Using Django and JavaScript to implement Model-Based Performance Testing as a Service with the MBPeT tool

Joana Martini

Master of Science Thesis
Supervisors: Adj. Prof. Dragos Truscan
MSc. Tanwir Ahmad
Software Engineering Laboratory
Faculty of Science and Engineering
Department of Information Technologies
Åbo Akademi University
May 2019

ACKNOWLEDGMENTS

I am deeply grateful to my supervisor Dragos Truscan for all your invaluable help with the project implementation and the writing process. Your feedback was always priceless and fundamental for achieving a successful thesis work. I would also like to thank Tanwir Ahmad for your assistance and guidance on all the questions I had.

I must express my profound gratitude to my parents and my sister who have always been there for me. Without your endless love, support and encouragement, this achievement would not have been possible. You taught me everything, especially how to never give up. Thank you for guiding and inspiring me in life. My only wish is to make you as proud as possible. I love and miss you!

I am also grateful to my boyfriend Olsi, for walking with me during the university path. Thank you for always being there and believing in me when I did not. Your overwhelming support and presence allowed me to overcome every difficulty I encountered during these years. I want to thank you for never losing hope and for bearing with me in my worst days. I love you every day more!

Finally, I am thankful to my best friend Nino. You have always been the brother I never had. Thank you for listening and showing interest in everything I do, even without understanding much. Our conversations have been the nicest distractions from the thesis.

ABSTRACT

The work behind the thesis consists of the implementation of Model-Based Performance Testing as a Service using an existing tool, named MBPeT. The current tool's usage is achieved by the local installation and execution. The new system dismisses the user from this requirement and provides an online version where only Internet connection is demanded. The solution is also enriched with an interactive web application used to provide the data for executing the tests. We will mostly concentrate on the technologies behind the system, since using them is the key leading to a successful project.

Three different fields are correlated together in the system implementation. Model-Based Performance Testing is an approach used for testing the performance of a given web application, thus software testing is one of the studied concepts. Web application and the technologies used for their development is the second theoretical notion. Deployment on the cloud is required for achieving the online availability of Model-Based Performance Testing. Therefore, cloud computing is the last field we will focus on.

The practical section can be divided into two steps which can also be referred as the system's goals. The first one relates to the implementation of the web application for the test data. Django is the technology used for the back-end side, while JavaScript is the focus in the front-end part. The second step affects the deployment on the cloud and the communication between the web application and the existing MBPeT tool. Its distributed architecture eases and benefits the cloud deployment. Among all cloud providers, Amazon Web Services was chosen for the implementation. The reasoning behind every choice is explained through the thesis with features and characteristics of the chosen technologies.

To evaluate the fulfillment of the system's requirements, the last step is to run experiments on a specific test case and validate how well they are accomplished. From the results, it can be determined what was successfully implemented and if the right choices were made. It can also be comprehended what needs to be improved as a future work and what changes are required to provide a better solution.

Keywords: software testing, Model-Based Testing, performance testing, web application, JavaScript, Django, cloud computing, Software as a Service.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
AMI	Amazon Machine Image
API	Application Programming Interface
CSS	Cascading Style Sheet
DOM	Document Object Model
EC2	Elastic Compute Cloud
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
JS	JavaScript
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MBPeT	Model-Based Performance Testing
MBT	Model-Based Testing
MIDAS	Model and Interface Driven – Automated testing of Services architectures
MTV	Model – Template - Controller
MVC	Model – View - Controller
OS	Operating System
PaaS	Platform as a Service
PTA	Probabilistic Times Automata
SaaS	Software as a Service
SCP	Secure Copy
SSH	Secure Shell
SUT	System Under Test
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine

1. Introduction

1.1 Problem Description

During software development, one crucial process is testing the software. It is the process which provides valuable output to the user regarding the system under test and helps identifying defects and incorrect behavior of different nature. When testing the software, it can be determined if the software's specified requirements have been followed and implemented as expected. The requirements can be divided into two major groups, functional and non-functional requirements. The first type mainly deals and checks if the software works as described in the behavior of the system, while the second one elaborates and analyzes specific characteristics of it, not related to the functionality. Typically, the following non-functional requirements are inspected among others:

- Efficiency
- Fault tolerance
- Maintainability
- Quality
- Reliability
- Response time
- Resource utilization
- Scalability
- Security
- Stability

Non-functional requirements are as important as the functional ones. A system cannot be considered as correctly working if it takes too much time to respond, or even stops working due to an overload.

Non-functional testing includes a wide variety of characteristics that can be checked while testing a system. But, on this project, we will narrow the set of requirements to the ones related to our focus, the performance testing type. It is defined as a testing approach which studies how the system under test performs with respect to *response time*, *stability* and *resource utilization* under a specific workload. The workload should imitate as much as possible the real workload that a system can encounter.

To achieve and perform this testing, there are different set of tools which can make the performance testing more effortless. Some of the tools are: httperf, JMeter, Selenium and LoadRunner. The shortcoming of using them is the fact they implement tests by using manually scripts or pre-recorded cases. This goes against the principal of mimicking the real workload as many inputs can stay not tested.

For this purpose, researchers at Åbo Akademi University developed the tool named MBPeT, Model-Based Performance Testing which is also used to achieve the goal of the thesis. Its target is to implement load generation towards the system under test and monitor different *key performance indicators* (KPIs) as response time, CPU, throughput etc. [1]. The term Model-Based comes from one of the testing techniques, widely used nowadays. Definition and explanations about MBT are provided in Section 4.3.

One limitation that can be directly noticed when working with the tool, is related to its usage. It can be run only locally and needs installation prior to it. This has some specific drawbacks which can be avoided by our proposed solution, which is presented in the next section. Some of the most obvious drawbacks are related to the fact that the tool requires a lot of CPU capacity and computing power as well.

1.2 Goals

The objective of this project is presented shortly and straightforwardly in the thesis topic. The main idea is to provide the Model-based Performance Testing approach as a service using the existing MBPeT tool. This is related to the third type of cloud service model, named Software as a Service (SaaS). We will present SaaS's definition and how it works in section 3.1.2.

In order to have a better vision on what the goals are in this project, it is necessary to give a proper definition and explain the project further. Before running the test, the user has to provide the necessary test input data for the execution. Currently, this is done by using the command line interface or the provided Graphical User Interface (GUI). The objective is to expand its usage by creating an interactive web application for the input data and deploy it in the cloud. This will facilitate considerably its operation and will provide an easier and more user-friendly solution to run the tests. The user will specify the input data needed to run the tests and check the reporting and all graphs in real time. Since it is important to accomplish a great user interface solution, the focus of the thesis is towards technologies used and how well they fit for the implementation of this software as a service.

After providing the web application available online, the next prerequisite is to bring out the connection and the communication with the MBPeT tool. Currently, the tool needs to be downloaded and installed locally, but the new solution consists of deploying it on the cloud and achieve the communication with the web application. The web application connected with the MBPeT will be considered as a new system which will execute the tests online using the Model-Based Performance Testing approach. No interaction will be required from the user, except for providing the test data. The online version of the whole system will have more updated features which add more values to the existing tool.

Throughout the thesis, explanations and reasons why the proposed solution meets the project's requirements can be found with respect to different points of view. It is important to understand why it is significant to have the stated solution for this problem.

1.3 Related Work

In section 1.1, we mentioned that there are a lot of performance testing tools that are already available. What about model-based testing performance testing tools deployed as a service? What options are already ready to be used for this purpose?

During the research, we came across with some sets of tools that fit partly in what we were looking for. One of these examples is the implementation of Model-Based Testing as a service using one project named MIDAS [2]. Its main goal is utilizing the cloud computing scalability feature in order to transfer there the tests' execution. This comprises one of the close similarities to our system. As we will discover when discussing about MBT, it allows the generation of a large number of tests. Due to its characteristics, cloud computing introduces a plausible solution for overcoming the possible lack of resources during the test implementation.

The main idea behind the project is to provide the possibility of using MBT as a Software as a Service (SaaS). This constitutes another resemblance with the solution we want to propose and implement. Both implementations emphasize the importance of using MBT for software testing. It is still considered a new approach, but it is referred as a leading-edge technology in the industry [3]. To use it as a SaaS application, the users are then required to just subscribe to the service and input what is requested for executing and running the tests.

The fact that our solution is especially focused on testing the performance aspects of the system under test is what differentiates this implementation from our solution. Our proposed solution provides a clear vision of what the tester can achieve by using it and what areas of software testing are particularly tested, while the MIDAS project is too broad in its definition.

As mentioned in the paper, one difficulty that was encountered during the implementation was related to the licenses required for the tools executing MBT. We did not experience this problem due to the usage of the MBPeT tool, which was developed within the university, making the deployment straightforward and much easier.

1.4 Thesis Structure

As this project regards and involves different fields, the following three chapters will present all technical concepts that are required to get started. In chapter 2, the focus will be on presenting the concept of web applications, how they work, and which web technologies can be used to develop the web applications. The following chapter gives a brief presentation of cloud computing. It also presents the benefits and the risks of using Software as a Service, which is exactly what we wanted to achieve at the end. Chapter 4 introduces software testing, different types of software testing, based on different techniques to divide them. The focal point will be on model-based testing and performance testing as they are the key factors of the MBPeT tool. An overview of the tool, the introduction of its architecture and what the user needs to do to use it can be found at the end of the chapter.

The following chapters explain in detail how the development process was realized, starting from the followed requirements in chapter 5. Chapter 6 presents how the proposed solution solves the problem with a feasibility analysis. More information on the development environment, tools and libraries can be identified in the same chapter. It also includes all design stages that were followed to achieve the goal. The last chapter is related to the evaluation of the solution, with some experiments verifying that it works as expected. It also summarizes all the findings, difficulties encountered during the process, what improvements can be implemented and how well the objectives were accomplished at the end.

2. Web applications

In this chapter, the focus will be on the theoretical part of developing a web application. As the main goal of this project is to provide an interactive web application for this tool, we will get started by explaining the concepts behind this definition and how it works. The technologies used during the development, their architecture and explanations why they fit perfectly to achieve the stated objectives will be also presented.

2.1 What is a Web Application?

Generally, people tend to misuse terms like “web pages”, “web sites” or “web application”, considering them as the same thing, even though they are totally distinct. While web page is just a document / web resource on the World Wide Web, web sites are referred as different connected web pages. Nowadays, with all the dynamic resources and information on the web, the term web site evolved to a new one, to “web application”. But what is it and how does it differ from a web site? The definition from [3] gives a proper answer to both questions:

> “Web application is a client/server application that uses a Web browser as its client program and performs an interactive service by connecting with servers over the Internet (or Intranet). A Web site simply delivers content from static files. A Web application presents dynamically tailored content based on request parameters, tracked user behaviors, and security considerations.”

In order to be able to understand and develop a web application, the programmer should know about the so-called principles of web application and the explanation of the keywords present in the above definition.

2.2 Principles of Web Applications

There are some key elements that form the basis of a web application, the protocols and languages needed to make it work: HTTP, URI and HTML. In few words, the difference between them is as follows: HTML is a way to represent information, URI is a way to reference it while HTTP is the protocol used to exchange data. Detailed explanations can be found below:

HTML - HyperText Markup Language

HTML is a markup language which is used for formatting hypertext documents and navigating between web pages [3]. With its usage, information on the web is displayed in a human-friendly format. Web browsers obtain documents from a web server and render it as a web page for the end-user. With HTML the structure of a web page can be implemented, but additional technologies are then needed, like CSS for a better style and JavaScript for interactive functionalities.

HTTP - HyperText Transport Protocol

It is an application protocol used for transporting messages over the network [3]. In other words, it is what the web client uses to request and access the resources, a web page for example, to a web server. HTTP is the fundamental way of communicating in the WWW (World Wide Web) and can be used not only for hypertext messages, but also distributed object management systems.

It is known as a request-response protocol where the client sends a request to the server and then, it is the server which returns a response message to this client. The picture 1 shows how HTTP uses the request / response paradigm [3].

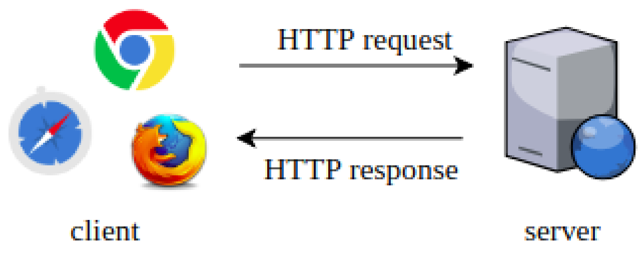


Figure 1: Communication between the client and the server

For each request, there is the need for a new connection between these two as it is interrupted as soon as the server provides the response. This indicates that it is also a stateless protocol [3] as there is no stored information between each HTTP request.

URI - Uniform Resource Identifier

The last principle that needs to be explained is related to how we can access the resources we are looking for in a web server. For this purpose, there is the URI which is used together with the HTTP to address the requested resources. In order to understand what URI is used for, it is important to understand two other notions, URL and URN, because URI is the combination of them.

