration Tool (TAToo)

TAToo is an internally developed target reservation tool at Nokia. TAToo's two main features are target reservation and a hardware inventory. The normal users of TAToo can reserve targets that are listed at a target booking table. The target can be a cluster which consists of two or more singular targets or just a singular target. Management of the targets is handled by the admin users.

TAToo can be used through the web UI or through the API which provides methods for reading target information, reserving targets and freeing targets. TAToo has an automatic target repairing system which checks the statuses of the targets in the tool. If the status is "to be checked", it starts a job which will then try to repair the target.

3.3.2. Velocity

Velocity is a cloud-based warehouse management system that is sold by following software as a service model (SaaS) [34]. Some business lines at Nokia use a customized version of Velocity in their laboratory setting for reserving hardware and keeping track of the hardware inventory in the laboratory. Velocity calls each hardware that is registered to the system as a concrete resource. The managing of these concrete resources is done by admin users who also have access to the laboratory to make physical connections. Normal users can build abstract resources that describe conditions on which to select concrete resources. Velocity also lets its users to build collections of resources called topologies. Both resources and topologies can be reserved by the normal users.

Velocity has different user types: normal users, admin users and unauthenticated users. Normal users are the users who use the system to reserve resources or topologies to their use. Admin users have several different roles in the system such as management of concrete resources, reservations, and users. Unauthenticated users cannot make reservations and have only read-only permissions to certain places in the system.

Reservations in the velocity can be scheduled to happen immediately or later. There is also an option to repeat the reservation hourly, daily, weekly, or monthly. User can

Table 1. Comparison of ReserveTool 1.0, ReserveTool 2.0, TAToo and Velocity

Feature	ReserveTool 1.0	ReserveTool 2.0	TAToo	Velocity
Reserve target configurations	yes	yes	yes	yes
Support for dynamic target configurations	no	yes	no	yes
Must pay for license	no	no	no	yes
Easy to implement new features	yes	yes	no	no
Hardware inventory	partial	yes	yes	yes
Power on targets	yes	yes	no	yes
API support	yes	yes	partial	yes
Automatic repair	no	yes	yes	-
End-user can build their own configurations	no	no	no	yes
Interactive web UI for building configurations	no	no	no	yes
Selection of tasks to run in start and end of reservation	no	no	no	yes

also select different automation assets that will be run during start-up and teardown of the reservation. For example, an automation asset could be a script that handles powering on the hardware or setting up a test environment.

Velocity has a REST API which provides an extensive interface for external applications that use the system. Velocity also provides a python library for communication with the REST API which makes it simple to integrate it in python projects.

3.3.3. *Choosing the Reservation System*

After exploring different solutions within the CI team, it was chosen that ReserveTool 2.0 should be developed as the reservation system for the PSCI environment. This was due to several reasons:

- Easy to implement new features as the development of ReserveTool is done within the team.
- No need to pay for a license to use the system like in Velocity.
- System is flexible so that it can be modified to better fit the needs of the PSCI.
- There is more control over the reservation system.

Ability to develop ReserveTool 2.0 from the ground up fixes a lot of the fundamental problems that exist in design of ReserveTool 1.0, e.g., the problem with the hardware inventory which is explained in subsection 3.2.1. ReserveTool 2.0 will introduce new features such as support for dynamic hardware configurations by providing the ability to save information about connections between the hardware. Old features that exist in ReserveTool 1.0 will also be improved and made more robust. Data collection for statistics such as the utilization rate will be improved to provide more meaningful data.

4. IMPLEMENTATION OF RESERVETOOL 2.0

This chapter describes different parts of the implementation of ReserveTool 2.0: the database, authentication and authorization, web UI, and REST API. In the next section, an overview and reasoning for different technologies used in ReserveTool 2.0 is given.

4.1. Technologies Used in the Implementation

As ReserveTool 2.0 is a web application, it was essential to choose some web framework to ease the development of the application. Django [35], which is a Python¹ based web framework, was chosen for this purpose. The main reason for choosing Django was that it was used with ReserveTool 1.0 which provided a few advantages:

- **Reuse of code** Some code can be reused between the different versions of the ReserveTool, e.g., powering modules which are used for powering the configurations.
- **Familiarity** The CI team, who are responsible for maintaining ReserveTool, are familiar with Django framework which leads to easier future development as it does not require learning a new framework or programming language.

REST API for ReserveTool 2.0 was built using Django Rest framework² (DRF) which is a powerful toolkit that provides a framework for building REST APIs on top of Django applications. Decision for choosing DRF to implement the REST API for ReserveTool 2.0 opposed to creating REST API functionality to the views of the web UI, was because DRF provides a lot of built in functionality such as browseable API and view sets supporting different HTTP methods which reduce a lot of development time. Keeping the REST API separate from the web UI views in the ReserveTool 2.0 also makes the code cleaner and easier to maintain. Documentation of the ReserveTool 2.0 REST API is generated with a library called drf-yasg³. Drf-yasg generates documentation that follows OpenAPI specification 2.0 (swagger)⁴ automatically based on the source code of the implemented DRF API.

Front-end of the ReserveTool 2.0 web UI was stylized using a framework called Bootstrap⁵ with a goal in mind to minimize the usage custom CSS to keep the UI uniform and easy to maintain. Bootstrap was also used in ReserveTool 1.0 which makes styling of the UI familiar to the users of ReserveTool 1.0. The dynamic aspects of the web UI utilize programming language JavaScript and its library jQuery⁶.

¹<https://www.python.org/>

²<https://www.django-rest-framework.org/>

³<https://github.com/axnsan12/drf-yasg>

⁴<https://swagger.io/specification/v2/0>

⁵<https://getbootstrap.com/>

⁶<https://jquery.com/>

4.2. Database

For interaction with the database, Django provides an object-relational mapper (ORM) which is a layer between a relational database and the application. ORM automates the transfer of data from relational database to python objects that can be used in the application code [36]. This provides us with high-level abstraction on top of the relational database and allows us to create the database using python classes, instead of complex SQL statements. These classes are called models and represent tables found in the relational database. Writing, reading, editing, and deleting data is done through the database models.

This section describes the structure of ReserveTool 2.0 database. The database consists of 18 different models with varying complexity. Entity-relationship (ER) diagram of the database is presented in Figure 9.

The next subsections in this section describe each database model, their fields, and their relationships to other database models. Hardware and Configuration models are described more in-depth in this section because they are the two centre pieces of ReserveTool 2.0 while the rest of the models are mostly built around them.

4.2.1. Hardware Model

The Hardware model is used for storing information about hardware in the PSCI laboratory. Each piece of hardware must be registered to the database before it can be used in configurations. This makes information about the hardware inventory as accurate as it can be, as we do not have to rely on a script that only works when the hardware is powered on.

The name of the hardware is automatically made from its unit id by combining it with an incremental number which makes the name unique. The name field is used as a lookup field in the URL of the hardware in both Web UI and REST API of the ReserveTool. Hardware model has seven different types:

- **Power switch:** Hardware that handles powering of the individual hardware in the configuration. The same power switch can be used in multiple configurations at the same time.
- **Base transceiver station (BTS)** A wireless communication device which is usually the system under testing (SUT). The same BTS can only be used in one configuration at a time.
- **Network switch** A type of hardware which handles a remote connection to the configuration. The same network switch can be used in multiple configurations at a time.
- **Serial port server** Hardware which handles serial connections to the configurations. The same serial port server can be used in multiple target configurations at a time.

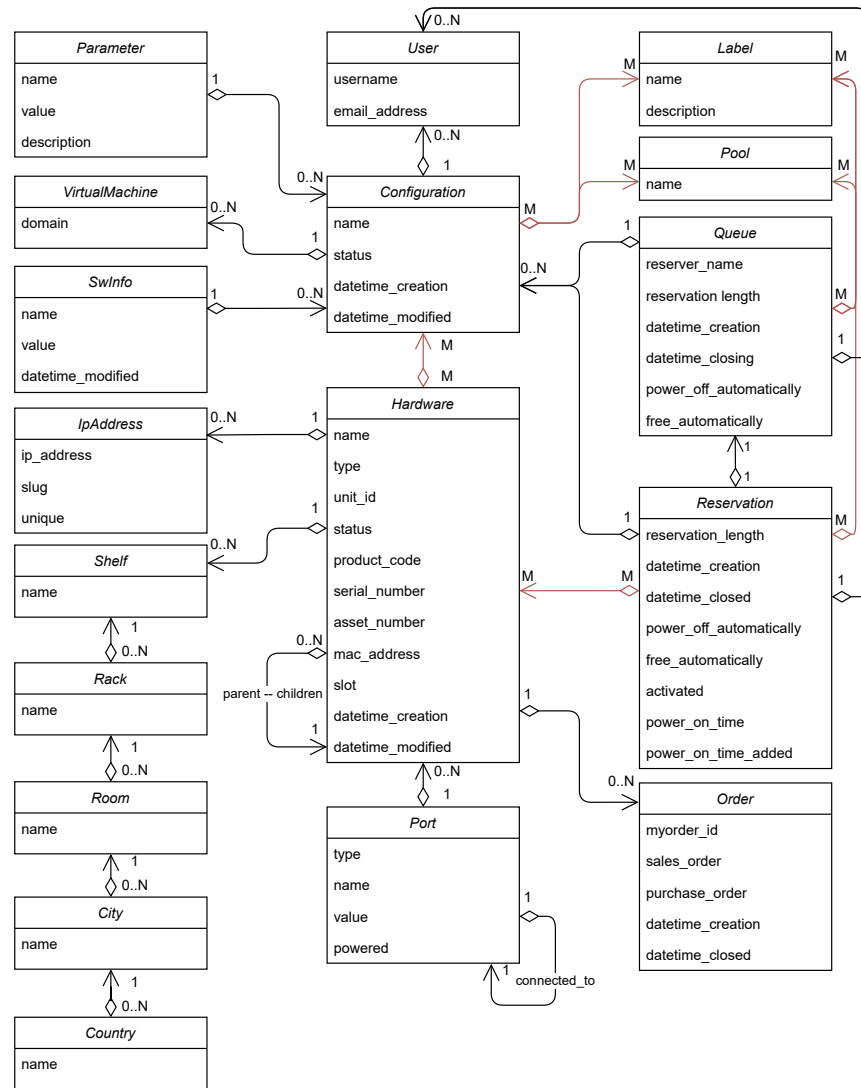


Figure 9. The ER diagram of ReserveTool database. Each relationship is defined in the database model where the arrow representing the relationship starts. Many-to-many relationships are marked with a red color and annotated with a letter **M** on both ends of the relationship.

- **Simics** Is a virtual version of a BTS. It is usually the SUT in the configuration that it can be found in. The same simics can only be used in one configuration at a time.
- **Virtual environment** A hardware type that is mainly reserved for docker containers. The same virtual environment can only be used in one target configuration at a time.
- **Other** A hardware type which can be used if the above mentioned types cannot be used. For example, could be used by some highly specified custom equipment.

Hardware also has fields for the serial number, product code, asset number, MAC address. The serial number is a number that can be found on most of the hardware in the PSCI laboratory. It is attached to the product when it is manufactured and provides unique identification for it during its whole lifetime. The product code identifies the product family which the hardware belongs to. The asset number is a number that is attached to expensive equipment and used by the company accounting. The Hardware model has relationships to seven different models which are shown in Figure 10.