URN stands for Uniform Resource Name and it refers to a fixed name of a resource and it never changes. But, the location of a resource frequently changes, and for this reason there is the need for a more general location reference, not dependent of the name. URL is the solution for this. It stands for Universal Resource Locator and specifies the location where a resource can be found [3], for example a web page in a web server. URI is the joining of URL and URN. The generic format of a URI can be found below, and short explanations related to each parameter as well. Not all URI are of this format, but this applies to the ones of our interest, HTTP(S). The example from [3] is of this format:

scheme :// host: [port#] / path / [?query-string] [#anchor]

It is composed by the following elements:

- scheme – the protocol that is used for the communication, e.g. https.
- host – the name or the IP address of the resource we want to access, e.g. www.imdb.com.
- port# – it is optional, and it defines the port number that the web server listens to, e.g. 80 for HTTP servers, so we can rarely see it.
- path – it is the location of the requested resource in the file system path from where the host resides, e.g. /list/ls040053231/.
- query-string – it is also an optional parameter to the URI and contains dynamic parameters that can be added to the request, e.g.?sort=user_rating,desc, to list the movies based on the user rating in descending order. The equal sign separates the parameter from its given value and by using &, we can add different key-value pairs.
- anchor – it is used for referencing a possible HTML tag/anchor/link inside the requested resource, e.g. #movies.

Complete example: https://www.imdb.com/list/ls040053231/?sort=user_rating,desc

2.3 Client-Server Architecture

After explaining and going through the principles of web application, it is also important to understand the expression client / server side which was mentioned in the definition above. The usage of the words client and server comes from the fact that Web is designed to have a **client-server architecture**. It consists of three parties communicating over the HTTP protocol.

Client is the first party, and, in most cases, it is the web browser or a computer from which the user makes requests to the server. The request can be made by surfing the web using the mentioned URI, by filling a form and many other ways. Then, it is the server which identifies the resource requested by the user, processes the data and returns it to the client [5]. The latter one displays the information to the user, as a web page for example.

The connection between the web browser and the web server can be observed in the picture below. There is also the third component of this architecture, known as the database. It provides and stores all the information which is needed for the application. It is the web server who looks through the database, and after getting the right requested document, it sends it back to the client.

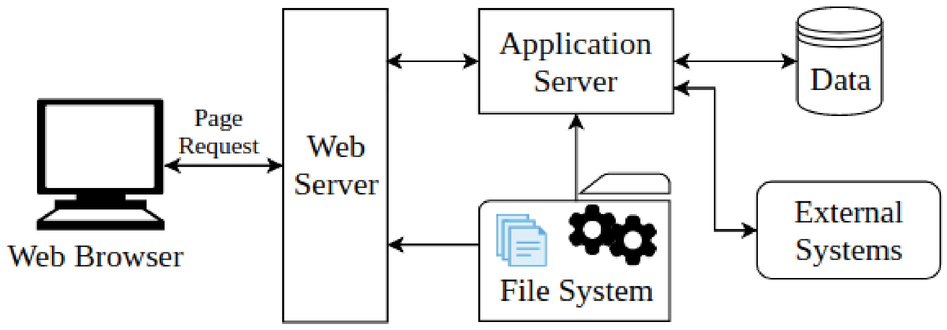


Figure 2: Client and Server Architecture

2.3.1 Client-Side Programming

As it can be easily deducted by the architecture description, client-side programming is related to the programs who are run on the client. The execution of these programs happens on the browser and this implies that they provide functionalities without accessing the server-side. This reduces a lot of time to the user, providing faster responses. It also minimizes overhead to the web server as, most of the times, there is no need to communicate with the server side or the database. [6]

It is the user who interacts with such programs, so the main idea when developing the client-side of a web application, is to make it as much user-friendly as possible. It should follow the known principle “look and feel”. The main tasks, that a program executed on the browser can do, are:

1. Validating forms filled by the user; checking if input fields are filled out properly.
2. Applying styles and different animations to the web application.
3. Communicating with the server and displaying the response to the user.
4. Controlling the elements present in a web page by providing responsive and interactive web pages adapted to different user’s devices.
5. Providing dynamic functionalities and content on the web page.

Client-side programming is composed by HTML, the markup language that provides the elements in a web page; CSS which applies styles to the HTML components and JavaScript, which is the main used programming language providing the functionalities in the web application. Nowadays, there are lots of JavaScript frameworks or libraries used not only as a client-side programming language, but also server-side one. Some of the client-side languages are AngularJS, ReactJS, jQuery, VueJS etc.

In this project, our focus is on the combination of JavaScript with jQuery and AJAX, so let us discover what they provide to the user.

2.3.1.1 JavaScript

JavaScript (JS) is mainly a client-side programming language and it has the following attributes: high-level, loosely typed, interpreted (not compiled) and dynamic [7]. Nowadays, it is a web standard supported by all web browsers. JavaScript code is incorporated in the HTML pages, so browsers can process it together with the HTML.

It is widely used for many purposes, which makes it suitable as a client-side programming language. Except for the main tasks mentioned in the previous section, JavaScript, as stated in [8], can also be used to:

- make web page more interactive for the end-users;
- insert dynamic content to the HTML, which is a big advantage when dealing with different web pages;
- use the large amount of JS libraries that help in achieving different functionalities. The libraries that are specific to this project can be found in the section 6.2.3.
- react to events fired by the user, like user clicking on a button or interacting with the document. At this stage, an event occurs, the JS code that is responsible for this event runs, and the document's appearance is changed based on the JS code's response.
- manipulate the DOM (Document Object Model) based on some conditions. DOM is the way of representing the document (web page) as a logical tree. By using JS methods, we can change the document structure, style or content. This is also related with the document changing due to an event.

The above reasons were decisive when deciding which client-side programming language to choose for the project.

2.3.1.2 jQuery and AJAX

jQuery and Ajax are different, but at the same time highly connected and this is the reason we decided to explain them in the same topic. jQuery is just a feature-rich JavaScript library that makes JavaScript functionalities easier and faster to handle [9]. Many tasks like DOM traversal and manipulation, animations or event handling can be achieved by jQuery in a few lines of code and much faster. One statement that is associated to it, is: "write less, do more" [9].

Another functionality that is made simpler by jQuery is related to Ajax, to the development of Ajax applications. But what is it? Ajax, *Asynchronous JavaScript and XML*, is a client-side technology which allows loading data in the background and update them on the web page, without reloading the whole page [10]. It asynchronously makes the client's request using HTTP Get and Post, connects to the web server, and returns the response in XML or JSON (mostly) format [11]. It is widely preferred over the normal JS calls, as data can be read from the web server even after the page has loaded and send data back in the background. The external data can be directly loaded into the specified HTML elements of the web page. The loading can be easily achieved by using jQuery.

Since JSON is extensively used through the project, it is important to explain shortly what it means. JSON stands for JavaScript Object Notation and it is a lightweight format used for data interchanging [12]. It is created from the JavaScript programming language and the main intention was to be easy for humans to understand and machines to parse / generate. Examples of its format will be given in Chapter 6.

2.3.2 Server-Side Programming

After explaining and going through some technologies used in client-side, it is also important to describe what happens on the server-side and which programming languages are the most common ones. It includes all programs which run on the server and provide information which are then made available to the user using the client-side technologies. They accomplish the functionality and the usability of a web application. [13]

As explained in [6] and [13], the main tasks that server-side scripts will always do are:

1. Processing the user inputs that are submitted to the server using a form.
2. Bringing to the client the requested page by the user.
3. Interacting with storages like file systems, external systems or databases.
4. Making possible the connection of the browser to the database, where the data from the web application is stored.
5. Querying and operating in the database, based on the type of the HTTP request.

The obvious disadvantage that server-side programming can bring is the latency and performance decrease of a web application. As the information is being processed on the server-side, it can take some time to return it to the client. [6]

The main languages that are widely used today when coding on the server-side are: PHP, Python, Java, C#, Ruby and many others. Together with these languages, there are also their respective frameworks as: Laravel, Symfony, Ruby on Rails, ASP.NET, Flask, Django, Node.js etc. We decided to develop the project using the Django framework. In order to explain and understand how Django works, it is necessary to first explain what MVC is as Django is built on this architecture.

2.3.3 MVC Architecture

MVC, which stands for Model-View-Controller, is an architectural design pattern which is used to separate the content (data) from the presentation (UI) for a more modularized system [14]. Nowadays, this architecture is widely used when implementing web applications. As mentioned in the name and described in [14], it is composed by three components whose functionalities are highly connected with each other:

1. Model – It includes the data and the state of the web application. It communicates with the external data / storage / database.

2. View – It represents the visual interface by displaying the view selected by the controller as an HTML. It also requests updates and prepares data from the model.
3. Controller – The last component communicates with the user, managing his requests. Based on them, it sends a request to the model and selects the view that needs to be displayed to the user. It handles the web's functionalities and logic.

Picture 3 shows how the components relate to each other and interact together. It also shows the connections with the client and the database.

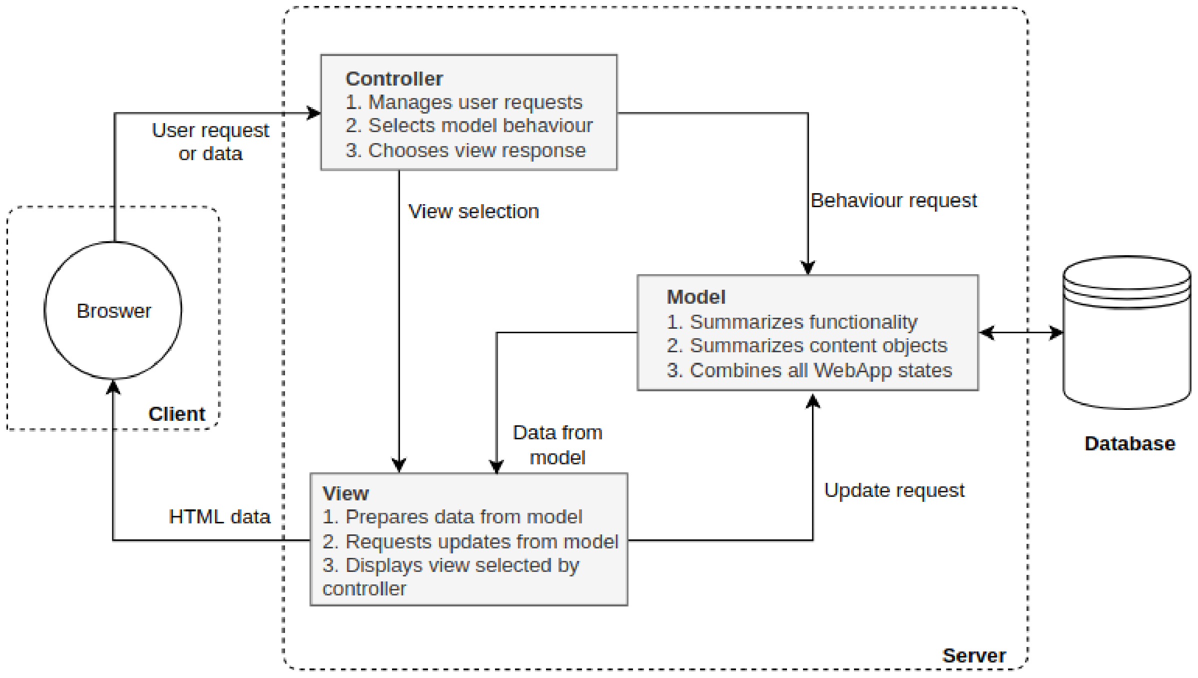

Controller
1. Manages user requests
2. Selects model behaviour
3. Chooses view response
User request or data
View selection
Behaviour request
Broswer
Client
Model
1. Summarizes functionality
2. Summarizes content objects
3. Combines all WebApp states
Database
Data from model
Update request
HTML data
View
1. Prepares data from model
2. Requests updates from model
3. Displays view selected by controller
Server

Figure 3: MVC architecture

2.3.3.1 Django, a Python Web Framework

Django is one of the widely used Python-based web framework. There are many reasons why it was chosen for developing this project. While searching for its attributes, the main ones that we come across are being fast, scalable and mature with many built-in components which makes the development easy. [15]

The main principle that Django uses is "Don't Repeat Yourself" (DRY). It is widely used, because of Django's code reuse, which minimizes repetition and allows to do as much as possible with few lines of code [16]. This also comes up from the fact that Django is created over the MVC architecture which encourages efficient code reuse.

So, as we just discussed, it is based on the MVC paradigm, but with a small change. The term MVC is changed to MTV which means: Model – Template - View. The Django model has not changed the functionality. It still works and deals with the database. The template is the component which displays the content to the user.

On the other side, the view is more than just controller's replacement. It communicates with the model to get data from the database, and after getting it, formats the data, creates a HTTP response object and returns it to the client [17]. So, the view is the decision maker on what data should be returned to the template. Views in Django can be created as a Python method or class and each view has its own template from which it gets input and returns some data.

Despite the main attributes, in my opinion, Django has many other features which, in my opinion, makes it preferable over other web frameworks. The following ones from [15] and [16] describe some of them:

- *Rapid development.* It allows developers being able to start and finish a Django project in a short time. This keeps in mind also the performance which is not affected at all by the speed an application can be developed.

- *The template system.* It is related to the DRY principle, because the templates use the template inheritance, which avoids repetition of redundant code.

- *Straightforward database and table generation.* After the database layout is defined, the relation between models and which fields are included, using the *migrate* command creates automatically all the tables that did not exist before. It also provides a variety of options for the programmer to define the models.

- *Security.* Django provides a lot of ways to protect your data, your accounts and passwords. It protects against SQL injections, clickjacking, CSRF attacks and many others.

- *Scalable.* Django's capability for flexible scaling to meet traffic demands and requests is a big advantage for the framework. This comes from the fact that its components are separated in different layers.

3. Cloud Computing

After describing one of the three theoretical concepts of this project, it is time to proceed with the following one, cloud computing. This chapter starts with the theoretical notions behind the term cloud computing and its definition. It advances with the description and analysis on its main characteristics. These features are indispensable as they differentiate it from the other types of computing. The last two sections focus on how cloud computing can be offered as a service and what alternatives of cloud services are provided to the users. There is also a comparison between these different options which explains what differs one from the others.

3.1 What is Cloud Computing?

Nowadays, it is inevitable to not have heard at least once the terminology "Cloud Computing", especially in the IT department. It started emerging in the early 1960s, but the term began to be used in 1997 [18]. Since its beginning, it has progressed with huge steps, playing an important role in IT and not only. But what does Cloud Computing mean? How can cloud and computing relate together? Why is it such an innovative technology?

Many definitions can be found regarding cloud computing, but the one given by National Institute of Standards and Technology seems to be widely accepted and used. [19] states that:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

There is also another straightforward definition from [20]:

> "The delivery of computing services - servers, storage, databases, networking, software, analytics, intelligence and more - over the Internet ("the cloud") to offer faster innovation, flexible resources and economies of scale."

The usage of the term cloud arises as a metaphor for the collection of networks. Users can share data between them from anywhere and access cloud computing services whenever they need to. Nowadays, the only requirement in order to use these services is to have a web browser. We do not need to deal with buying, installing, configuring and managing the software and hardware. [21]. With cloud computing, the person / entity requiring the services can just subscribe and pay only for the services they have used, while in classical computing they must buy and own everything we need: hardware, system software etc. To have a clear understanding of the main differences between cloud and classical computing, please refer to the table below:

Cloud Computing	Classical Computing
Subscribe to services	Buy and own
Rent what you need (computing capacity and software)	Hardware, software, applications
Use	Install, configure, test, verify, manage...
…	Use
Pay for what you use based on the “pay-as-you-go" model	Repeat the above steps periodically

Table 1: Comparison between Cloud and Classical Computing [22]

Going back to the first explanation which is also referred as the official one, it can be noticed that it includes fundamental aspects and characteristics of cloud computing. With one single interpretation [23], it mentions and explains:

Five typical features: on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service.

Three service models: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Four deployments models: private cloud, community cloud, public cloud and hybrid cloud.

In the next sections, the focus will be on explaining shortly its characteristics and the 3 service models, especially SaaS, which is the one used for this project.

3.1.1 Features of Cloud Computing

As it was mentioned above, there are some aspects that make Cloud Computing unique and different from other Computing types. We will shortly go through the most significant ones presented by [23] and [24]. These features are also considered the cloud computing objectives.

1. *On-demand self-service.* Each user is provided with services they are paying for and they have the freedom to change the computing capabilities based on their needs. For example, they can change the software or network configuration.

2. *Broad network access.* As already mentioned above, the cloud services can be accessed by any device connected to Internet.

3. *Resource pooling.* This feature is related to the great availability of resources. The user has no real knowledge where his/her resources are located. If it is necessary, they can specify

the country from where they want their resources. But generally, resources are assigned to different users only based on their needs, not their location.

4. *Rapid elasticity and scalability.* Referred as two main advantages when using Cloud Computing, they allow the user to regulate the resource capabilities whenever is required. It signifies that there is no need to plan well in advance for resources as computing, networking and storage, as in other computing types. Resources and services can be supplied and released automatically, for example to cope with an increased number of users.

5. *Security.* Security of personal data has been and will always be a hot topic. Cloud providers can manage better physical and logical security than classical computing providers. Even if a server containing that data can be damaged, the data will be stored in other servers as well, prohibiting other users to access it. This ensures the privacy and security of user's data.

6. *Measured service.* The term "measured service" refers to the possibility of monitoring and controlling how many resources/services are being used until that moment. This ensures clarity between the user and the provider as cloud computing is based on "pay-as-you-go" model and the user will pay only for what has been used, no extra costs.

3.1.2 Cloud as a Service

There are three abbreviations that everyone must have run across while reading for cloud computing. These three acronyms are IaaS, PaaS and SaaS which respectively represent: Infrastructure as a Service, Platform as a Service and Software as a Service. They are the primary service models under which cloud computing solutions can be provided to users. So, cloud computing delivers them as services to users who subscribe based on the already mentioned "pay-as-you-go" model. [22]

In the picture below, we can distinguish the differences of what the user has to manage when using each cloud computing service model and the traditional IT. The latter one is frequently referred as on-premises computing.

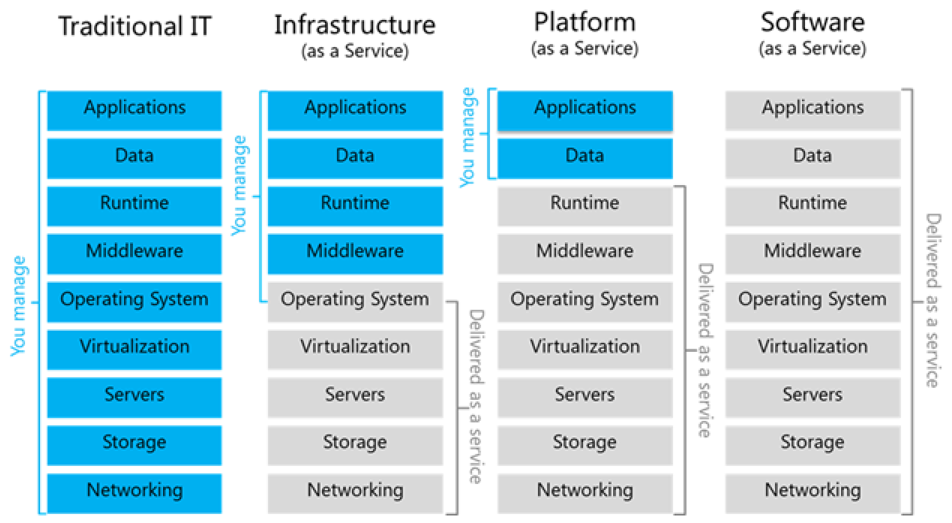


Figure 4: Levels of Cloud Infrastructure [25]

When comparing cloud computing with the classical one, we noticed that the user must own the software and the hardware in order to run the application on-premises (traditional IT). This can be clearly seen in the diagram, where the user has to manage everything; from the applications, data, to the hardware like servers, storage and networking.

To compare the service models provided by cloud computing, it is not enough to just refer to the picture. So, we will explain each model and how they differentiate from each other.

Infrastructure as a Service (IaaS) IaaS delivers virtualized IT infrastructure resources to their customers. These resources include virtualized hardware like computing, storage, CPU type, networking and sometimes OS. [22] They can subscribe to a chosen environment with some specifications and set up their applications on top of that environment. One of the most famous examples is Amazon Web Services. It provides an enormous number of virtualized infrastructures. Using the service Amazon Elastic Compute Cloud, the user can create virtual machines which are called instances or store data by using Amazon Simple Storage Service. [23]

Platform as a Service (PaaS) PaaS is the other type of cloud service in which the user has more limited control than in IaaS. It provides a cloud platform with a specific environment: OS and runtime libraries; and allows the users to develop and deploy their applications on top of it. As we can imagine, the service provider manages the resource provisioning, but also the programming language to be used. It offers some programming APIs and an application framework that will be used on the application. [23] Some well-known examples of PaaS are Google AppEngine, Microsoft Azure, Heroku and many others.

Software as a Service (SaaS) SaaS is the easiest service model to be understood on what it offers. They are software applications provided as a service to the users. Costumers do not take care of anything; they just need Internet to access it from anywhere at any time. It is the service provider who runs the web application or web service. [19] Users do not need to install and run anything locally, but we will discuss in the next section about the benefits that SaaS brings. Some of the most common examples that everybody must have used once are Gmail and Google Drive.

3.2 What does SaaS offer more than on-premises?

The goal of the project, which is to provide the Model-Based Performance Testing as a service, is directly related to the third type of service model, the SaaS. The model “pay-as-you-go" will be also applied as paid cloud services will be adopted when using it as a service.

One question that might easily arises is WHY? Why is it important to provide it as a service? What are the benefits compared to the existing one running locally? Is it just to avoid the installation or there are other advantages? To answer these questions, it is indispensable to compare SaaS towards the on-premises computing and check the advantages and disadvantages of each method.

Few years ago, it was more common to buy and own your own hardware and software. The user was responsible for downloading and installing it on his own machine and this method of software delivery is known as “on-premises”. But, since the moment cloud computing started to gain popularity, things took a different direction. Nowadays, SaaS is growing so fast inside the cloud computing industry. Businesses tend to adopt the latter method of software delivery, but is it the right choice to do or it is just a decision based on its fame? [27] There are many factors that need to be considered in order to make the right choice, because there is no exact answer to which one is the best.

The following factors from [26] and [27] were also considered when starting this project:

- *Implementation*. It is one of the main advantages, because SaaS are faster to be implemented with respect to the on-premises solutions. It is the provider who takes care of all hardware and software, while the user just subscribes and uses the services.

- *Upgrades and maintenance.* If an upgrade is required to the on-premises, more time and money will be spent on providing the best resources and the user is in charge of it. While in the case of SaaS, it is the service provider who maintains, validates and checks the correct functionality of the software.

- *IT knowledge.* As an extension of the previous argument, technical knowledge is required, in order to take part in the upgrade and maintenance of the software. When using the SaaS, little or no technical knowledge is necessary, since it is the company providing the software which does all the support.

- *Cost.* Another important aspect that needs attention is related to its cost. The two methods differ in terms of expenses. It depends on user’s needs, but paying only for what we use, for example a subscription once per month, is cheaper than buying the whole platform with software and hardware since the beginning. SaaS pricing is affordable and flexible, providing different options for downgrade, upgrade or even interrupt the subscription and the whole services. On the other hand, the on-premises solution requires higher charges due to continuously support and maintenance.

- *Scalability.* Scalability is one of the most obvious advantages of cloud computing, for instance, also of SaaS. The scalability, as the previous services, is managed by the provider, so the user does not handle it. SaaS's solutions are just scaled up and down to adopt to the application's usage, without extra costs in time and money. While in the on-premises software, the provider needs to plan ahead for all needed resources.

Despite the above advantages, there are also aspects that benefit the on-premises software. We will go through some of them, but our solution is not directly affected from those:

- *Security.* Security issues are often relative and depend on the business (user) using the software. SaaS offers, of course, secure data and privacy as there are providers who take care of it. On the other hand, in the on-premises solutions, privacy and secure requirements can be extended and enforced as the user has control over it.

- *Software customization.* This feature is related to the opportunity of making changes to the software based on user's needs. Customization is fairly easy when it is the user who owns all resources of the software. This means that on-premises solutions are simpler to be customized. This is also related to the fact that the user can have full control and access to the provided resource depending on the software's license.

4. Software Testing

4.1 What is Software Testing?

Software testing is one of the phases in the development process of a software. Depending on the development model that is being used, testing is held in different stages. For example, in Waterfall Model, which is one of the most common development processes, testing is handled only at a specific phase of the cycle, the testing phase. [28] While in the V-Model, the development and testing occur in parallel and for each test level there is coming information from the corresponding development step. [29]

A question which instantly arises is related to the definition of software testing. What is software testing and why a lot of developers consider it the most challenging phase during the development process? We can come across a lot of definitions and all of them represent different aspects of software testing. Some of the most explanatory definitions are the following ones:

> "Software testing is the process of executing a software system to determine whether it matches its specification and executes in its intended environment." [30]

Another straightforward and easy definition to understand, is:

> "Testing is the process of executing a program or system with the intent of finding errors." [31]

To make the idea clearer of what software testing is, there are also explanations related to what it is not. Errors, in many ways, are almost always present during the software development cycle. So, software testing is NOT when we show that there are no errors, but instead when we are able to find them. This is when it can be stated, that testing was successfully executed.

4.1.1 Why is it important to test the software?

Many reasons and factors should be considered when referring to the usefulness of software testing. One of the most obvious advantages that comes directly from successfully executed tests is the quality improvement. Identifying errors makes possible the correction of what was wrong. So, in the next testing execution, if fewer defects are found, it portends that the product quality has been enhanced. [30]

Errors can have different degrees of impact in the behavior of the software, from severe, major to minor. The outcome deriving from these failures can affect, for example, in terms of safety and economy. [32] This emphasizes the importance of testing the software before release. Planning the tests and executing them in the right moment helps in reducing costs and meeting customer requirements.

Another circumstance when testing becomes crucial, is when implementing a new feature / task for the project. After all, we need to be sure that it did not break any previous functionality, and everything is working correctly and as expected.

4.1.2 Testing at each Development Phase

There is one model named V-Model development model, where testing is done on each phase of the development. It is one of the most common way of testing as it has many benefits compared to the other processes. Testing starts in early face, from the requirements; and errors can be identified in each step and help decreasing the final number of errors. This benefits the process in terms of cost, but also reduces the time of development. [32] Each step of the development phase and testing can be viewed at the picture 5.