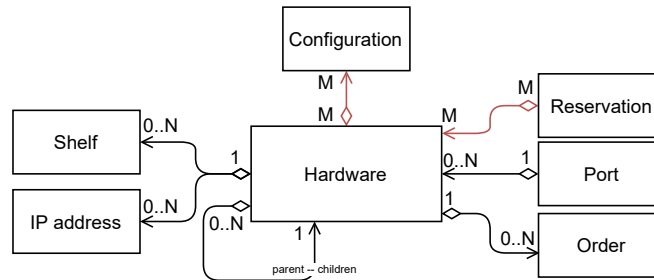


Figure 10. The ER diagram that describes the relationships of the Hardware model. Each relationship is described with an arrow that shows the direction of the relationship and labels that show the cardinality of the relationship.

The description about each relationship in the Hardware model and their role can be found in Table 2. For more detailed description about each field of the Hardware model, see Table 5 in the appendices.

Table 2. Relationships of the Hardware model.

Model	Cardinality	Description
Configuration	many to many	Hardware can be thought as a building block, from which, the configurations are made of.
Shelf	one to many	Tells the physical location of the hardware.
IP Address	one to many	The IP address allocated to the hardware.
Reservation	many to one	Reservation object that is made when a configuration that the hardware belongs to, is reserved. Allows tracking of reservation history of a specific hardware.
Order	one to many	Holds the purchasing information related to the hardware.
Port	many to one	Tells the ports that are found in the hardware. Used for mapping the physical connections between hardware. Important for finding the correct power switch port in powering of a configuration.
Hardware	one to many	Recursive relationship to Hardware model which allows to construct hierarchical relationship between different hardware. Makes creation of configurations more convenient.

4.2.2. Configuration Model

The Configuration model is used to represent configurations that are built from hardware and reserved by the users of the ReserveTool. The Configuration model has status which has four different states: available, reserved, maintenance and broken. This status is used to determine what actions the user may perform to the configuration, e.g., users can only reserve available configurations and cannot reserve broken configurations. The Configuration model is one of the centre pieces of ReserveTool 2.0 which makes it the model with the most relationships to different models. Configuration model's relationships are presented in Figure 11.

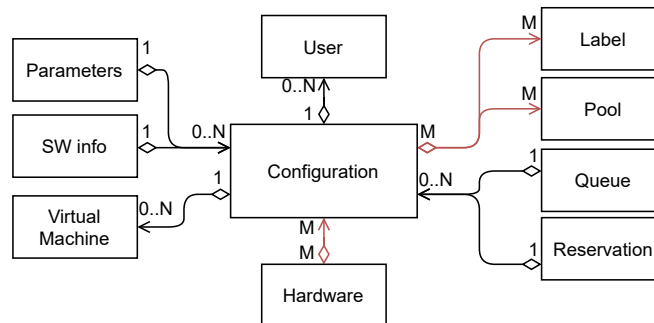


Figure 11. The ER diagram of the relationships of the configuration model.

The description about each relationship in the configuration model can be found in Table 3. For more information about each field of the configuration model see Table 4 in appendices.

4.2.3. Other Models

This section describes the rest of the database models of ReserveTool 2.0. These models are described more briefly than Configuration and Hardware models because they are not as complex and mainly just provide some functionality for them.

Reservation model

The Reservation model stores information about reservations that are made in ReserveTool 2.0. Reservation objects are made when a configuration is reserved. This happens when a user reserves an available configuration or when the user has queued reservation to a reserved configuration and the previous reservation ends. In the case where reservation is made from a queue object, the user has to activate the reservation within a certain time limit, or it will be cancelled.

Although reservation status can already be seen in the Configuration model, Reservation model stores other important information about the reservation such as the length of the reservation, the starting and ending timestamps of the reservation, if the reservation has been activated and how much time the configuration was powered on during the reservation. Time related information in the reservation model is used to compute different properties about reservation e.g., how much time is left on the

Table 3. Relationships of the Configuration model.

Model	Cardinality	Description
Hardware	many to many	Hardware objects are the main building blocks which the configuration is made of.
User	one to many	User model is used in three different relationships in the configuration model: current user that is reserving the configuration, user who created the configuration and the user who has last modified the configuration.
Label	many to many	Labels provide a quick way to describe what hardware is in the configuration.
Pool	many to many	Tells the pool which the configuration is allocated to.
Queue	many to one	Queued reservation. Used for creating a reservation object once the configuration is available.
Reservation	many to one	Every time the configuration is reserved, new reservation object is made and attached to the configuration.
SwInfo	many to one	Tells the version of the software used in the configuration.
Parameter	many to one	Manually defined parameters that the user might use while running tests in the configuration.

reservation or if the confirmation time for the reservation has been exceeded. A reservation also holds information about two settings that user may choose when making a reservation: power off the configuration automatically after the reservation and end the reservation automatically after the time runs out.

Starting and ending timestamps and power on time from past reservations are used to calculate two different statistics: the reservation rate and the power on rate. How the information from the reservation model is used to compute these statistics is explained in subsection 4.5.8. For more information about the fields of the Reservation model see Table 6 in the appendices.

Order model

The Order model is used for saving information about orders related to the hardware used in the laboratory. Purpose of the model is to make keeping track of new hardware shipments easier. Most of the information saved in the Order model comes from the company's internal purchasing tool. For more information about the fields of the Order model see Table 7 in the appendices.

Queue model

The main purpose of the Queue model is to allow queuing reservations to configurations which are already reserved. The Queue model is very similar to the Reservation model, as it contains all the information needed to create a reservation

once the queue is closed. A queue object can also be used to make a reservation even if the configuration is not reserved. This is especially useful for the automated CI as it lets the CI use only one endpoint for the reservation. More information about the fields of the Queue model can be found in Table 8 in the appendices.

Port model

The Port model is used to store information about ports that are found in the hardware. Each port has a type and value. There are five different types of ports: power, fiber, HDMI, serial, and generic ports. Value of the port identifies a specific port on hardware that has multiple ports of the same type. The Port model has a relationship with itself called `connected_to` which is used to map a physical connection between two ports in the real world. For more detailed description about the fields of the Port model, see Table 9 in appendices.

Pool model

The Pool model is used for storing information about pools. The main purpose of the pool model is to allocate the use of a certain configuration to a specific user group. For example, **developer team A** has their own pool called **developer A** which is attached to all the configurations that allocated to their use. The Pool model can also be used to assert certain settings for reserving the configurations that belong to that pool, e.g., freeing the configuration automatically after the testing or powering off the configuration automatically after ending the reservation. The detailed description of the fields is provided in the appendices in Table 10.

Label model

The Label model is used for storing information about labels. The purpose of labels is to provide information about the configuration at a high level. Labels should provide all the information that the user needs for finding a suitable configuration to reserve. One configuration can have many different labels and the same label can be in multiple configurations. Detailed description about the fields can be found in the appendices in Table 11.

VirtualMachine model

The VirtualMachine model is used for storing information about the domain names of virtual machines that are used by some of the configurations. For more detailed information about the fields see Table 12 in appendices.

IpAddress model

The IpAddress model stores the information about IP addresses that used in the ReserveTool. IP address can be attached to hardware which then tells the IP address from where the hardware can be reached. For detailed description about the fields of the IP address model see Table 13 in appendices.

SwInfo model

The SwInfo model is used to save version information about the software loaded in the configuration. The thorough description of its fields can be found in appendices in Table 14.

Parameter model

The Parameter model is used by the Configuration model to save information about manually defined parameters which are used in automated testing or for giving additional information to the user reserving the configuration. For accurate description about its fields see Table 14 in appendices.

User model

The User model⁷ is Django's built-in model for handling the users of the application. It is used in the authentication and authorization of Django applications. For more on how authentication and authorization is handled in ReserveTool 2.0 see section 4.3.

Location models

There are five different models that are used to represent the physical location of the hardware in ReserveTool 2.0: Country, City, Room, Rack and Shelf. Relationships between the models are hierarchical: Country has a relation to the city, the City has a relation to the Room and so on. With these models, accurate location for the hardware can be retrieved. For accurate descriptions of the fields of the location models see tables 16 to 20 in the appendices.

4.3. Authentication and Authorization in ReserveTool 2.0

ReserveTool 1.0 did not have any restrictions to different user groups which is not a great practice in general as the users are just people and people make mistakes. Therefore, ReserveTool 2.0 has restrictions in place for certain actions with the purpose in mind that a regular user should not be able to disrupt other user's activities in the ReserveTool in any way, e.g., accidentally powering off a configuration or a hardware that is currently reserved by some other user.

Authentication and authorization in ReserveTool are handled using Django authentication system⁸. The Django authentication system allows us to authenticate the users of ReserveTool and set up permissions for different users at a user level or at a user group level. At the core of the authentication system are the User objects which represent the users of the system. User objects are used to authenticate and authorize users when they interact with ReserveTool. If the user is authenticated, a user object is associated with every request that the user makes in ReserveTool. The User object can then be used to authorize the user's actions by checking the permissions associated with the user.

⁷<https://docs.djangoproject.com/en/3.1/ref/contrib/auth/#user-model>

⁸<https://docs.djangoproject.com/en/3.1/topics/auth/>

4.3.1. User Groups

The users of ReserveTool are divided into four different groups: admin, developers, automated CI, and unauthenticated users. Each of these groups has different sets of permissions that are discussed in the subsections under this section. The user groups only provide a set of general permissions the user belonging to the group should have. Admin users can freely set additional permissions for each individual user when needed.

Admin

Admin is reserved for the staff of the PSCI laboratory. Admin users can access the admin panel of the Django application and their actions in ReserveTool are not restricted by any permissions.

Developers

Developers are the typical end-users of ReserveTool. By default, they are only permitted to do reservation related or read-only actions in ReserveTool.

Automated CI

Permissions for the automated CI are basically the same as for the developer as the automated CI only uses ReserveTool for reservation related activities.

Unauthenticated users

Unauthenticated users are only permitted to do read-only actions such as browsing the hardware inventory or statistical data.

4.3.2. Authentication

ReserveTool uses LDAP [37] as the authentication back-end which allows users to log in to ReserveTool through the website using their company credentials. When a user first time logs into ReserveTool, User object is automatically generated to ReserveTool's database. Further log ins by the same user using their company credentials are always associated with the User object found in ReserveTool's database.

REST API of ReserveTool provides three different ways for authentication:

- **Basic authentication:** In basic authentication, the username and password are sent in the header of the HTTP request. This is not ideal in production, as you would have to send your credentials in plain text with every request.
- **Token authentication:** When a User object is created in ReserveTool, an authentication token is automatically generated for that specific User object. This token can then be used in a header of the HTTP request to authenticate the user. ReserveTool REST API provides an endpoint for obtaining the token

which requires the user to send their username and password in a POST request. Admin user can also provide a token for the user through Django's admin panel.

- **Session authentication:** Session authentication uses Django's default session back-end which is the same that is used in the web UI of ReserveTool. With session authentication cross-site request forgery (CSRF) token must be included with non-safe methods like POST, PUT, PATCH and DELETE to protect from CSRF attacks. Session authentication is typically used in ReserveTool when browsing the REST API with a browser using the interface provided by the DRF.

Because ReserveTool REST API supports basic authentication and token authentication all the requests must be done using HTTPS protocol.

4.3.3. Object-Level Permissions

Object-level permissions refer to permissions which determine whether the user should be able to perform certain actions on a specific object. In the case of ReserveTool, object-level permissions are utilized in the reservation and powering of configurations; user who has active reservation to a configuration is the only non-admin user who can end the reservation or change the power setting of the configuration.