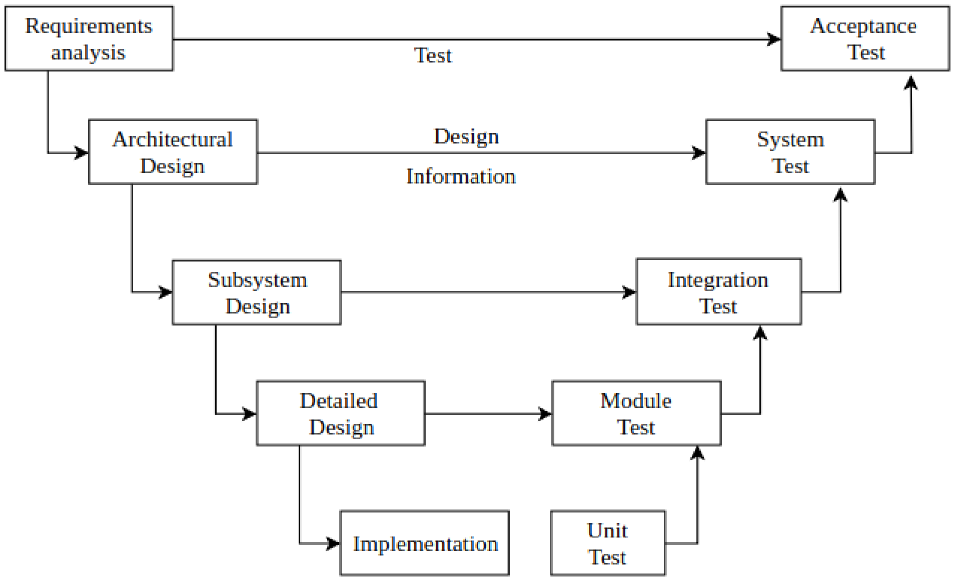


Figure 5: V-Model, a Development Process [32]

There are 5 stages starting from Acceptance Test which is related to the software requirements analysis and tests if the software will meet the user's needs. It continues with the System Test which is associated with the architectural design, and it tests if the complete system meets its requirements. The next one is Integration Test which connects to the subsystem design and tests the communication between the software's modules, the subsystems. After it, there is the Module Test related to the detailed design and tests the modules present in the software. Since the modules are mostly classes and packages, it is the programmer who does this type of testing. And, the last one is Unit Test, linked to the software implementation. It tests the units that are being implemented during this step, mostly functions and methods. [32]

4.2 Functional and Non-Functional testing

There are many ways of grouping the different techniques of software testing. For example, there is one division into levels, based on what phase of the development process as described above. We will concentrate on another type of analysis based on the purpose of the testing, in other words, what we want to test. There are two major techniques widely used in all levels during the testing process. They are named:

Functional Testing

It deals with the functionalities of the system being tested and what the software is meant to do. Testers run some tests against the software and then the obtained output is compared to the expected one for similarity. [33] In this way, defects and errors related to the functionality are discovered, but this is not enough. The quality of the product also depends on other aspects which can be identified using the second technique, the non-functional testing.

Non-Functional Testing

Non-functional testing consists of testing different aspects of the software, like the performance, loading, maintainability, usability and many more. As the functional testing, it also can run at all test levels. [33] Errors found are not only related to the functionalities, but also to non-functional characteristics like the ones mentioned above. So, it is crucial to properly identify as many errors as possible in order to provide a better software quality. [1] The focus of this thesis is on the tool which implements exactly one of the non-functional testing techniques, the one related to the performance of the software. Before moving to the next definition, we need to discover what performance testing represents.

In other words, performance testing intends evaluating how the tested system will react under loads in terms of responsiveness and then, investigate the detected problems. [35] The tester specifies one workload and registers whether the obtained workload during the test matches approximately with the specified one. [1] During these tests, different attributes related to the performance can be checked, like response time, throughput, scalability, resource usage and many others.

4.3 Model-Based Performance Testing

In order to define and comprehend better what model-based performance testing is, we need to divide it into two terms: performance testing and model-based testing (MBT). We already explored in the previous section what performance testing is, so the focus will be on model-based testing.

Model-based Testing (MBT)

Software testing has always been evolving at a rapid pace. From manual testing to automation one, there is a new approach nowadays, named model-based testing. It is considered an improvement of the previous approaches. [34] There is one definition regarding model-based testing stating:

"Model-based testing is a testing technique where the runtime behavior of an implementation under test is checked against predictions made by a formal specification, or model."

Colin Campbell, Microsoft Research

The picture below shows the flow of actions when using MBT to run a test.

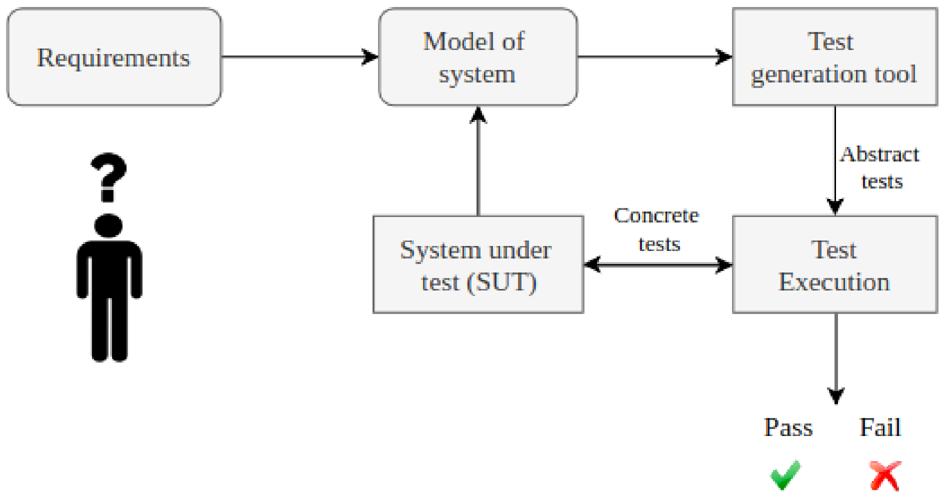


Figure 6: Workflow in MBT

But what is the formal specification (model) they are referring in this statement? How is it defined? Is it specific or general to the system under test (SUT)?

An abstract model describes the expected behavior of a SUT, for example what outputs should be expected in response to specified inputs. There is no exact way of defining the model, but it is clearly specific to the SUT. Based on what it suits better for the SUT and for the tests, there are different ways of defining the right model. For example, state table, finite state machines, probabilistic timed automata, Markov chains and many others can be used.

Publication [34] explains the steps in order to achieve Model-Based Testing:

1. Study the system/software being tested, the SUT.
2. Select the right model that suits the system.
3. Design and build a test model.
4. Select some test generation standards from all possibilities, which assures the desired effect from the test execution, e.g. coverage criteria.
5. Generate the tests. It uses the test cases implemented in the test model and runs as an automated process.
6. Run the tests on the SUT.

7. Analyze the outcome and check what errors were found during the execution. Some tests could have passed while others failed.

Therefore, model-based performance testing is a way of using the MBT approach to implement performance testing. More details about it will be described when explaining the Model-Based Performance Testing tool.

4.4 MBPeT Tool

As it is clearly affirmed by its name, Model-Based Performance Testing tool is used for analyzing the software / system performance using MBT. The tool is implemented using the Python programming language. Its main characteristics are load generation and system monitoring. As described in [1] and [36], the tool differs from the similar ones, because it allows virtual users (VU) to interact with the system by executing the models.

The defined models, from which the load is generated, use Probabilistic Timed Automata (PTA). What does PTA mean?

> "Probabilistic timed automata (PTAs) are a modelling formalism for systems that exhibit probabilistic, nondeterministic and real-time characteristics." [37]

Figure 7 shows one example of defining the models using PTA. We can see that a model using PTA has different locations (circles) and transitions (arrows) connecting the locations with each other. Each transition has its own label where three parameters can be found:

1. The probability of the action being fired. For example, from location 1 to 2, the left transition has a probability of 0.6.
2. The think time which is the time that PTA must wait (think about) until the next transition will be fired. Referring to the same previous transition, the think time is equal to 2. There are also cases where the think time is not specified, like in the transition from 4 to 5. In this case, a default value, or the one specified in the settings file will be used. More details regarding this can be found in section 4.3.2.
3. The action taken between these two locations, connected by the transition. One action we can mention is the one in the first transition: action1(). These actions are real operation executed by the SUT and the translation from action to operation understood by the SUT will happen through the adapters, which is another tool's module.

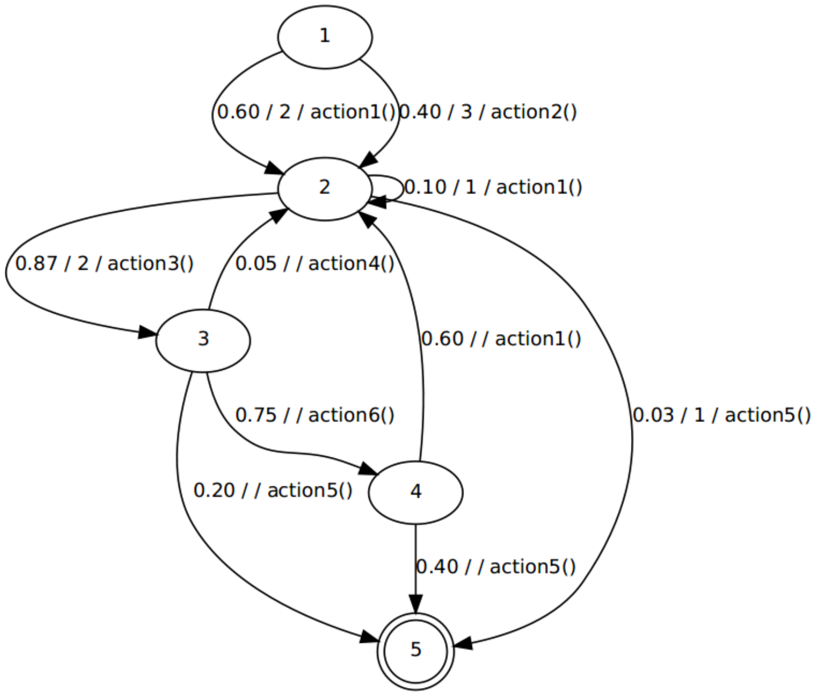


Figure 7: Example of PTA model

PTA is considered to suit to the SUT, because all its characteristics allow the VUs to act like a real user by deciding when to fire an action, with which probability and not follow static values. [36] The tool illustrates real life scenarios where all user input combinations are tested. [1] This makes a great difference from the other similar tools, which do not always follow this principle.

The tool has these 3 major functionalities [1]:

1. Generating load from the designed models and send it to the SUT.
2. Monitoring the Key Performance Indicators (KPIs) the tester is interested in. The most common ones in testing and which can be used with this tool as well, are:
    a. Mean response time
    b. Maximum response time
    c. Number of concurrent users before the system is corrupted
    d. Throughput, number of requests per second
3. Creating an HTML file with all results generated from executing the tests.

4.4.1 Tool Architecture

MBPeT tool has a distributed architecture [1] with a master node and several slave nodes connected as in the picture below.

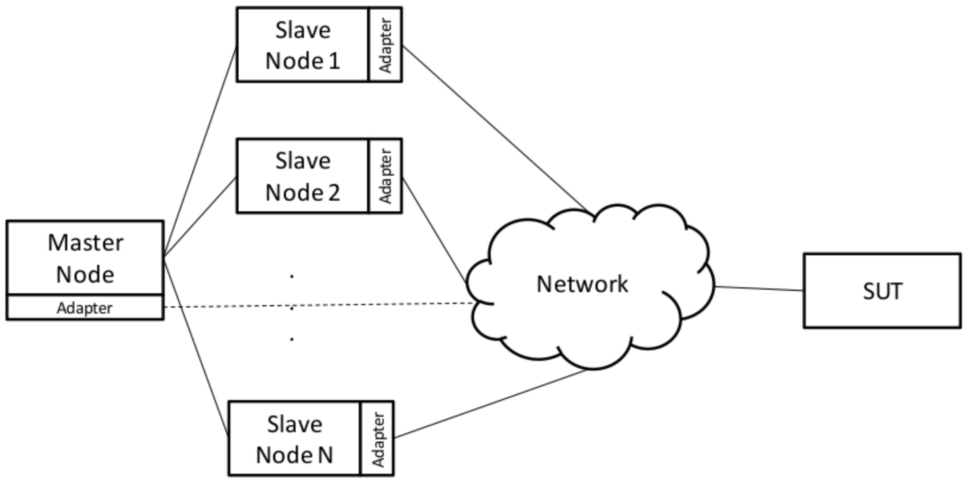


Figure 8: The distributed architecture of MBPeT tool [1]

It is the master node who gets all parameters to make the tests running and forward them to those slaves that will be used. After the slaves receive the proper input from the master node, they will be ready for load, but only the first node will be loaded from the master. The rest of the nodes will wait for their turn, until the previous one is saturated, determining that this node cannot be used for load anymore. The slave nodes are the ones who check the monitoring and report all information to the master node. The latter one creates a report which can be downloaded at the end of the test.

There are different modules present in the Master and Slave Node, but it is out of the scope of the thesis to go through all of them. We will explain those modules of our interest and consider the rest of them as a black-box.

Master Node

The architecture of the master node is shown in Figure 9. The explanations of how the master node works and its main functionalities can be found below.

As already mentioned before, the master node takes as input the *Models* and the *Configuration* file which are fundamental to run the test and distributes them to the appropriate modules inside itself. Before being forwarded to the slave nodes, the models are tested for some syntax analyses and checks. If these analyses are passed with success, the models and the configuration file will be transferred to another module inside the node. The current module is the link between each slave and the master node. Their number is the same as the number of slaves (number defined by the user) and their presence is crucial, so that the master node is not required to deal with all slaves and can progress with other tasks as well. This architecture boosts its scalability and parallelism.