4.4. ReserveTool 2.0 REST API

REST API is a web API which conforms to the REST architectural style [38 p.5]. According to Fielding T. [39 p.76-85] REST is a web architecture which can be derived by applying a set of constraints to the elements within the architecture:

- Separation between a client and a server.
- Communication between the client and the server must be stateless; each request to the server must contain all the information necessary to understand the request.
- Cache constraints: data within a response must be labeled to be cacheable or non-cacheable.
- Uniform interface: Four interface constraints: the identification of resources, the manipulation of resources through representations; self-descriptive messages and hypermedia as the engine of the application state.
- Layered system: Allows architecture to compose of hierarchical layers improving evolvability and reusability [39 p.46].
- Code-on-demand is an optional constraint which allows client functionality to be extended by allowing the client to download and execute code in the form of applets or scripts.

A resource is an abstraction of information in REST. This information can be a document, a service, a collection of other resources or anything that can be named. The resource identifier is used to identify a particular resource. Actions performed on the resources are done using representations. Representations can indicate the current state of the requested resource or the desired state for the requested resource. [39 p.88-91]

4.4.1. Implementation of ReserveTool 2.0 REST API

A state diagram of the ReserveTool 2.0 REST API is presented in Figure 12. The state diagram shows different resources that can be reached through the REST API and the hyperlinked relations between them. The hardware, configuration, port, and queue resources contain custom methods which are shown inside their resource representation.

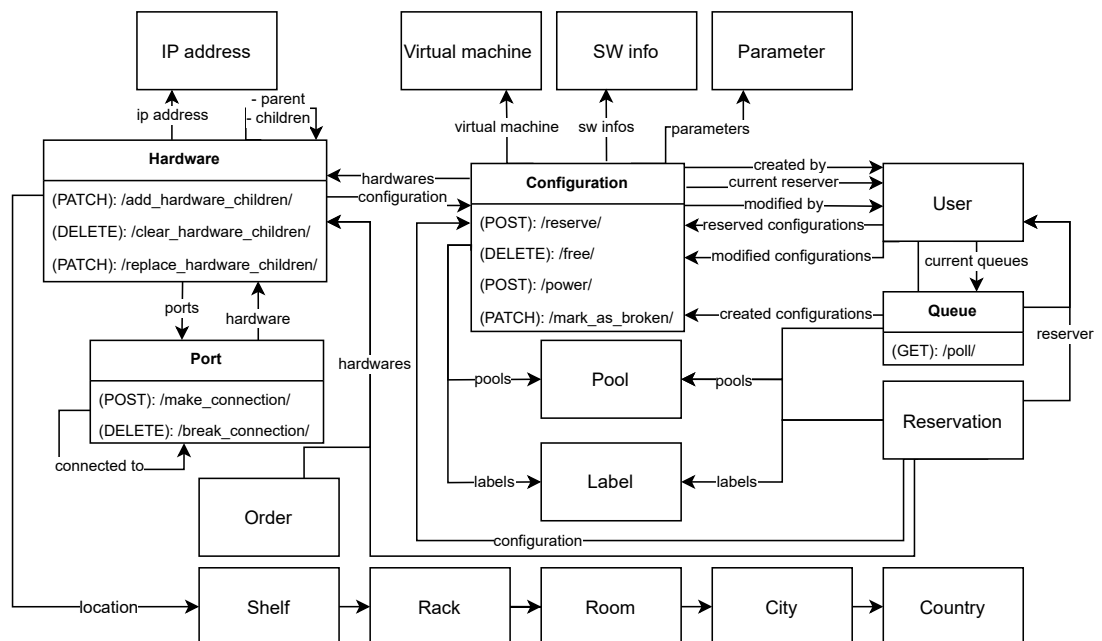


Figure 12. The state diagram of the ReserveTool 2.0 REST API. Each resource that found in the REST API is shown in the diagram.

The state diagram has been simplified in a way that list versions of the resources are omitted because all the resources found in the REST API have it. Figure 13 displays how a relationship between a list and items in it works using the configuration resource as an example. The end-point for the list version of a specific resource displays the objects associated with the resource in a list and handles the POST method which can be used to create new objects of the model the resource represents. The item endpoint is used to handle the methods like PUT, PATCH and DELETE that manipulate a singular object.

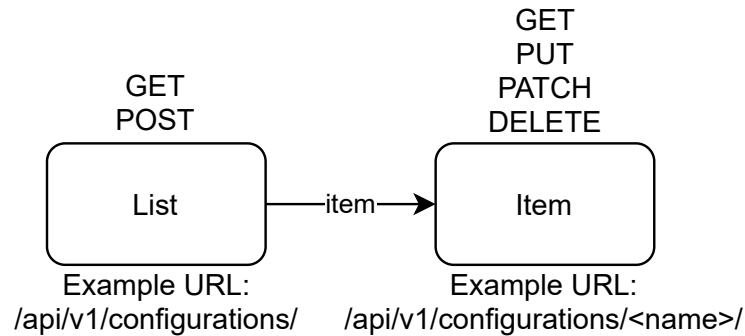


Figure 13. An example using the configuration resource on how the list-item relationship works in ReserveTool REST API. List endpoint shows a list of configurations and handles the POST method for creating new configurations. Item endpoint handles methods that are used to manipulate a single configuration.

4.5. Web UI of ReserveTool 2.0

This section describes the web UI of ReserveTool 2.0. ReserveTool 2.0 web UI consists of views which handle the communication between the database and the front-end. As ReserveTool 2.0 is greatly database driven application, most of the views in the web UI make use of Django’s built-in generic class-based views which provide a lot of functionality for the common use cases such as displaying, creating, editing, and deleting database objects. There five different generic class-based views that are used for working with database objects in ReserveTool 2.0:

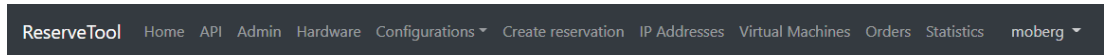
- **ListView** Shows information about multiple objects.
- **DetailView** Shows information about a single object.
- **CreateView** Provides and validates a non-populated form for the creation of a single object.
- **UpdateView** Provides a form populated with existing data and validates the form for editing the single object.
- **DeleteView** Deletes a single object.

Names for the views in ReserveTool 2.0 are composed from the model’s name and action the view handles, e.g. a hardware-detail view shows information related to a specific hardware and a hardware-delete view provides functionality to delete a specific hardware.

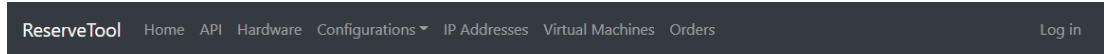
4.5.1. Navigation

Structure of the ReserveTool 2.0 web UI can be thought as consisting of three layers. The first layer consists of views that can be navigated to using a navigation bar which is provided at the top of each web page in ReserveTool 2.0, see Figure 14. Options in the

navigation bar change based on if the user is authenticated and from their permissions: e.g., link to the Admin panel is only shown for admin users.



(a) Navigation bar (user logged in)



(b) Navigation bar (user logged out)

Figure 14. Screenshots of the navigation bar of ReserveTool 2.0. Each option in the navigation bar provides a link to a different view.

The second layer consists of views that can be navigated to from the views of the first layer. The third layer consists of views that are navigated to from the views that belong to the second layer. Although there are a few exceptions with reservation related views where they are provided with a shortcut from home and configurations views. A chart detailing how the web UI of ReserveTool 2.0 can be navigated is presented in Figure 15. At the top row of the chart are the views which are linked to directly from the navigation bar of ReserveTool 2.0. Navigation consists of three different layers which results in that everything in the web UI can be reached within three actions. API and Admin links in the navigation, link to the index ReserveTool 2.0 REST API and to Django's built in admin panel. As they are outside the implementation of ReserveTool 2.0 web UI, they are not discussed in this section.

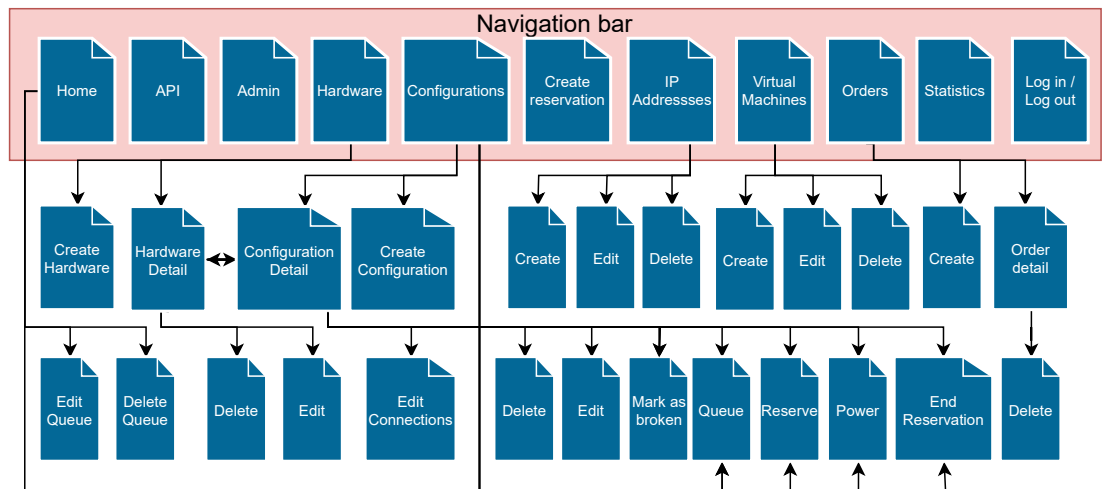


Figure 15. The navigation chart of the web UI of ReserveTool 2.0. Items represented in the chart are views that can be found in ReserveTool 2.0. Arrows between the views tell how a user can navigate through the web UI of ReserveTool 2.0.

4.5.2. Home

The homepage of ReserveTool 2.0 shows the user their currently reserved configurations, their open reservation queues, and their previously reserved

configurations. Screenshot of the homepage is presented in Figure 16. Purpose of the homepage is to provide user with a quick way to manage all their reservation related activities in the ReserveTool and to show contact information or other essential information to the users of the ReserveTool. The homepage has three different tables for authenticated users: Current reservations, queued reservations, and previously reserved configurations. Current reservations table shows the configurations that are currently reserved by the user. Queued reservations show the active queues the user has and provides functionality to edit or delete them. Previously reserved configurations table lists configurations from the past 100 reservations the user has made.

The screenshot shows the ReserveTool homepage with a navigation bar at the top containing links like Home, API, Admin, Hardware, Configurations, Create reservation, IP Addresses, Virtual Machines, Orders, and Statistics. The main content area is divided into three sections:

- Current reservations:** A table with columns Name, Actions, Status, IP addresses, Virtual machine, Labels, and Pools. It lists two reservations: 'test3' and 'test22', both with a status of 'reserved'.
- Queued reservations:** A table with columns Configuration, Actions, Duration, and Created at. It lists two configurations: 'test3' and 'test22', both with a duration of 2:00:00 and created on March 30, 2021, at 6:59 a.m.
- Previously reserved configurations:** A table with columns Name, Actions, Status, IP addresses, Virtual machine, Labels, and Pools. It lists four configurations: 'test3' (reserved), 'power_switchin-testaus-conf' (available), 'testi2222' (available), and 'example_configuration' (available).

On the right side of the page, there is a section for 'Contact information:' and 'More information:'.

Figure 16. A screenshot of the homepage of ReserveTool 2.0. In the homepage, a user can find their currently reserved and previously reserved configurations. The homepage also provides a way to manage queued reservations.

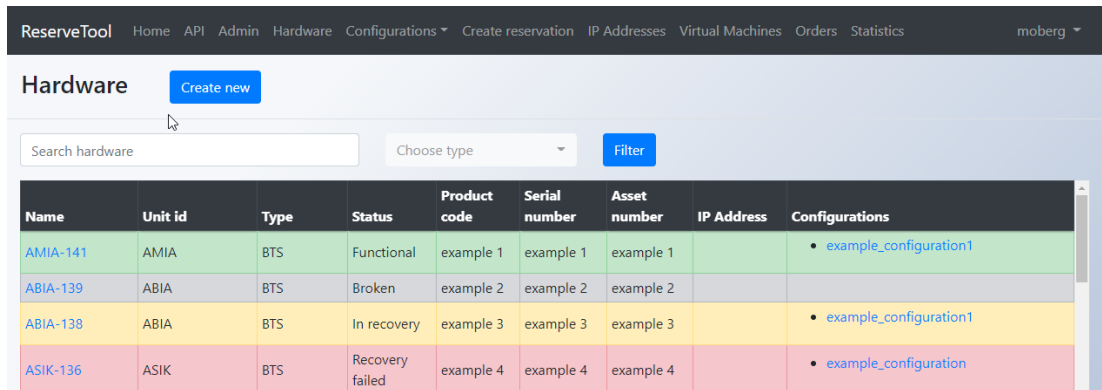
4.5.3. Hardware

This section describes the views related to Hardware model: hardware-list, hardware-detail, hardware-create, hardware-update, hardware-delete views which are used for listing hardware, showing details about individual hardware, creating new hardware, editing, and deleting existing hardware.

Listing hardware

Listing hardware is done with a hardware-list view which displays hardware instances on a single table, each row providing the most essential information about the hardware and a link to the detail view of the individual hardware. Figure 17 shows a screenshot of the hardware-list view. The name of the hardware provides a link to a detail view of the individual hardware. Rows on the table are coloured based on the status of the hardware. Specific hardware can be searched by using a search bar that is provided

above the table. The selector provided at the top of the table can be used to filter the table based on the type of the hardware. Button that links to a view for creating new hardware is provided for admin users above the table.

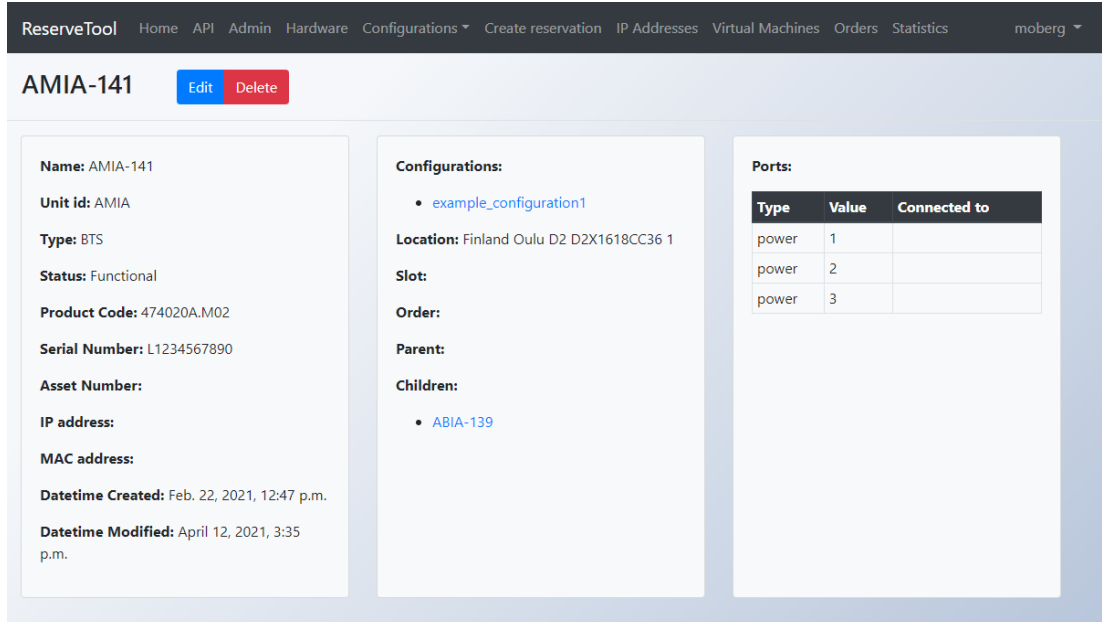


Name	Unit id	Type	Status	Product code	Serial number	Asset number	IP Address	Configurations
AMIA-141	AMIA	BTS	Functional	example 1	example 1	example 1		• example_configuration1
ABIA-139	ABIA	BTS	Broken	example 2	example 2	example 2		
ABIA-138	ABIA	BTS	In recovery	example 3	example 3	example 3		• example_configuration1
ASIK-136	ASIK	BTS	Recovery failed	example 4	example 4	example 4		• example_configuration

Figure 17. A screenshot of the hardware-list view. Each individual hardware is listed on the table with its most essential information.

Showing details of a single hardware

Hardware-detail view is used for displaying all the information that is saved about one hardware instance and to provide controls for editing and deleting the hardware instance. Figure 18 shows a screenshot of the hardware detail view.



Ports:		
Type	Value	Connected to
power	1	
power	2	
power	3	

Figure 18. A screenshot of the hardware-detail view.

Creating, editing, and deleting hardware

Creation of hardware is done with the hardware-create view which uses a form called hardware form that has input fields for all the different fields and relationships that are used by the Hardware model. Editing a hardware instance is done through a hardware update view which uses hardware form with a few non-editable fields such as unit id and type omitted from the form. Creating and editing of ports that are related objects used for mapping connections between different hardware is done with a dynamic formset which is contained within the hardware form. This saves time, as the ports that belong to the hardware can be created and edited within the same view. Creating, editing, and deleting hardware requires user to be admin user.

4.5.4. Configurations

Configurations represent the test environments which are managed by admin users and reserved by regular users of ReserveTool 2.0. Views related to the configurations are configuration-list, configuration-detail, configuration-create, configuration-update, and configuration-delete views. They handle listing configurations, showing details about a single configuration, and creating, editing, and deletion of configurations.

Listing configurations

The listing of configurations is done with the configuration-list view. Configurations are displayed on a single table which can be filtered based on a search term provided by the user or by choosing pools and labels with the two selectors located next to the search bar. Figure 19 shows a screenshot of the configuration-list view.

Name	Actions	Status	IP addresses	Virtual machine	Labels	Pools
example_configuration1	End Reservation Power on	reserved			ABIO	HWAPICI ASD
example_configuration	Reserve	available			FOO AMIA ASIB	HWAPICI
testi2222	Queue	reserved		reservetool.com	FOO	HWAPICI
testingparams		broken			FOO	HWAPICI
power_switchin-testaus-conf	Queue	reserved	• ASIL-114 192.168.1.2	reservetool.com	ABIO	HWAPICI

Figure 19. A screenshot of the configuration-list view. Configurations are listed on the table which can be searched using a search field or filtered by selecting labels and pools. Rows of the table are coloured based on the status of the configuration.

Actions column on the table provides four different actions based on status of the configuration object level permissions which are derived from the user who is reserving the configuration:

- **Reserve:** Shown when the configuration is available, and the user is authenticated.

- **End Reservation:** Shown only to the user reserving the configuration.
- **Queue:** Shown when the configuration is reserved, and the user is not the one reserving the configuration.
- **Power on:** Shown only to the user reserving the configuration when the configuration is powered off.
- **Power off:** Shown only to the user reserving the configuration when the configuration is powered on.

Showing details of a single configuration

Showing information about a single configuration instance is handled by the configuration-detail view. Figure 20 shows a screenshot of the configuration detail view.

The screenshot shows the configuration detail view for 'test_configuration'. The top navigation bar includes 'ReserveTool' and various menu items. Below the navigation bar, there are several action buttons: 'Edit', 'Delete', 'End Reservation', 'Queue', 'Power on', and 'Mark as broken'. The main content area is divided into three sections:

- General Information:**
 - Name: test_configuration
 - Status: reserved
 - Virtual Machine: reservetool.com
 - Pools: HWAPIC1
 - Labels: ABIO
 - Hardware in the configuration:
 - ABIA-138
 - AMIA-141
 - ABIA-139
 - Current reserver:
 - Created by: moberg
 - Datetime created: March 8, 2021, 10:18 a.m.
 - Modified by:
 - Datetime modified: April 13, 2021, 11:35 a.m.
- Ports:**

Type	Value	Hardware	Connected to
power	1	ABIA-138	Nothing selected
generic	2	ABIA-139	Nothing selected
generic	66	ABIA-139	Nothing selected
hdmi	66	ABIA-139	ASIL-114-hdmi-1
- Parameters:**

Name	Value	Description
param1	123	test parameter for picture
param2	321	test parameter for the picture
- Software information:**

Software	Version	Updated
----------	---------	---------

Figure 20. A screenshot of the configuration-detail view. Configuration-detail view shows all the information related to a single configuration. Buttons for editing, deleting, reservation, powering and marking the configuration as broken are provided at the top.

Configuration-detail view provides buttons for three additional actions for the configuration that are not shown in the configuration-list view:

- **Edit:** Links to the configuration-update view.

- **Delete:** Links to the configuration-delete view.
- **Mark as broken:** Changes the status of the configuration to broken, ends the current reservation and cancels queued reservations if there are any. This action can only be used by the user currently reserving the configuration or by admin users.

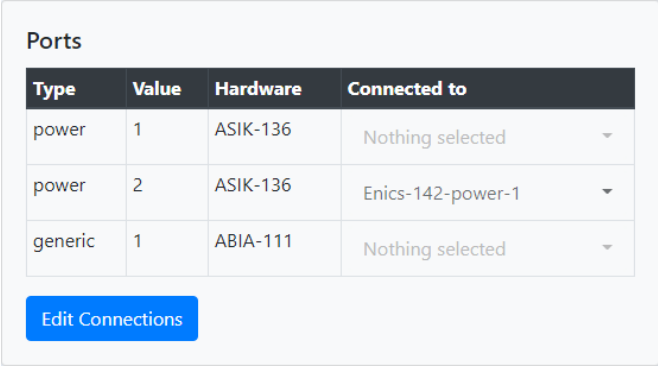
Configuration-detail view also contains a connection making view. For more information about the connection making view, see subsection 4.5.5 below.

Creating, editing, and deleting configurations

Creating configurations is done through configuration-create view which uses a form that has input fields for the fields of the Configuration model and selectors for the relationships that the Configuration model uses. Configuration form also has a dynamic formset for the creation of parameter objects related to that specific configuration. Configuration-update view uses the same form as the configuration-create view with the exception that regular users are allowed only to edit the parameter formset.

4.5.5. Connection Making View

Connections between different hardware can be made using the connection making view which is an asynchronous⁹ view found in the configuration-detail view. Screenshot of the connection making view is presented in Figure 21. Editing connections requires user to be admin user.



The screenshot shows a web interface titled "Ports" containing a table with four columns: Type, Value, Hardware, and Connected to. Below the table is a blue button labeled "Edit Connections".