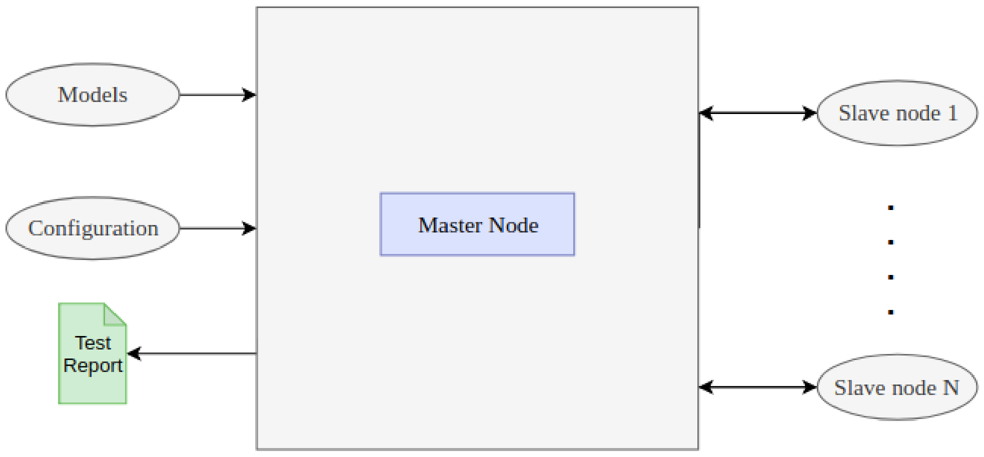


Figure 9: Master Node Architecture

The last functionality provided by the master node occurs at the end of the test session. There is one component of the node that collects all results from each slave and combines them into one unique test report which is then demonstrated to the user. Different graphics, analytics and results about the selected KPIs can be visualized in this test report.

Slave Node

In this section, we will explore the architecture and the functionality of the slave node. Its architecture can be seen in the Figure 10.

All slave nodes are initiated with a specific command and an argument passed to the command, which is the IP address of the master. It is the core of the slave node which opens a connection with the master, after receiving the command. When the connection is achieved, the communication between the master and slave node will ensue.

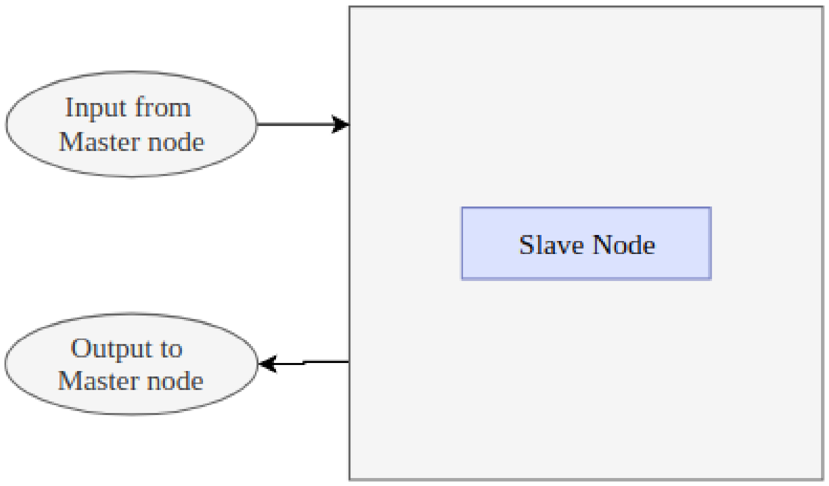


Figure 10: Slave Node Architecture

After load is generated and the test is executed, there is one component inside the node which collects all results from the session and creates a file for each simulated user. This functionality is similar to the one of the master node. The files are then compressed into a zip file and sent to the master at the end of the test execution.

4.4.2 Implementation of tool's functionalities

In order to start executing the MBPeT tool, there are some steps that need to be followed. The process, named "Test Setup" is divided into 3 major steps as [1] describes:

Defining models. In this step, it is important to define the models using the PTA for each user. Tests can have multiple categories of users and each test must have a specific user types model, which displays the distribution in % of each user category (type). The name of such model is also *root model*. Despite the root model, there are as many models as there are user categories (types). They are all defined using the DOT language. DOT language is one of the graph description languages [38].

Configuring the settings file. There are some parameters that need to be defined in order to continue with the testing. They can be given using the Graphical User Interface (GUI), or directly in the settings file. It will be our target in the next chapter, when presenting the project requirements.

Explanations are given below for some parameters:

1. Test duration

2. Maximum number of concurrent users
3. Ramp function. It specifies the number of users at specific moment during the test execution. It is dynamic, as the number of users can increase or decrease during the whole test.
4. Dstat. It is a Linux utility used to monitor resource utilization of the SUT. MBPeT has the ability of parsing the dstat file and present the results in the test reports at the end. This parameter can take three options:
    a. *SSH (Secure Shell)* if dstat is installed on the SUT and credentials (username and password) are provided in this settings file. During the test session, the MBPeT master will login into the SUT using SSH and run the dstat.
    b. *file* if it is not possible to login into the SUT. In this case, the tester can execute manually dstat in the SUT and then, provide the file to the MBPeT master.
    c. *none* if dstat cannot be installed on the SUT or the tester is not interested on this specification.

    Since SSH will be mentioned and used throughout the implementation, it is important to present its definition. The definition in [39] states that:

    > "SSH, the Secure Shell, is a popular, powerful, software-based approach to network security. It is a protocol and gives a specification of how to conduct secure communication over a network."

    Its main usage consists of login into remote systems, file copying and invocation of remote commands. All these applications have in common the fact of being secure. Our focus will be especially on the remote logins, since we need the communication between different servers. More information can be found in Chapter 6.
5. Threshold. It defines how much resource can be loaded into a slave node.
6. Monitoring Interval. It establishes the periodic interval that the slave will be checked if it has reached the saturation value, known as the threshold.
7. Think time. It is optional and used if there is no think time specified in the PTA when building the model.
8. Standard deviation. Its purpose is the same as the think time. It is used to define the think time for models which lack of it.
9. Root model. It specifies which model is the one containing all types of users for this system and their distribution.
10. Models folder. It is the directory where all specified models are stored.
11. Reports folder. It is the folder used to store the report generated at the end of the test session.

Define adapters. It is a module used to interact with the system under test. It includes files that are specific to the system, in XML or python format. These files contain the translation of each action found in the user model into operation that are comprehended by the system. Examples of adapters will be given in Chapter 6.

5. Requirements for MBPeT as a Service

In this chapter, we will present the purpose of building this system, explain the provided services to the user and indicate the target group of users. On a general overview, the project requirements refer to the tasks that need to be concluded in order to consider the system as successfully functional. In this way, the designer, the developer and the user, have a clear picture of the system's goals and objectives. From the general point of view, we will introduce the functional and non-functional requirements in detail. The first one specifies what functionalities will be supported by the system, and how the user interface supports the user in achieving the requested services. The latter type of requirements points out how the system should perform an action, or, in other words, it describes "the quality of the system". They can be related to the system's safety, security, performance, usability, maintainability, quality, testability and other similar specifications. At the end of the chapter, we will display how the workflow of the system's services is designed to be working in order to accomplish the functional and non-functional requirements.

5.1 Project Requirements Specification

As introduced in the first chapter, the scope of the project is to provide Model-Based Performance Testing as a service using the MBPeT tool. Additional features will be added to the current provided tool which will enrich the software functionalities.

The software system will be designed to allow users to run different test sessions, generate load towards the system under test and follow the monitoring in real-time. The implemented web application will provide a graphical user interface where the user(s) can specify all input data and parameters required for executing the tests. After specifying all parameters successfully, the user can visualize and monitor the execution of the tests in real-time. Many graphs and tables will show the current values of various KPIs like throughput, response time, disk, CPU, network etc. At the end, a report containing all information about the KPIs and other values is displayed and can be downloaded locally as well.

As the main objective of the project is to ease the tool's usage by supplying it online, the target group of users will significantly expand. Currently, it has a limited usage due to the local installation and only specific researchers at the university can use and are using it. With the new implementation, only general IT knowledge will be required to run the tests and the user-friendly web application will make the user experience smoother. It can also affect the industries which seek new methods for testing their software. As the tool is based on the MBT technique, companies can take advantage of this and move from the automated testing to this approach, which is considered to be better than the previous ones. Many researchers have arrived at the conclusion that using MBT can lower the testing costs [40].

5.2 Functional Requirements

It is important to go through all system's requirements and present to the stakeholders what functionalities are provided in this project. There are different panels present in the system, and for each one there are some services that the users can utilize. All services are listed underneath, and diagrams explaining and displaying the connection between them can be found in the last section.

1. User Interface
    a. Any person with access to a web browser and Internet can register.
    b. A registered user can login and access the web service.

2. Test Session
    a. The user can create one or more test sessions.
    b. All needed artifacts (models, test configuration, adapters) in a test session are stored until deleted from the user.

3. Dashboard
    a. Only logged in user can access the Dashboard Interface.
    b. They should be able to provide all required inputs in Models / Test Configuration and Adapters tabs.

    or

    c. Upload the whole project containing the necessary files.

4. Models Interface
    a. The user can upload each model file in .gv (.gviz) format.

    or

    b. The logged in user should be able to create user models for each test session in two ways:
        i. Writing the code of the model in DOT language in the online code editor.
        ii. Drawing the model interactively, adding edges and nodes, editing nodes, edges and labels.
    c. The code in the online editor and the drawn model should always be synchronized.
    d. The user can delete one or more model file(s).
    e. Every file can also be downloaded in .gv format.

5. Test Configuration
    a. The logged in user can upload the Settings file in .py format which defines all necessary parameters used in the experiment.

       or

    b. The test configuration file can be created in two ways:
        i. Filling out required or/and optional fields of the form, providing all required parameters for the test execution.
        ii. Writing the code of the Settings file directly in the online code editor in python language.
    c. The code in the online editor and the filled form should always be synchronized.
    d. After saving the filled-out form, the user can download the file in .py format.

6. Special input in Test Configuration: Ramp function
    a. The user can specify the ramp function as an array of tuples.

       o*r*

    b. The ramp can be drawn graphically.
    c. The drawn ramp and the one defined textually are always harmonized.

7. Special input in Test Configuration: Target Response Time
    a. The user should be able to define the TRT textually in JSON format.

       o*r*

    b. The user can define the TRT interactively by using the popup form.
        i. Actions can be imported from the models stored in the database or just created by specifying the name.
        ii. Values of action's name, average and medium time can be changed in the input fields.
        iii. One or more actions can also be deleted.
    c. Both formats need to be matched and checked for syntax errors.

8. Adapters Interface
    a. The user should be able to upload the adapters in python or/and XML language.

       or

    b. The adapters can be written directly in the provided online editor, the first editor supporting python language and the second one XML.
    c. The adapters are complementary of each other, they are not in synchronization.
    d. After creating the adapters, the user can download them.

9. Monitoring Interface
    a. The user should be able to view and monitor in real-time graphs displaying different KPIs.
    b. The KPIs are:
        i. response time for each action and combined option,
        ii. throughput,
        iii. memory,
        iv. CPU,
        v. disk and
        vi. network utilization.
    c. Information about the slaves' status will be displayed, showing which slave is allocated, idle, generating or saturated.

10. Reporting Interface
    a. The final reports created after a test execution can be accessed in the web application
    b. The user should be able to download all the reports.

5.2 Non-Functional Requirements

Non-functional requirements are as important and critical to the system as the functional ones. They fulfill the functional requirements by describing how they should behave. The non-functional requirements are firmly related to the architectural decisions and they support each other in this aspect.

As we described in the chapter's introduction, there are many non-functional requirements that can be considered when developing a project. In this system, the focus will be towards the following ones:

1. Security
    a. Only logged in users can use the system. Registration prior to login is required in order to access the application.
    b. Accounts and password details are secured and encrypted so nobody can retrieve them, neither the system's administrator.
    c. Data about tests are private and can be checked only by the tests' owner.

2. Usability
    a. The software should follow the principle "look and feel", meaning that it should be easy and user-friendly to be used by everyone.

   b. The forms filled by the user should follow the MBPeT's workflow, starting from specifying the models, defining the test configuration and the adapters. After the test is initiated, the following process is the real-time monitoring of the test execution. When the test successfully ends, a report containing different information is generated.
   c. Validation error messages should be displayed to the user explaining what the user did wrong to avoid it in the future.
   d. Documentation should be provided on the web application, so the user can search for needed information or help in using it software.

3. Performance
   a. The growing number of simultaneous users should not increase the response time of the application.
   b. The response time of the web application for different operations should be less than 1 second.
   c. Monitoring in real-time should not include any delay and the update of KPIs and graphs should be done every 1 second.

4. Maintainability
   a. The system should be developed in the way that allows future improvements and the implementation of new features.
   b. The ease of correcting possible defects should be also taken into consideration.

5.3 Workflow Diagrams

As the login and register services follow the logical and well-known workflow, we will concentrate on describing and displaying the workflow of the other services offered in this system.

1. Creating a test session

The following diagram displays the workflow when the user creates a new test session. At this phase, the only thing required from the user is to provide a unique name for the test. As we see in the diagram, it also shows when an error can occur and what happens when the test is successfully created.