Type	Value	Hardware	Connected to
power	1	ASIK-136	Nothing selected
power	2	ASIK-136	Enics-142-power-1
generic	1	ABIA-111	Nothing selected

Figure 21. A screenshot of the connection making view. Ports found in the configuration are listed on the table. For each port listed on the table, there is a dropdown selector which is used to select the port to which it is connected to.

Currently, only the mapping of power connections provides functional purpose in ReserveTool 2.0. In the Figure 21 which shows ports of one configuration, we can see that *power-2* port of the ASIK-136 hardware is connected to a *power-1* port

⁹Asynchronous views can send and retrieve information from the web server without having to refresh the whole web page.

of Enics-142 which is the power switch used in the configuration. To power on the configuration, each power port of the power switch of the configuration that is connected to the other hardware in the configuration has to be turned on. In the case shown in Figure 21, to power on the configuration, the power-1 port on the Enics-142 power switch is switched on.

4.5.6. IP Address and Virtual Machine Views

There are two views in ReserveTool 2.0 web UI for managing IP Address and Virtual Machine objects. The implementation for both views is very similar with only a few differences: database models used are different and IP addresses are represented on two tables instead of one based on the unique property on the IP address object which determines if it can be used by multiple hardware. Figure 22a shows a screenshot of the IP address view and Figure 22b shows a screenshot of the Virtual machine view. Both views provide controls for editing, deleting, and creating new objects.

Unique IP addresses			Common IP addresses		
IP Address	Actions	Hardware	IP Address	Actions	Hardware
1.2.3.4	Edit Delete		1.1.1.2	Edit Delete	
10.34.143.104	Edit Delete	• Enics-123	0.0.0.0	Edit Delete	
192.168.1.2	Edit Delete	• ASIL-114			

(a) IP address view

Domain	Actions	Configurations
google.com	Edit Delete	
testidoma.in	Edit Delete	
reservetool.com	Edit Delete	<ul style="list-style-type: none"> • ABIO-92 • power_switchin-testaus-conf • testi2222

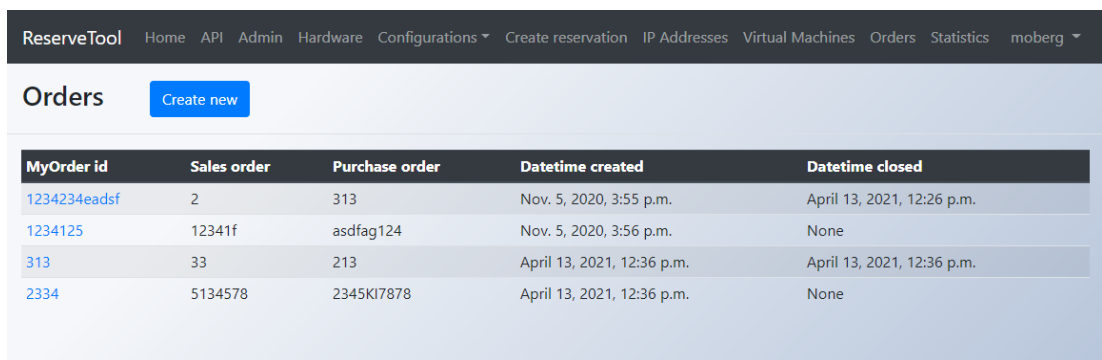
(b) Virtual machine view

Figure 22. Screenshots of IP address and virtual machine views. Both views are similar with edit and delete buttons provided for each represented object. The button for creating new objects is provided at the top in both views.

4.5.7. Orders

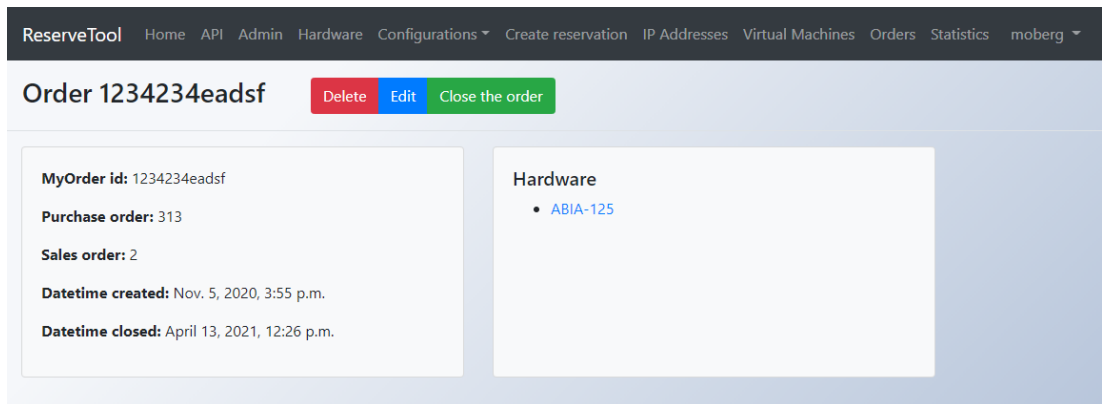
Instances of Order model are used to store purchasing information about hardware. Order views are meant to be used by the admin users of ReserveTool to keep track of new hardware shipments. Order is pending or open when it does not have a closing time in the database. Once the hardware in the order is received by ReserveTool admins, order can be closed. There are five different views related to order objects: order-list, order-detail, order-create, order-edit, and order-delete.

Order-list view is used to list orders in a table with most essential fields of the orders presented. Screenshot of the order-list view can be seen in Figure 23a. Each order in the table has a link to the order-detail view of the order which shows all the details about a single order and provides the controls for the manipulation of the order instance. Screenshot of the order-detail view can be seen in Figure 23b.



MyOrder id	Sales order	Purchase order	Datetime created	Datetime closed
1234234eadsf	2	313	Nov. 5, 2020, 3:55 p.m.	April 13, 2021, 12:26 p.m.
1234125	12341f	asdfag124	Nov. 5, 2020, 3:56 p.m.	None
313	33	213	April 13, 2021, 12:36 p.m.	April 13, 2021, 12:36 p.m.
2334	5134578	2345K17878	April 13, 2021, 12:36 p.m.	None

(a) Order-list view



MyOrder id	Sales order	Purchase order	Datetime created	Datetime closed
1234234eadsf	2	313	Nov. 5, 2020, 3:55 p.m.	April 13, 2021, 12:26 p.m.

Hardware

- ABIA-125

(b) Order-detail view

Figure 23. Screenshots displaying order-list and order-detail views. Link to order-create view which is used for creating new orders is provided in the order-list view at the top. Controls for editing, deleting, and closing the order are provided in the order-detail view.

4.5.8. Statistics

Purpose of the statistics view is to provide the admin users of ReserveTool 2.0 statistics about the usage of ReserveTool 2.0. There are two different statistics that are

computed from the reservation objects attached to the hardware and configurations: the reservation rate and power on rate.

Reservation rate ranges between zero and one, one being that object in question was reserved the whole available time and zero being that the object was not reserved at all during the whole available time. Equation (1) shows how the reservation rate is calculated.

$$\text{reservation rate} = \frac{\text{reserved time}}{\text{total time available}} \quad (1)$$

The power on rate is calculated somewhat similar to the reservation rate and ranges from zero to one. The difference is that it measures the ratio of time which a configuration or hardware was powered on. This provides more accurate information about the true usage of the hardware because a configuration is not necessarily used all the time during a reservation. Equation (2) shows how the power on rate is calculated.

$$\text{power on rate} = \frac{\text{power on time}}{\text{total time available}} \quad (2)$$

Both statistics are represented in a bar chart. The time scale of the chart can be adjusted to display data from the last day, week or month. User can also choose which configurations or hardware the data will be fetched from. Screenshot of the statistics page is presented in Figure 24. Different selectors at the top of the page numbered from

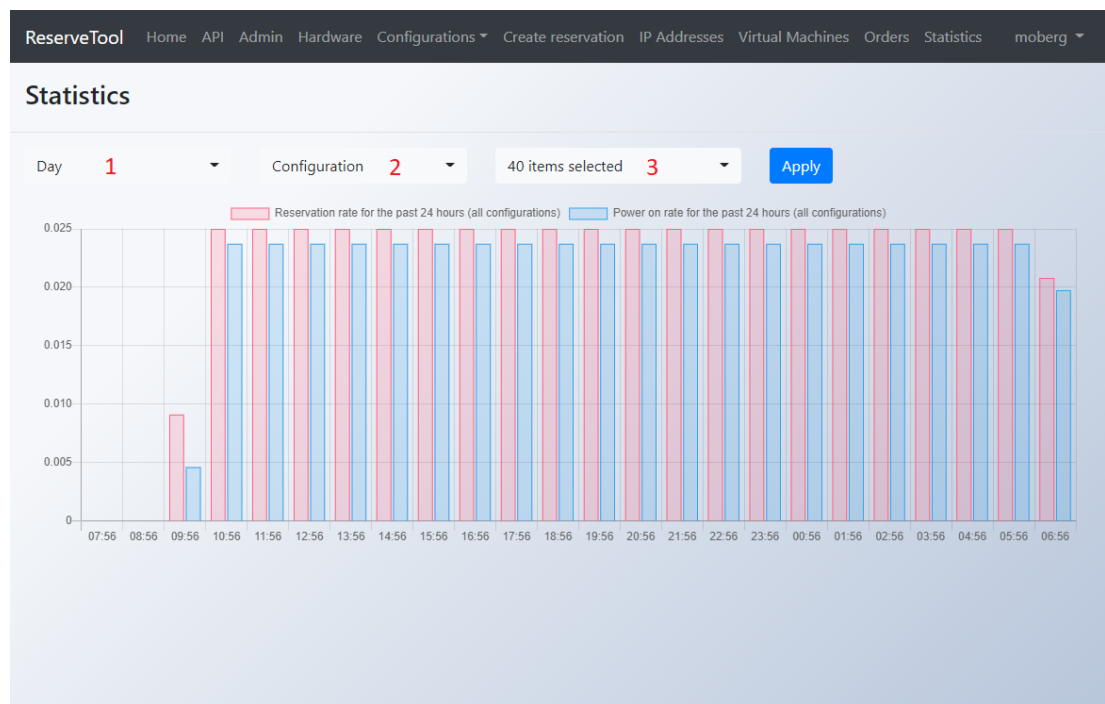


Figure 24. A screenshot of the statistic page. The reservation rate (red) and power on rate (blue) are both displayed in the same bar chart where they can be compared.

one to three, determine the data that is displayed on the chart. The first selector (marked with 1) lets the user select time frame for the chart. The options for the time frame are day, week, or month. Second selector (marked with 2) is used to select the type for

the third (marked with 3) selector. There are three different types: a configuration, hardware, and pool. The third selector lists either configurations, hardware, or pools depending on the selection of the second selector and determines the hardware or configurations from which the data is queried from.

4.6. Configuration Reservation

ReserveTool 2.0 offers several different ways to reserve a configuration. Reservation can be made through the web UI or through the REST API. Making a reservation requires user provide a few key pieces of information about the reservation:

- **Duration:** A user specified duration for the reservation. Must be under the maximum duration which is set globally for all the reservations by the admins.
- **Configuration or pools and labels:** User must provide either configuration that should be reserved or pools and labels which the desired configuration should match.
- **Power off automatically:** Setting which determines whether the configuration should be powered off after the reservation.
- **Free automatically:** Setting which determines whether the reservation should be ended automatically once the duration runs out. This setting is enabled by default and it can only admin users can disable it.

4.6.1. Reservation System

At the core of ReserveTool 2.0 reservation system are Reservation and Queue objects. Reservation objects hold all the information about reservations such as the length of the reservation, who made the reservation, and the configuration which is the target for the reservation. A Queue object is created when a user queues a reservation to a configuration which is already reserved. When the current reservation ends, if there is a queue object in the reservation queue, a new reservation object is created using the information contained in the queue object. Reservation queues in the ReserveTool use first in, first out -principle which means that the oldest queue object that is not closed, is processed first. Flowchart describing the steps of creating and ending a reservation is presented in Figure 25.

Automatic ending of reservations and powering off configurations is handled by two different management commands *check_reservations* and *check_powers* that are run periodically every few minutes by their respected systemd services in the ReserveTool server. Flow charts describing how the management scripts work are presented in Figure 26.

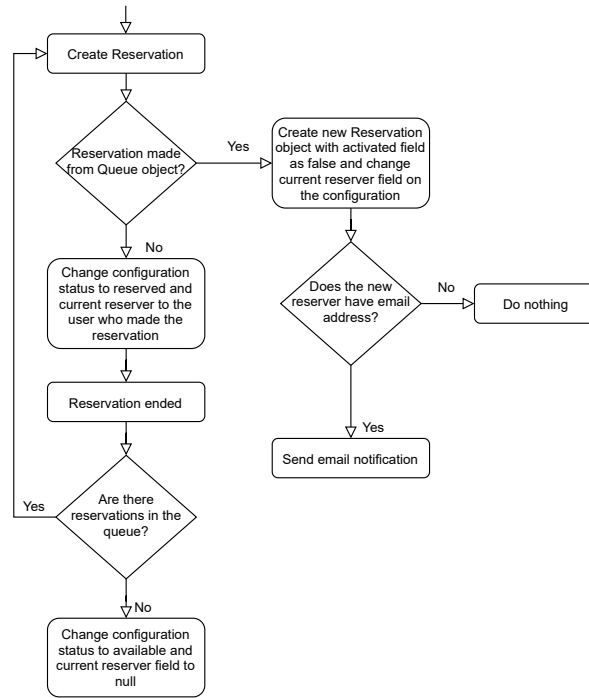
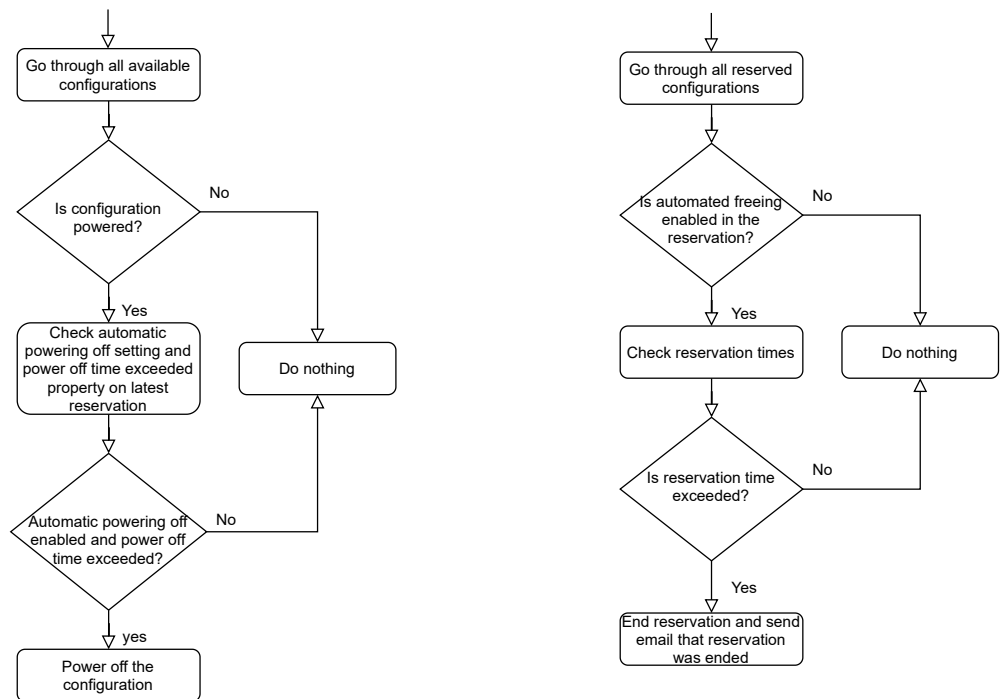


Figure 25. Flowchart displaying what steps are taken by ReserveTool 2.0 when a reservation is created and ended.



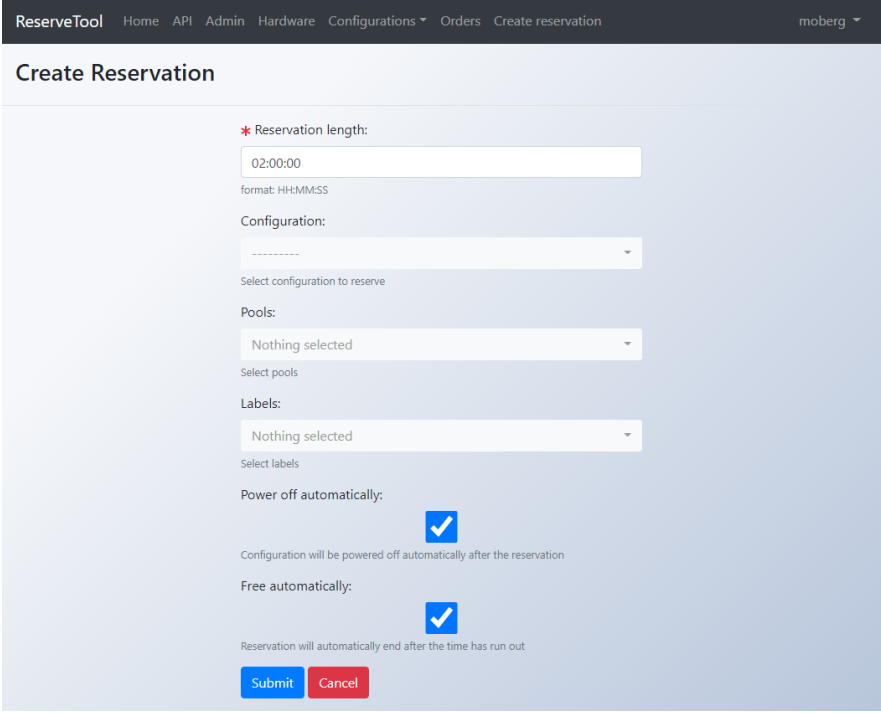
(a) check_reservations script

(b) check_powers script

Figure 26. Flowchart displaying the decision process of the two management scripts: check_reservations and check_powers.

4.6.2. Reservation through the Web UI

There are two ways that a user can use to reserve a configuration. Explicitly finding a configuration they want to reserve or by choosing pools and labels as criteria when making a reservation and letting ReserveTool 2.0 choose a configuration that matches that criteria. A configuration can be reserved from the configuration-list view, configuration-detail view, or from a reservation-create view. The support for reservation using pools and labels is provided only in the reservation-create view. Figure 27 shows a screenshot of the reservation-create view. After submitting the form displayed in the view, the user will be redirected to the detail view of the configuration that was reserved for them.



The screenshot shows the 'Create Reservation' form in the ReserveTool web interface. The form is titled 'Create Reservation' and is located under the 'Orders' menu. The form contains the following fields and options:

- Reservation length:** A text input field containing '02:00:00'. Below it, the format 'format: HH:MM:SS' is indicated.
- Configuration:** A dropdown menu with a placeholder '-----'. Below it, the text 'Select configuration to reserve' is displayed.
- Pools:** A dropdown menu with the text 'Nothing selected'. Below it, the text 'Select pools' is displayed.
- Labels:** A dropdown menu with the text 'Nothing selected'. Below it, the text 'Select labels' is displayed.
- Power off automatically:** A checkbox that is checked, with a blue checkmark icon. Below it, the text 'Configuration will be powered off automatically after the reservation' is displayed.
- Free automatically:** A checkbox that is checked, with a blue checkmark icon. Below it, the text 'Reservation will automatically end after the time has run out' is displayed.

At the bottom of the form, there are two buttons: 'Submit' (blue) and 'Cancel' (red).

Figure 27. A screenshot of the reservation-create view. In the form, the user has to choose either a configuration or pools and labels which the configuration should match.

Reservations done from the configuration-list view or configuration-detail view, are explicit reservations aimed for a specific configuration and use a modal window which asks for the reservation length and settings for power off automatically and free automatically. Free automatically setting can only be changed by admin users. Screenshot about the reservation modal window is presented in Figure 28.

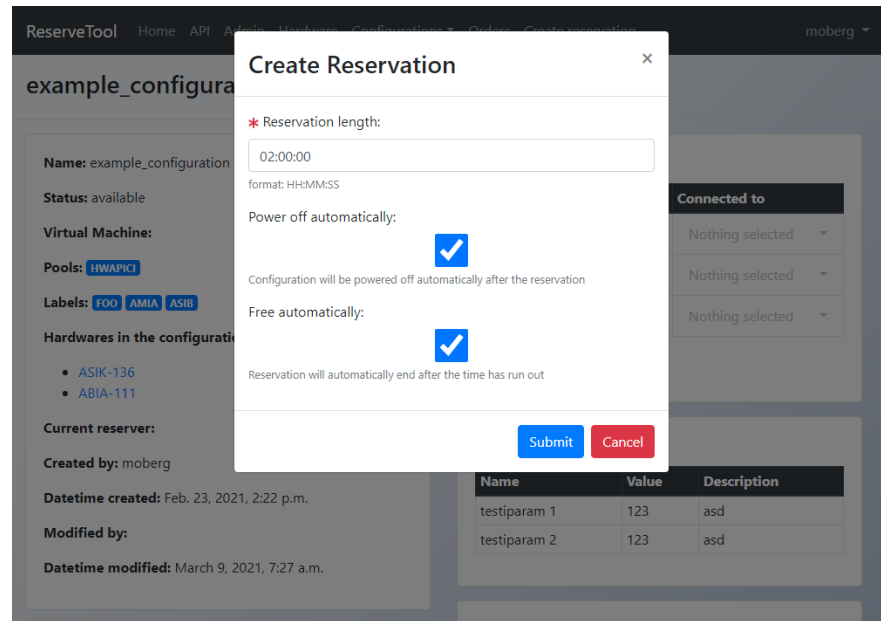


Figure 28. A screenshot showing a modal window when a reservation is made using the configuration-detail view.

4.6.3. Reservation through the REST API

There are three different ways a reservation can be made in the ReserveTool REST API:

1. Creating a new Queue object by doing HTTP POST to `/reservetool/queues/` endpoint. Creating a queue object requires user to provide length for the reservation and either link to the configuration or pools and labels that match some configuration. Figure 29 shows a flowchart detailing how ReserveTool 2.0 handles reservations using the Queue endpoint. This method is the preferred way for the CI servers to reserve configurations due to its flexibility, as it returns a Queue or Reservation object depending on the status of the configuration.
2. Creating a new reservation object by doing HTTP POST to `/reservetool/reservations/` endpoint. This method requires similar data as the queue endpoint. Only difference is that for the request to succeed, the configuration must be available. This method can be used when the user wants to utilize reservation with pool and labels but does not want to queue reservation if there are none currently available.
3. HTTP POST request to `/reservetool/configurations/<name>/reserve/` endpoint which is unique to a specific configuration. This method is the simplest way of reserving a configuration through the REST API. The downside of this method is that the configuration has to be available for the request to succeed, as this method is limited to a single configuration and configuration selection using pools and labels cannot be utilized.