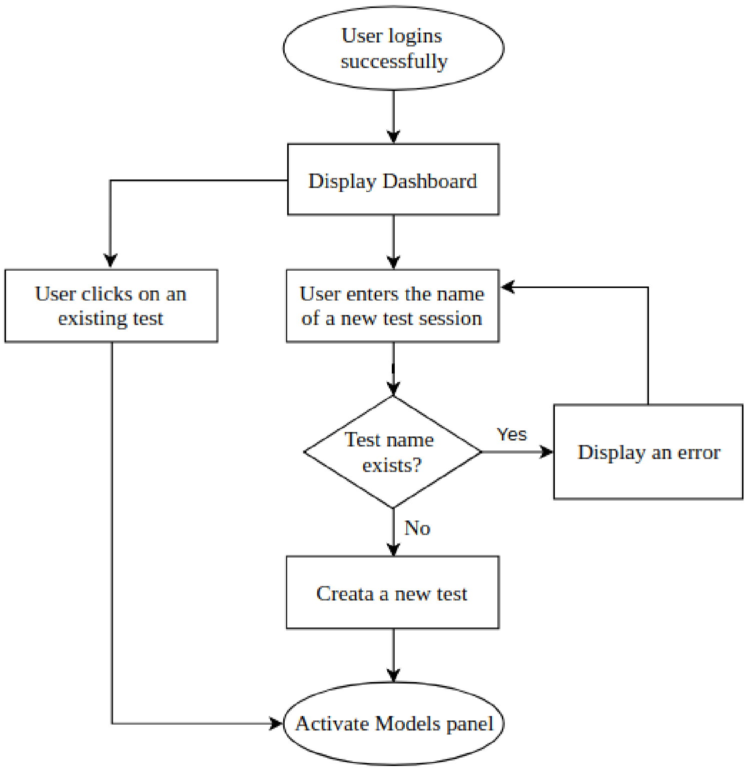


Figure 11: Test Creation Workflow

2. Actions in the Models panel

The Model panel will consist of different actions that can be performed by the user in order to fulfill the functional requirements. The following diagram shows some of the most important operations, like creating a new model, uploading it, deleting one/many and updating their content. As in the previous workflow, we can also view how errors are handled and during which wrong performed operations they can occur.

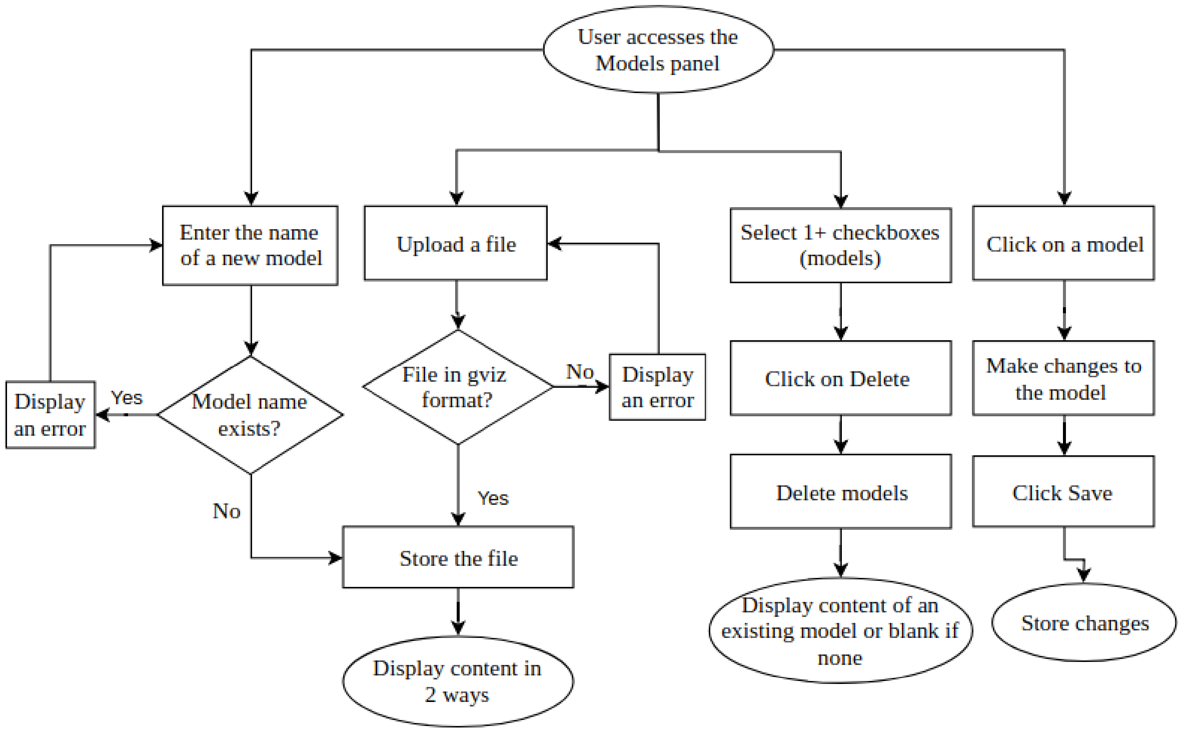
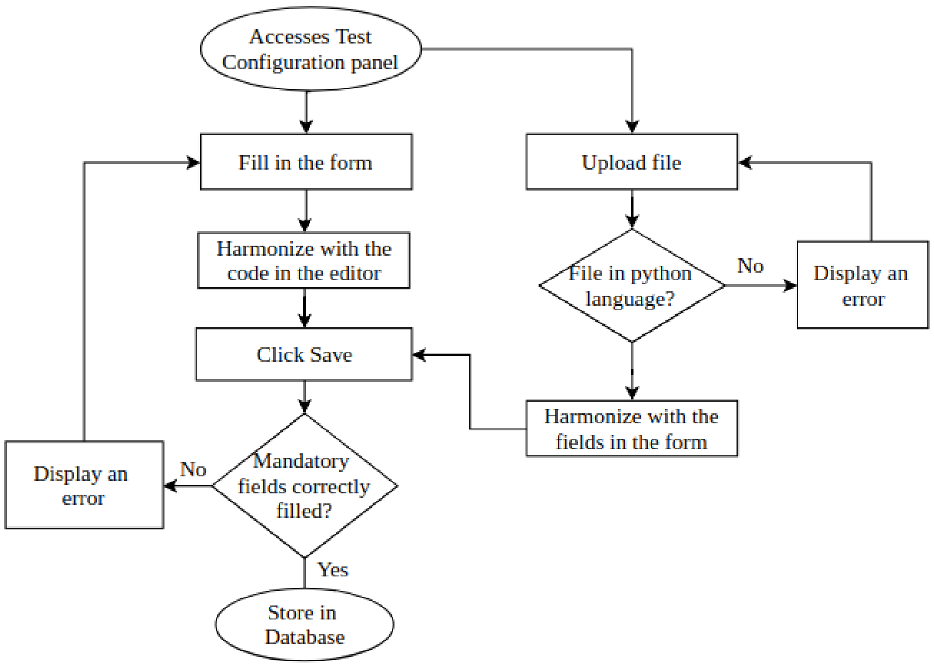

User accesses the Models panel
Enter the name of a new model
Upload a file
Select 1+ checkboxes (models)
Click on a model
Model name exists?
Yes
Display an error
No
File in gviz format?
No
Display an error
Yes
Store the file
Display content in 2 ways
Click on Delete
Delete models
Display content of an existing model or blank if none
Make changes to the model
Click Save
Store changes

Figure 12: Models Workflow

3. Actions in the Test Configuration panel

Since test configuration is related to filling correctly the fields in the form, the workflow of the actions is easier than the previous one. In the workflow, the actions of filling the form and directly uploading the settings file are shown. It also demonstrates in which cases the errors can arise.



Accesses Test Configuration panel
Fill in the form
Upload file
Harmonize with the code in the editor
File in python language?
No
Display an error
Click Save
Harmonize with the fields in the form
Display an error
No
Mandatory fields correctly filled?
Yes
Store in Database

Figure 13: Test Configuration Workflow

4. Actions in the Adapters panel

The adapters panel will allow users to upload files in XML and/or python language. It will also provide the possibility of coding directly there using the online editor. As we saw in the other examples, the errors can occur when the user tries to upload a wrong file format, neither in python nor XML language.

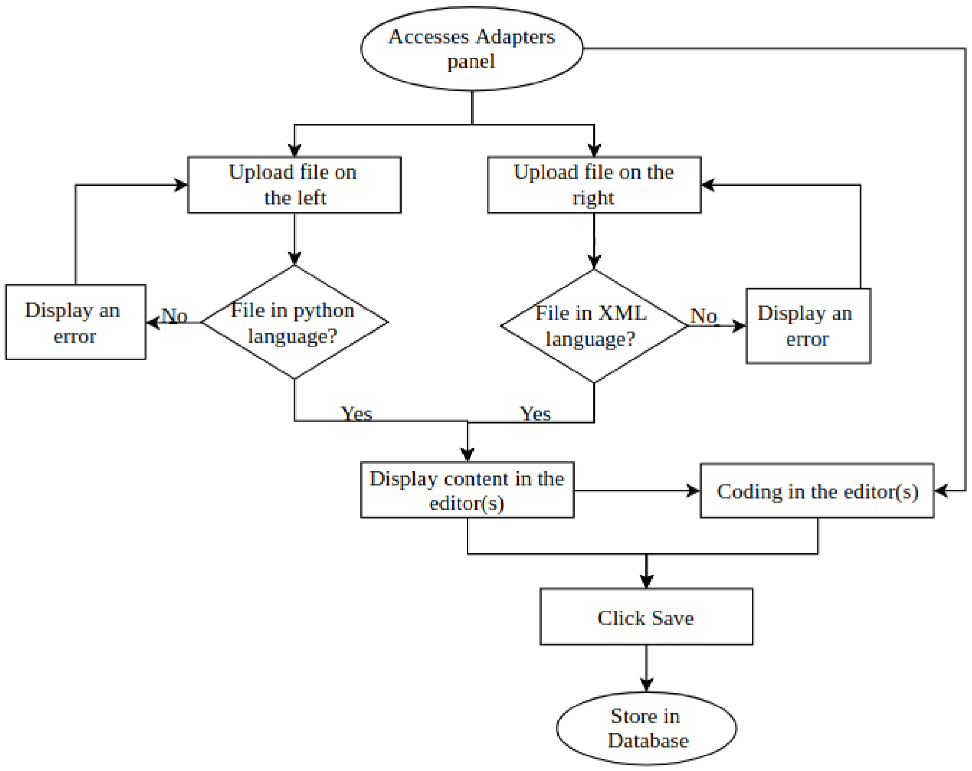


Figure 14: Adapters Workflow

6. Proposed Solution

In this chapter, the focal point is on the practical part of the chosen solution for the stated problem. After explaining all the theoretical concepts needed for understanding the system, it is necessary to analyze how the implementation was realized to obtain the result. We will start by presenting how the system architecture is designed and describing its main components. Afterwards, all technologies that have been used will be demonstrated. Each technology choice that was made is followed by a rational analysis. We will also present how the modules of the architecture have been implemented in order to perform correctly the communication between them, as stated in the architecture section. The chapter continues with a section where the focus is on the user interface of the web application. Information about the additional features and corresponding pictures of examples can be found in the same section. The last part of the chapter contains information about how the system deployment in the cloud was carried out.

6.1 System Architecture and Design

By looking at the Figure 15, we can investigate the components of the system architecture and how they are connected. There are four modules that communicate with each other through the whole test execution. As it can be seen from the figure, each module is deployed on the cloud on different communicating nodes.

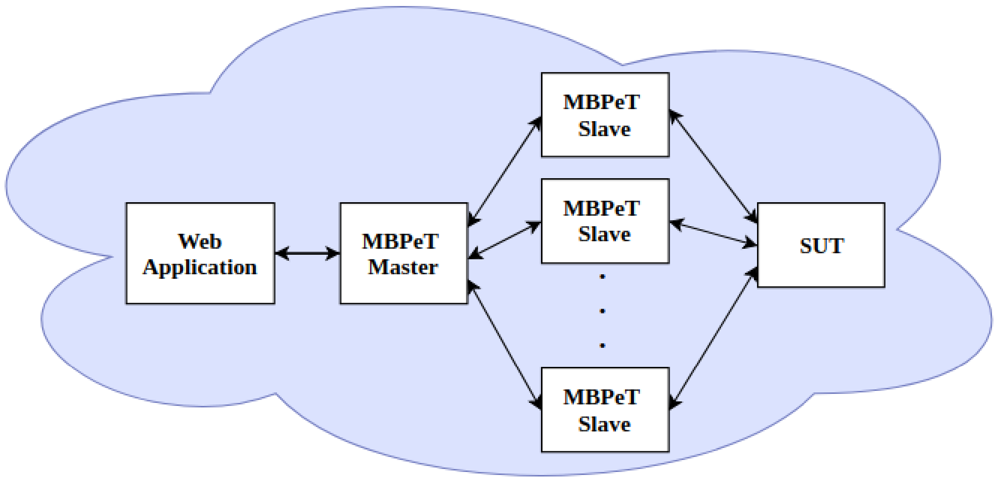


Figure 15: System Architecture

The first component is the Web Application which is the interface that links the user to the tool executing the tests. The user accesses the web application through a specific URL and inputs what is necessary for running the tests. The application resides on one of the Virtual Machines (VMs) initiated on the cloud. The same applies to the master node of the MBPeT tool. When the test is initiated, a connection between the web application and the master node is opened using SSH.

Subsequently, there is the need to create a connection between the master node and the slave ones. The MBPeT tool requires that the number of slaves is predefined by the user before starting the test execution, but it can be different for each test session. This means that their creation and deletion are achieved dynamically during the test execution thanks to the library boto3 [41]. More discussions about the library can be found later in this chapter. We will also explain how the dynamic provisioning of the slave nodes was made possible due to the library's APIs.

The last module of the system is the SUT, which contains the web service that the user is interested on testing. The SUT deployment on the cloud is performed in the same way as with the web application and the master node. As observed in the picture above, each of the mentioned modules is located on a different VM.

Since the web application was the focus through the whole implementation, we will also present separately how the communication with the user is handled. Previously, it was pointed out what the web application requires as input and what produces as output, and to have a better understanding, it can also be observed at the picture below. It displays that the input data is constituted by the user models, the test configuration and the adapters. The output is provided in two ways. The first way is in real-time while monitoring the test and the second one provides the user with the reports generated only at the end of the test execution.

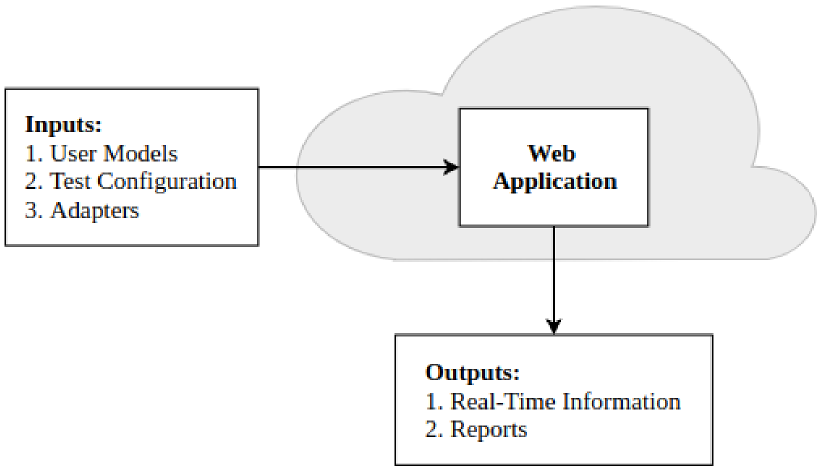


Figure 16: User Interaction with the GUI

6.2 Implementation

In this section, more details about how the implementation was handled will be introduced. We will go through all technologies that were used, which features were additionally added to the existing solution and how the chosen technologies were a great fit in accomplishing the stated requirements.

6.2.1 Technologies

As mentioned in the second chapter when explaining the web technologies, Django is used for the implementation of the web application. Since it is a web framework, it consists of different programming languages on the server and the client side.

The core technologies used for the front-end side are HTML, CSS and JavaScript. They are considered as inevitable for the implementation of every web application. In addition to the mentioned ones, AJAX is also used, mainly for handling requests to the server side and jQuery for manipulating the DOM. Bootstrap also needs to be mentioned since it allows to build a responsive user interface and lots of prebuilt styling.

Since Django is a Python web framework, it is obvious that Python is the technology used for the server-side development. The rationale behind these choices is firstly related to Python's characteristics and benefits. It is described as interactive, interpreted, modular, dynamic, object-oriented and its focus is mainly towards maintainability and code readability [42]. The extension of Python usage in web development is due to its extensive support libraries, which includes different areas. From the list of Python framework, Django was then chosen because of all its attributes described in detail in section 2.2.3.1.

Since the requirements include the need for storing all test artifacts (models, configuration parameters and adapters file), the usage of a database was necessary. SQLite was chosen over options like MySQL, Oracle, PostgreSQL and many others. The official website [43] describes it a small, fast, high-reliable and full features SQL database engine. All the mentioned databases are included with Django nowadays and there is no need to install any additional package.

Some arguments from [44] were taken under consideration while choosing SQLite. The first one is the fact of not requiring any setup, installation or configuration step to get it working. The second reason is related to being file based, in other words, it is a single disk file which can be placed anywhere in the project. This fact makes the database extremely portable and easy to be accessed, possibility of reading and writing to it, but less secure on the other hand. Another characteristic is its compactivity. Since it is a single file, after size optimization, the SQLite library can be around 500KB, which is really small compared to other databases.

Amazon Web Services [45] was chosen over other cloud computing services as the infrastructure to host all the modules. The motivation behind the decision is related to the ease of the system's deployment and the library used for the communication between the modules.

6.2.1.1 Components

Django is based on the MVC design pattern but with a twist, it is transformed into MTV, which stands for Model-Template-View. A brief explanation of each term was presented earlier. We will go through and present what files are included respectively in model, template and view.

The **model** component is composed by the models.py file. In this file, one needs to define the database of the system, what tables are found in it, the fields of each table and their types (char, integer, float, datetime etc.). It can also contain the relations between different tables through the foreign keys. The code below illustrates how the model is defined through the class named Model. The id of the table is automatically added as the primary key.

```
class Model(models.Model):
        name = models.CharField(max_length=200)
        file = models.FileField(upload_to='media/')
        uploaded_at = models.DateTimeField(auto_now_add=True)
        testsession = models.ForeignKey(TestSession, on_delete=models.CASCADE)
```

Because Django is an MVC framework, it separates the server-side logic from the client-side through a component called **template**. The HTML can be found in here. It is composed by static elements that will be rendered on the web page, as well as specific dynamic components which are injected by the server-side. Static files like JavaScript and CSS codes are easily loaded in the templates, providing them with styles and functionalities.

The last component of MTV is obviously the **view.** It is the other component where the server-side logic is placed, in this case the Python code. In the view, one can find the logic of the application. Views are composed by classes or functions which are invoked by a web request and return a response. The response varies from returning a template, an error like 404 Not Found page, a redirect or just a JSON response. When the response is an HTML template, the render function is used, which takes as parameters the template being rendered and possible arguments which are passed to it for dynamic content. An example of such response is the following:

```
return render(request, 'monitor.html', {'tests':tests, 'testID':0})
```

After specifying how the views and templates relate, it is necessary to also explain how the views can be accessed through the URLs in a web browser. This is done in the urls.py file which defines the mapping of each specified URL with a specific function/class found in the views. The navigation begins with the user who makes a request by typing an URL. This URL is checked through the tuple of URLs to see if there is a mapping with a view. One example displaying the mapping between the dashboard view when the user navigates in that panel is shown below:

```
urlpatterns = [
    path('dashboard/', Dashboard.as_view(), name="dashboard"),

    …
]
```

The connection between all the aforementioned components is shown in the picture 17. It starts with a request coming from a web browser and the workflow proceeds as explained. The picture also displays how the database is linked to the model component. More information about the database can be found in the following section.

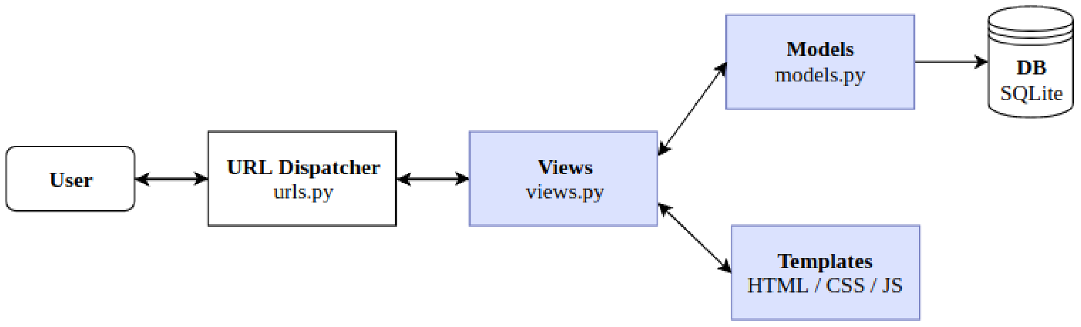


Figure 17: Web Application Components

6.2.1.2 Database

The descriptions of the used database, tables and their relationships are provided in this section. For the implementation of the project, four tables were needed:

1. *User* table which stores all registered users and their information.
2. *Testsession* which saves the created tests by the users. It has the unique field, which is the id of the created test, the name of the test defined by the user and the user_id which is the foreign key and connects to the previous table. All the users can create one or more test sessions.
3. *Model* table links to the testsession table through the test's id. Every test session can have one or more user models. The model table stores only the user model's id, name and the id of the test which is part of. The actual file containing the code is never stored in the database, but instead, in the file system. The reason behind this choice was to improve the performance and keep the database as light as possible.

4. *Configuration* is the last table which stores all the input data needed to run a test. It is specific to the test session. This implies that the configuration and the testsession tables are linked through the test's id. As it can in the figure below, the configuration table has many fields. These fields match with the parameters required in the settings file. The connection of the tables can be viewed in the picture below.

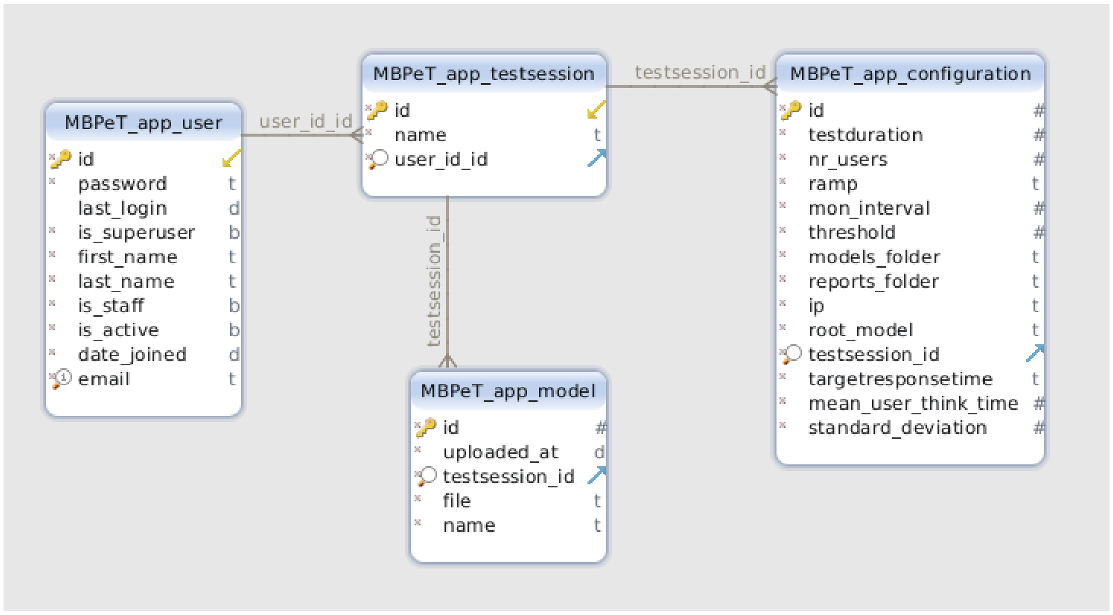


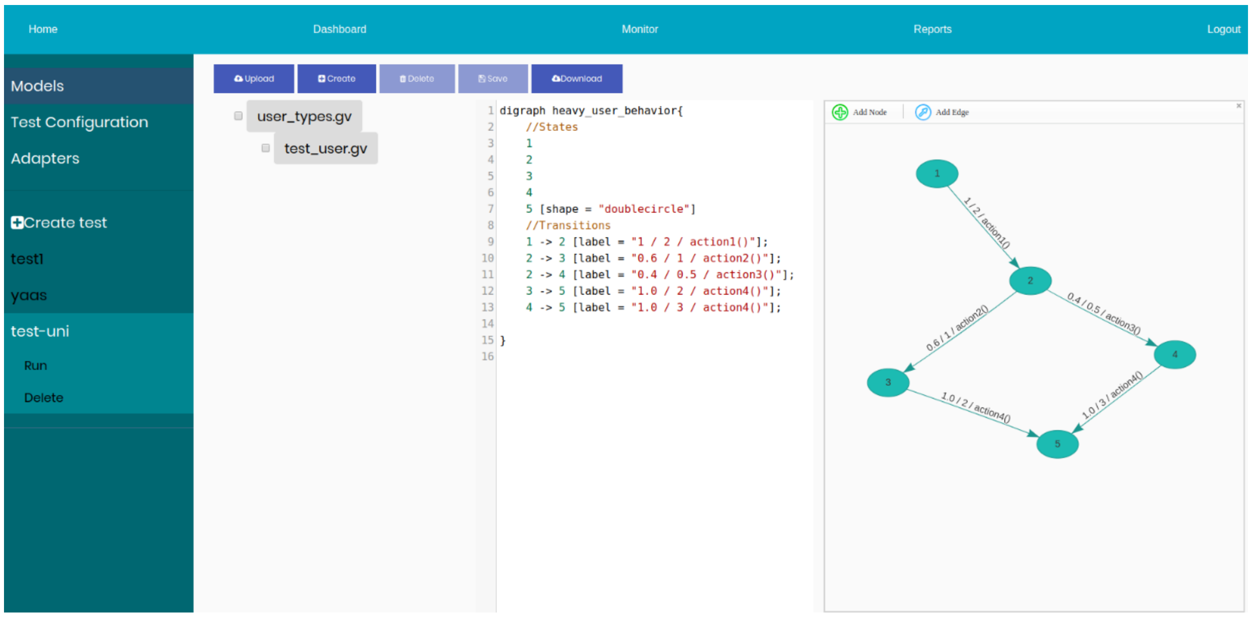
Figure 18: System's Database

In order to create the database table defined in the models.py file, it is necessary to run this Django command, *python manage.py migrate.* After running it, a series of tables are created, not only those defined in the models.py file. These tables are used by the web application to manage users, groups, sessions etc., but they are out of our interest. This is the reason why they are not displayed in the picture above.