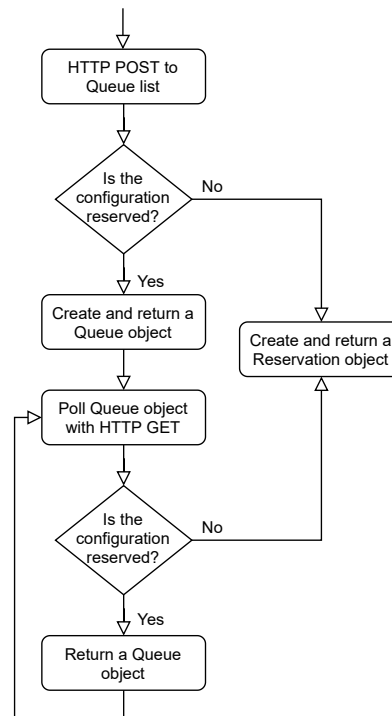


Figure 29. Flowchart showing how queue endpoint is utilized when making a reservation through the REST API. The client can determine when it is their turn in reservation when they are returned a Reservation object.

4.7. Importing the Hardware Inventory

Importing the hardware from ReserveTool 1.0 is necessary before ReserveTool 2.0 can be taken into use. Because all the information that ReserveTool 2.0 requires is not available from ReserveTool 1.0 it was decided within the PSCI team that all the hardware in the PSCI laboratory should be catalogued in a spreadsheet which can be used to import all the necessary data to ReserveTool 2.0.

Importing from the spreadsheet was done by converting the spreadsheet to a comma-separated values (CSV) format and reading it with a Python script and creating necessary objects based on the information provided in the spreadsheet.

4.8. Tests to Ensure Operability of ReserveTool 2.0

Testing in ReserveTool 2.0 is currently done only for the REST API. Testing is implemented using Django's test framework that is extended by DRF. Purpose of the tests is to provide a set of automated tests that can be run always when there is a change to either database models or to the views of the REST API. The tests consist of creation, modification, and deletion of database objects, and reservation related actions such as reserving and queuing configurations.

5. EVALUATION AND DISCUSSION

The goal of this thesis was to improve ReserveTool 1.0 by increasing its usability and reliability, improving the hardware utilization rate and provide support for new features such as dynamic hardware configurations, hardware inventory, and reservation using pools and labels. The product created during the thesis work was a new application called ReserveTool 2.0 which was designed to achieve the goals mentioned above. Most of the set goals such as the improved hardware inventory and the support for dynamic configurations were achieved by ReserveTool 2.0. The goal of improving reliability and hardware utilization could not be evaluated, because there was not enough time to take ReserveTool 2.0 to production and monitor its performance. Although, it can be argued that the poor hardware inventory was one of the causes for low hardware utilization in ReserveTool 1.0, as it was not as apparent what hardware the configurations in ReserveTool 1.0 represented. Having a better hardware inventory in ReserveTool 2.0 should lead to a smaller number of obsolete configurations in the application.

Achievement of functional goals such as support for dynamic hardware configurations, hardware inventory, and reservation using pools and labels can be validated by examining the implemented application. The support for dynamic hardware configurations and hardware inventory was achieved by designing the database in a way where configurations are built from hardware objects that have port objects attached which can be used to map the connections between different hardware. Reservation using pools and labels was implemented for both the REST API (subsection 4.6.3) and web UI (subsection 4.6.2) sides of ReserveTool 2.0. Usability of ReserveTool 2.0 was validated with a user experiment which is detailed in the next section.

5.1. User Experiment

User feedback for ReserveTool 2.0 was gathered in a form of user experiment and a survey. The participants for the user experiment consisted of members of the PSCI team with most of them having previous experience from using ReserveTool 1.0. In the task used in the experiment, participants had to create a configuration and test it by reserving and powering it. The environment for the experiment was a development server running the latest version of ReserveTool 2.0 application. The experiment consisted of eight different steps:

1. Creating IP address object for the power switch.
2. Creating the pool and label objects for the configuration in the admin panel.
3. Creating two hardware: a power switch and a BTS type hardware.
4. Creating a configuration using the created hardware, pool, and label.
5. Making a power connection to the power switch in the configuration-detail view.

6. Reserving the created configuration in the detail page of the configuration and powering it on.
7. Powering off the configuration and ending the reservation.
8. Answering the survey.

5.1.1. Survey

The survey consisted of various of different qualitative questions. Questions were inspired by the Jakob Nielsen's ten usability heuristics [40].

1. *Was the system feedback in ReserveTool clear while performing the task? If not, where was it unclear?* Purpose of this question is to evaluate the feedback ReserveTool 2.0 gives to the user when they perform actions in it.
2. *Please describe any problem(s) you had using the ReserveTool during the task.* The aim of this question is to find any usability problems that were found during the experiment which can then be added to the list of future improvements.
3. *Were there any instances where using the ReserveTool was unintuitive? (The ReserveTool behaved differently than you expected)* This question relates to Nielsen's fourth usability heuristic which is consistency and standards. Perspective from different users is valuable feedback because different people have different expectations.
4. *Did you get any error notifications when performing the task?* This question is simple yes and no question which determines whether the user should be asked the next two questions.
5. *What were the errors?* This question is used to determine in which part of the experiment errors happened and what were they.
6. *Was it stated clearly how the errors you had can be avoided? If not, how would you improve it?* It is important that there are no unhandled error cases in ReserveTool 2.0. Error messages are meant to help the users, so they have to be able to understand them which makes it valuable to get their opinion.
7. *How did the navigation in the new version of ReserveTool feel compared to the old one?* This question is aimed towards participants who have experience using ReserveTool 1.0. It helps the transition from ReserveTool 1.0 to 2.0 if the navigation feels familiar. This question is also somewhat linked to the question about unintuitive behaviour as the users of ReserveTool 1.0 have expectations on how the ReserveTool is navigated.
8. *Was there anything in the old version of ReserveTool that you would have liked to see in the new version?* This question is also aimed at participants who have used ReserveTool 1.0. The aim of this question is to find the features of ReserveTool 1.0 that should be ported in some way to ReserveTool 2.0 in the future.

9. *How would you describe your overall experience when compared to previous version of the ReserveTool?* Purpose of this question is to survey how well ReserveTool 2.0 is received by the users of ReserveTool 1.0.
10. *Additional feedback* If the participant has any feedback about ReserveTool 2.0 which does not respond to any other question in the survey, they can use the field provided under this question for that.

5.1.2. Survey Results

The experiment was done by four different participants. From the responses in the survey, it could be determined that three of the four participants had experience using ReserveTool 1.0. With a limited number of responses in the survey, extensive thematic analysis for the responses was not carried out. Instead, three different key topics were identified based on the content of the responses. They are discussed in the following subsections under this section.

Problems and errors encountered during the experiment

There were two participants who encountered errors during the experiment. The first error was related the authentication. The user said that he could not log in to the ReserveTool during the first five minutes of the experiment using the credentials provided in the instructions for the experiment. After they managed to log in, they did not encounter the error again and could not reproduce it. The error might be related only to the user account used for the experiment, and as it could not be reproduced, it might not be a serious one.

The second error that was encountered was related to data validation. The participant created a port with a value "1234" which was not in the specified range for the power switch type that was used. This error was communicated with a message that stated that "*TestPowerSwitch has maximum of 1 outlets. Given value is 1234*". This error can be avoided in the future by adding more validation in the form that creates the port.

Direct feedback and comparison to ReserveTool 1.0.

Overall, participants were positive about their experience using ReserveTool 2.0 during the experiment. The survey question 3. about the unintuitive usage of ReserveTool 2.0 yielded two different viewpoints:

1. Two participants reported that usage of Django's admin panel for the creation pool and label objects felt unintuitive and would have rather used UI similar to the creation of other objects. This is understandable because Django's admin panel differs greatly from the rest of the web UI of ReserveTool 2.0. The design choice of using Django's admin panel for pools and labels was driven by the fact that creation of new pool or label is a rare task performed by admins that the admin panel already has the functionality built-in which reduces the development time.

2. One participant reported that usage felt intuitive but very different, because the approach of *"building a configuration from pieces"* was different than they have used to when working with ReserveTool 1.0. This is to be expected as the creation of configurations is very different between ReserveTool 1.0 and 2.0. In ReserveTool 1.0, user creates a configuration which does not require any hardware information, because the hardware information about a configuration is fetched with a script directly from the hardware residing in the IP address allocated to the configuration. This has a drawback that the information can be obsolete, because the hardware information can only be fetched while the configuration is powered on. In ReserveTool 2.0 user has much better visibility on what hardware the configuration they are creating has, as the hardware must be created prior to the creating of the configuration.

When comparing the navigation of ReserveTool 2.0 and 1.0. The participant reported that the navigation felt smoother in ReserveTool 2.0 than in ReserveTool 1.0. There is some threat to the validity of this result, because the participants have experience from using the live version of ReserveTool 1.0 which has a lot more data in its database which then increases the loading times when navigating the ReserveTool due to longer query times.

When comparing their overall experience between ReserveTool 2.0 and 1.0, participants reported that ReserveTool 2.0 is more complicated and requires more steps from the user while providing more information. One participant also mentioned that UI of ReserveTool 2.0 is visually more pleasing.

Suggestions for future development of ReserveTool 2.0

There were a few suggestions about features that could be developed in the future. Some of them were features that are found in ReserveTool 1.0 such as a description field for the configurations or reservation history of configurations. The latter is already implemented in ReserveTool 2.0 but was not explored during the experiment which makes it understandable that the participant missed it. The most interesting suggestion that would greatly speed up the creation of a configuration was to have the creation of all the simple related models Pool, Label, IPAddress, and VirtualMachine in a single form.

5.2. Future Work

The future work for the ReserveTool 2.0 includes various of things ranging from important features that are still needed for adopting ReserveTool 2.0 to production, to endless small improvements and expansions in the future. One important feature before ReserveTool 2.0 can be taken to production is an API client that simplifies the usage of ReserveTool 2.0 REST API for the CI servers.

Once the ReserveTool 2.0 is properly taken into production, tests for both the REST API and web UI should be implemented to ensure that the ReserveTool 2.0 works as intended. Data integrity in ReserveTool 2.0 could be increased with a periodically run management command which would check the hardware information from the real

hardware and compare it with the information in the database object of the respected hardware. Any anomalies detected would be reported to the admin users.

The statistics view currently has only one bar chart which provides information about the reservation rate and power on rate. This view could be expanded to house other charts that measure some important metric related to the performance of ReserveTool 2.0.

Groundwork for dynamic hardware configurations was done by allowing to store information about different hardware connections. In the future, using a fiber switch, dynamic configurations could be made on the fly using that information.

As one of the goals of ReserveTool 2.0 was a better hardware inventory, exporting and importing of hardware information should be supported in the future. The file format used for importing and exporting should be a format such as CSV or JSON to make it portable to different systems and tools.

6. SUMMARY

The objective of this thesis was to improve existing test environment management application called ReserveTool 1.0 which operates in a continuous integration environment. ReserveTool 1.0 had several limitations that were found by the users of the application and fixing or eliminating them was the priority of this thesis work. Among those limitations were a low hardware utilization rate, no support for dynamic hardware configurations, usability limitations such as permissions handling, and most importantly, a poor hardware inventory.

To fix the limitations, a new web application called ReserveTool 2.0 was developed. A new database structure for the ReserveTool 2.0 was designed in mind that every piece hardware in the laboratory is registered to ReserveTool 2.0 and test environments called configurations are built from the hardware which leads to a much more accurate hardware inventory. Both web UI and REST API were implemented for ReserveTool 2.0 to provide interfaces for management and reservation of configurations. Proper authentication and authorization methods were implemented for the application which allowed separation between different user groups and their permissions.

The usability of the web UI of ReserveTool 2.0 was evaluated with a user experiment. The feedback from the participants of the experiment was positive with a few future suggestions provided. Comparison between the utilization rate and reliability between ReserveTool 1.0 and 2.0 could not be done as there was not enough time to take ReserveTool 2.0 to production and monitor its performance. The implemented application provides an easily expandable base for the future with several improvements and a support for new features such as dynamic hardware configurations.

7. REFERENCES

- [1] Fowler M. & Foemmel M. (2006), Continuous integration.
- [2] Mårtensson T., Ståhl D. & Bosch J. (2016) Continuous integration applied to software-intensive embedded systems—problems and experiences. In: International Conference on Product-Focused Software Process Improvement, Springer, pp. 448–457.
- [3] Hammant P. (2017). URL: <https://trunkbaseddevelopment.com/>. Accessed 24.11.2020.
- [4] URL: <https://www.gerritcodereview.com/>. Accessed 24.11.2020.
- [5] Myers G.J. & Badgett T. (2004) The art of software testing, vol. 2. Wiley Online Library.
- [6] Krasner H. (2021), The cost of poor quality software in the us: A 2020 report. URL: <https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-2020-report.htm>, accessed 15.01.2021.
- [7] Broekman B. & Notenboom E. (2003) Testing embedded software. Pearson Education.
- [8] Jorgensen P.C. (2018) Software testing: a craftsman’s approach. CRC press.
- [9] Khan M.E., Khan F. et al. (2012) A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl 3.
- [10] Acharya S. & Pandya V. (2012) Bridge between black box and white box—gray box testing technique. International Journal of Electronics and Computer Science Engineering 2, pp. 175–185.
- [11] Naik K. & Tripathy P. (2011) Software testing and quality assurance: theory and practice. John Wiley & Sons.
- [12] Trautsch F., Herbold S. & Grabowski J. (2020) Are unit and integration test definitions still valid for modern java projects? an empirical study on open-source projects. Journal of Systems and Software 159, p. 110421.
- [13] Kassab M., DeFranco J.F. & Laplante P.A. (2017) Software testing: The state of the practice. IEEE Software 34, pp. 46–52.
- [14] Wong W.E., Horgan J.R., London S. & Agrawal H. (1997) A study of effective regression testing in practice. In: PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering, IEEE, pp. 264–274.
- [15] Yoo S. & Harman M. (2012) Regression testing minimization, selection and prioritization: a survey. Software testing, verification and reliability 22, pp. 67–120.

- [16] Hilton M., Nelson N., Tunnell T., Marinov D. & Dig D. (2017) Trade-offs in continuous integration: assurance, security, and flexibility. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 197–207.
- [17] Elbaum S., Rothermel G. & Penix J. (2014) Techniques for improving regression testing in continuous integration development environments. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 235–245.
- [18] Janzen D. & Saiedian H. (2005) Test-driven development concepts, taxonomy, and future direction. *Computer* 38, pp. 43–50.
- [19] Duvall P.M., Matyas S. & Glover A. (2007) *Continuous integration: improving software quality and reducing risk*. Pearson Education.
- [20] Humble J. & Farley D. (2010) *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
- [21] Zhao Y., Serebrenik A., Zhou Y., Filkov V. & Vasilescu B. (2017) The impact of continuous integration on other software development practices: a large-scale empirical study. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp. 60–71.
- [22] Vasilescu B., Yu Y., Wang H., Devanbu P. & Filkov V. (2015) Quality and productivity outcomes relating to continuous integration in github. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 805–816.
- [23] Hilton M., Tunnell T., Huang K., Marinov D. & Dig D. (2016) Usage, costs, and benefits of continuous integration in open-source projects. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp. 426–437.
- [24] Beck K. (2000) *Extreme programming explained: embrace change*. Addison-Wesley Professional.
- [25] Ghaleb T.A., Da Costa D.A. & Zou Y. (2019) An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering* 24, pp. 2102–2139.
- [26] Jin X. & Servant F. (2020) A cost-efficient approach to building in continuous integration. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), IEEE, pp. 13–25.
- [27] Garousi V., Felderer M., Kuhrmann M., Herkiloğlu K. & Eldh S. (2020) Exploring the industry’s challenges in software testing: An empirical study. *Journal of Software: Evolution and Process* , p. e2251.
- [28] Karlesky M., Williams G., Bereza W. & Fletcher M. (2007) Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In: Proc. Emb. Systems Conf, CA, USA, pp. 1518–1532.

- [29] Vöst S. & Wagner S. (2016) Trace-based test selection to support continuous integration in the automotive industry. In: Proceedings of the International Workshop on Continuous Software Evolution and Delivery, pp. 34–40.
- [30] Aarno D. & Engblom J. (2014) Software and system development using virtual platforms: full-system simulation with wind river simics. Morgan Kaufmann.
- [31] Engblom J. (2015) Continuous integration for embedded systems using simulation. In: Embedded World 2015 Congress.
- [32] Continuous delivery. URL: <https://continuousdelivery.com/>. Accessed 03.02.2021.
- [33] Continuous integration vs. continuous delivery vs. continuous deployment. Accessed 03.02.2021.
- [34] URL: <https://velocitywms.com/about-us/>. Accessed 19.10.2020.
- [35] Django Software Foundation, Django. URL: <https://djangoproject.com>.
- [36] Makai M. (2012), Object-relational mappers (orms). URL: <https://www.fullstackpython.com/object-relational-mappers-orms.html>. Accessed 30.12.2020.
- [37] Wahl M., Howes T. & Kille S. (1997), Rfc2251: Lightweight directory access protocol (v3).
- [38] Masse M. (2011) REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. " O'Reilly Media, Inc."
- [39] Fielding R.T. (2000) Architectural styles and the design of network-based software architectures, vol. 7. University of California, Irvine Irvine.
- [40] Nielsen J. (2005), Ten usability heuristics.

8. APPENDICES

Appendix 1 Description of the fields of ReserveTool 2.0 database models.

This appendix shows the fields of each database model in ReserveTool 2.0 in table format. Name, type, constraints and deleting behaviour are given for each field.

Table 4. The fields used in Configuration model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
status	CharField	required, choices: <ul style="list-style-type: none"> • available • reserved • maintenance • broken 	-
datetime_creation	DateTimeField	-	-
datetime_modified	DateTimeField	-	-
pools	ManyToManyField to Pool model	required	set null
labels	ManyToManyField to Label model	required	set null
virtual_machine	ForeignKey to VirtualMachine model	-	set null
created_by	Foreignkey to User model	-	set null
modified_by	Foreignkey to User model	-	set null
current_reserver	Foreignkey to User model	-	set null

Table 5. The fields used in Hardware model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
unit_id	CharField	required, must be one of the unit_id choices	-
type	CharField	required, choices: <ul style="list-style-type: none"> • power switch • bts • network switch • serial port server • simics • virtual environment • other 	-
status	CharField	required, choices: <ul style="list-style-type: none"> • functional • broken • in recovery • recovery failed 	-
product_code	CharField	-	-
serial_number	CharField	-	-
asset_number	CharField	-	-
mac_address	CharField	-	-
slot	IntegerField	choices: 0-7	-
datetime_creation	DateTimeField	-	-
datetime_modified	DateTimeField	-	-
configuration	ManyToManyField to Configuration model	bts, simics and virtual environment can only be in one configuration	set null
parent	ForeignKey to self	-	set null
ip_address	ForeignKey to IP address model	-	set null
location	ForeignKey to Shelf model	-	set null
order	ForeignKey to Order model	-	set null

Table 6. The fields used in Reservation model

field name	field type	constraints	delete behaviour
reservation_length	DurationField	must be under maximum duration	-
datetime_creation	DateTimeField	-	-
datetime_closed	DateTimeField	-	-
configuration	ForeignKey to Configuration model	-	set null
hardwares	ManyToManyField to Hardware model	-	set null
reserver	ForeignKey to User model	-	set null
pools	ManyToManyField to Pool model	-	set null
labels	ManyToManyField to Pool model	-	set null
power_off_automatically	BooleanField	-	-
free_automatically	BooleanField	-	-
activated	BooleanField	-	-
power_on_time	DurationField	-	-
power_on_time_added	DateTimeField	-	-

Table 7. The fields used in Order model

field name	field type	constraints	delete behaviour
myorder_id	CharField	unique, required	-
purchase_order	CharField	unique, required	-
sales_order	CharField	unique, required	-
datetime_creation	DateTimeField	-	-
datetime_closed	DateTimeField	-	-

Table 8. The fields used in Queue model

field name	field type	constraints	delete behaviour
reservation_length	DurationField	must be under maximum duration	-
datetime_creation	DateTimeField	-	-
datetime_closed	DateTimeField	-	-
configuration	ForeignKey to Configuration model	-	set null
reserver	ForeignKey to User model	-	set null
pools	ManyToManyField to Pool model	-	set null
labels	ManyToManyField to Label model	-	set null
power_off_automatically	BooleanField	-	-
free_automatically	BooleanField	-	-
created_reservation	OneToOneField to Reservation model	-	set null

Table 9. The fields used in Port model

field name	field type	constraints	delete behaviour
type	ChoiceField	required, choices: <ul style="list-style-type: none"> • power • fiber • hdmi • serial • generic 	-
value	CharField	-	-
name	CharField	-	-
powered	BooleanField	-	-
hardware	ForeignKey to Hardware model	required	cascade
connected_to	OneToOneField to Port model	-	set null

Table 10. The fields used in Pool model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-

Table 11. The fields used in Label model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
description	CharField	-	-

Table 12. The fields used in VirtualMachine model

field name	field type	constraints	delete behaviour
domain	CharField	unique, required	-

Table 13. The fields used in IpAddress model

field name	field type	constraints	delete behaviour
ip_address	CharField	unique, required	-
slug	SlugField	unique	-
unique	BooleanField	default = true	-

Table 14. The fields used in SWInfo model

field name	field type	constraints	delete behaviour
name	CharField	required	-
value	CharField	required	-
datetime_modified	DateTimeField	-	-
configuration	ForeignKey to Configuration model	-	cascade

Table 15. The fields used in Parameter model

field name	field type	constraints	delete behaviour
name	CharField	required	-
value	CharField	-	-
description	CharField	-	-
configuration	ForeignKey to Configuration model	-	cascade

Table 16. The fields used in Country model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-

Table 17. The fields used in City model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
country	ForeignKey to Country model	-	cascade

Table 18. The fields used in Room model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
city	ForeignKey to City model	-	cascade

Table 19. The fields used in Rack model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
room	ForeignKey to room model	-	cascade

Table 20. The fields used in Shelf model

field name	field type	constraints	delete behaviour
name	CharField	unique, required	-
rack	ForeignKey to Rack model	-	cascade