6.2.2 Panels in the Web Application

The web application can be divided into major panels and subsections inside them, which are connected to the order of the steps taken to execute the tests. It was also considered the necessity of fulfilling the requirements and providing a logical progress of the information to the users. The main sections are Dashboard, Monitor and Report.

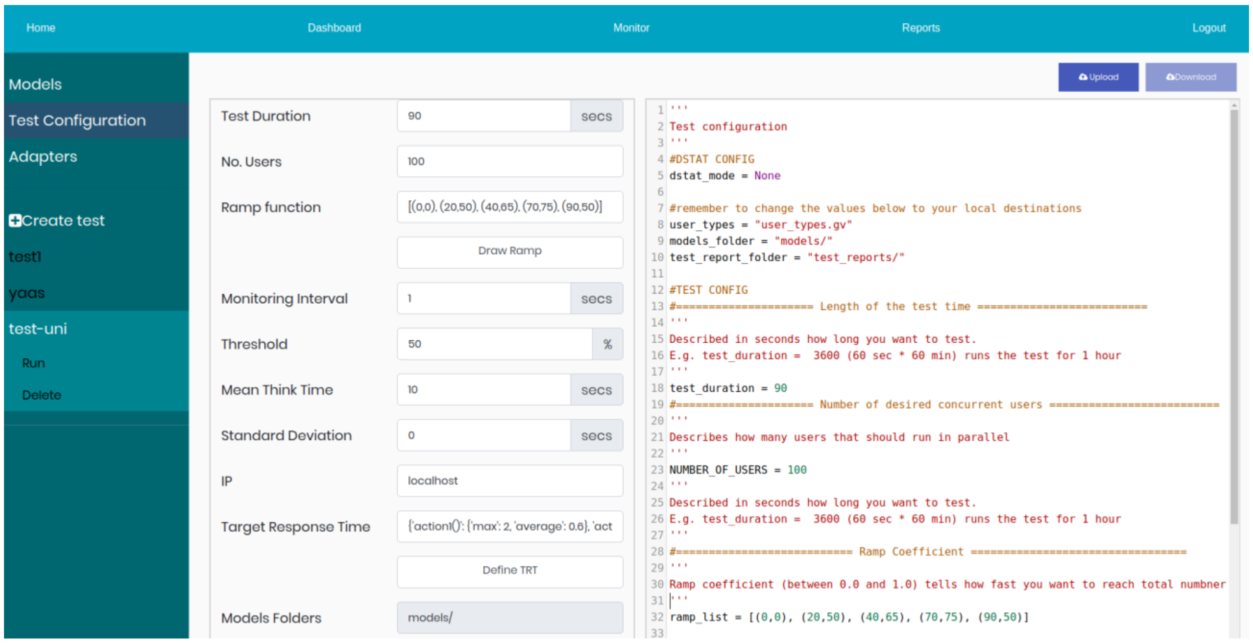
In the dashboard panel, the user is required to input / upload all information needed for running the tests. It is itself divided into three subcategories that we already mentioned and explained when showing the workflow diagrams: models, test configuration and adapters. Pictures of the achieved implementation in the dashboard panel are shown below. We can also look at the actions that were additionally implemented, as: upload, create, delete, save and download.



```
digraph heavy_user_behavior{
    //States
    1
    2
    3
    4
    5 [shape = "doublecircle"]
    //Transitions
    1 -> 2 [label = "1 / 2 / action1()"];
    2 -> 3 [label = "0.6 / 1 / action2()"];
    2 -> 4 [label = "0.4 / 0.5 / action3()"];
    3 -> 5 [label = "1.0 / 2 / action4()"];
    4 -> 5 [label = "1.0 / 3 / action4()"];

}
```

Figure 19: Models Section



Test Duration	90 secs
No. Users	100
Ramp function	[(0,0), (20,50), (40,65), (70,75), (90,50)]
Monitoring Interval	1 secs
Threshold	50 %
Mean Think Time	10 secs
Standard Deviation	0 secs
IP	localhost
Target Response Time	{'action1()': {'max': 2, 'average': 0.6}, 'act
Models Folders	models/

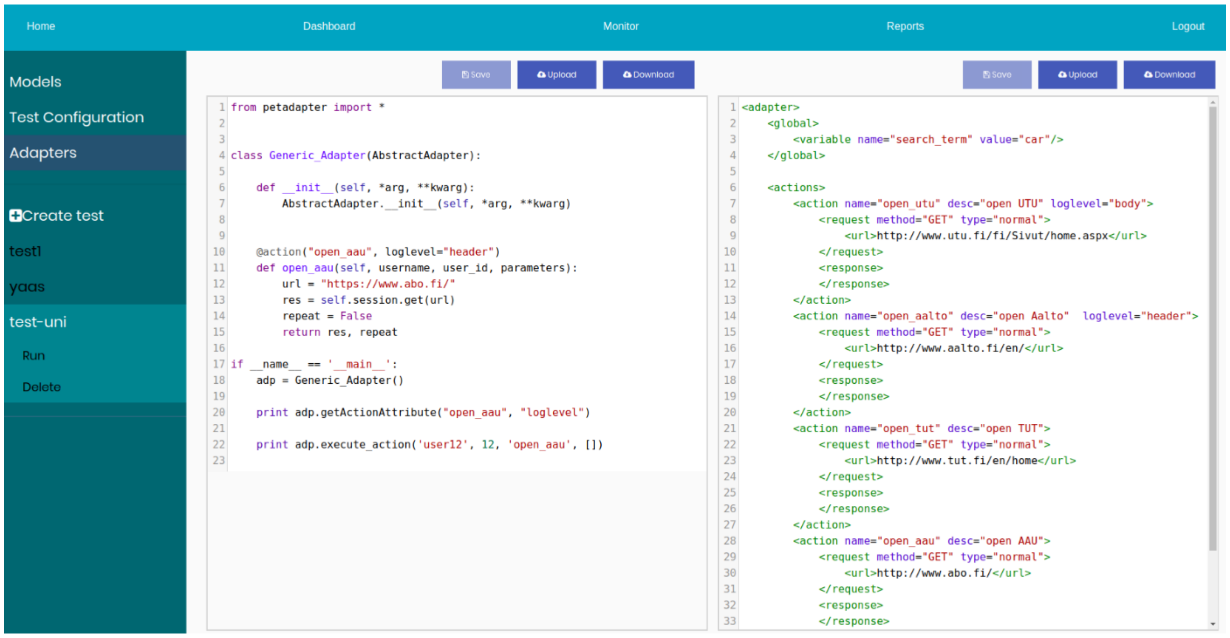
```
'''
Test configuration
'''
#DSTAT CONFIG
dstat_mode = None

#remember to change the values below to your local destinations
user_types = "user_types.gv"
models_folder = "models/"
test_report_folder = "test_reports/"

#TEST CONFIG
#==================== Length of the test time =========================
'''
Described in seconds how long you want to test.
E.g. test_duration =  3600 (60 sec * 60 min) runs the test for 1 hour
'''
test_duration = 90
#==================== Number of desired concurrent users =========================
'''
Describes how many users that should run in parallel
'''
NUMBER_OF_USERS = 100
'''
Described in seconds how long you want to test.
E.g. test_duration =  3600 (60 sec * 60 min) runs the test for 1 hour
'''
#========================== Ramp Coefficient =================================
'''
Ramp coefficient (between 0.0 and 1.0) tells how fast you want to reach total numbner
'''
ramp_list = [(0,0), (20,50), (40,65), (70,75), (90,50)]

```

Figure 20: Test Configuration Section



```python
from petadapter import *


class Generic_Adapter(AbstractAdapter):

    def __init__(self, *arg, **kwarg):
        AbstractAdapter.__init__(self, *arg, **kwarg)


    @action("open_aau", loglevel="header")
    def open_aau(self, username, user_id, parameters):
        url = "https://www.abo.fi/"
        res = self.session.get(url)
        repeat = False
        return res, repeat

if __name__ == '__main__':
    adp = Generic_Adapter()

    print adp.getActionAttribute("open_aau", "loglevel")

    print adp.execute_action('user12', 12, 'open_aau', [])

```

```xml
<adapter>
    <global>
        <variable name="search_term" value="car"/>
    </global>

    <actions>
        <action name="open_utu" desc="open UTU" loglevel="body">
            <request method="GET" type="normal">
                <url>http://www.utu.fi/fi/Sivut/home.aspx</url>
            </request>
            <response>
            </response>
        </action>
        <action name="open_aalto" desc="open Aalto"  loglevel="header">
            <request method="GET" type="normal">
                <url>http://www.aalto.fi/en/</url>
            </request>
            <response>
            </response>
        </action>
        <action name="open_tut" desc="open TUT">
            <request method="GET" type="normal">
                <url>http://www.tut.fi/en/home</url>
            </request>
            <response>
            </response>
        </action>
        <action name="open_aau" desc="open AAU">
            <request method="GET" type="normal">
                <url>http://www.abo.fi/</url>
            </request>
            <response>
            </response>
```

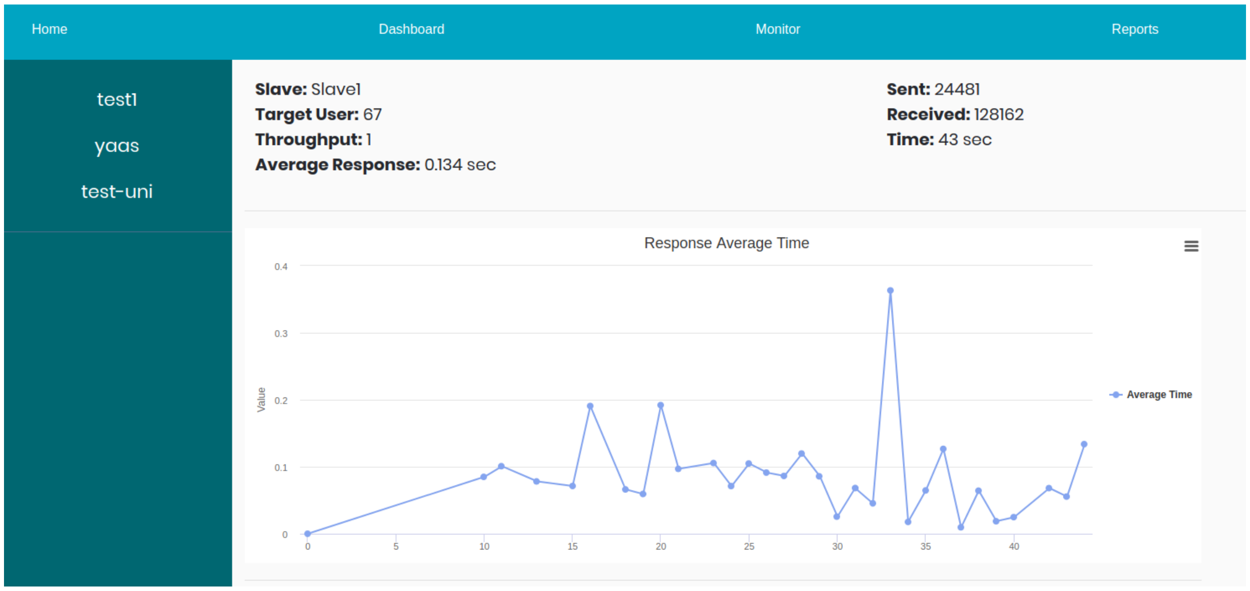
Figure 21: Adapters Section

The monitor panel displays information in real-time about the various KPIs. The information is updated in periodic intervals and different charts are also shown. The information regarding the KPIs are made available from the MBPeT tool in JSON format. It is necessary to parse and display the correct data in the monitoring panel. In our case, JavaScript's method, JSON.parse() was used in achieving the JSON parsing. An example of the data strusture in JSON format can be seen below:

```
{
   Slave1: {
         values: {
              resp: [
                 {
                      action: "action1",
                      tag: "resp",
                      val: 0.099612,
                      message_time: "2019-04-07 23:01:22",
                 },
                 {…},
                 {…},
              ],
              net_recv: [
                 {
                       tag: "net_recv",
                       val: 371028,
                       message_time: "2019-04-07 23:01:22",
                 }
              ],
            net_send: [
                {
                    tag: "net_send",
                    val: 39926,
                    message_time: "2019-04-07 23:01:22",
                }
            ],
            error: [],
        },
        summary:
            net_send_total: 39926,
            throughput: 6,
            net_recv_total: 371028,
            resp_avg: 0.113868
        },
        target_user: 20,
        slave_time: "2019-04-07 23:01:22",
        slave_name: "Slave1"
    }
}
```

All presented data are related to the current active slave. As it can be seen in the above example, each individual action is monitored and the response time during the test execution is shown. The net_recv and net_send sections give information about the SUT network utilization, demonstrating how many bytes are being transmitted during that moment. In the summary part, all previous information is aggregated and average/total values of the KPIs are given.

The picture below gives an overview of the monitor panel during one test execution. It displays the basic information from the JSON response and different graphs as well.

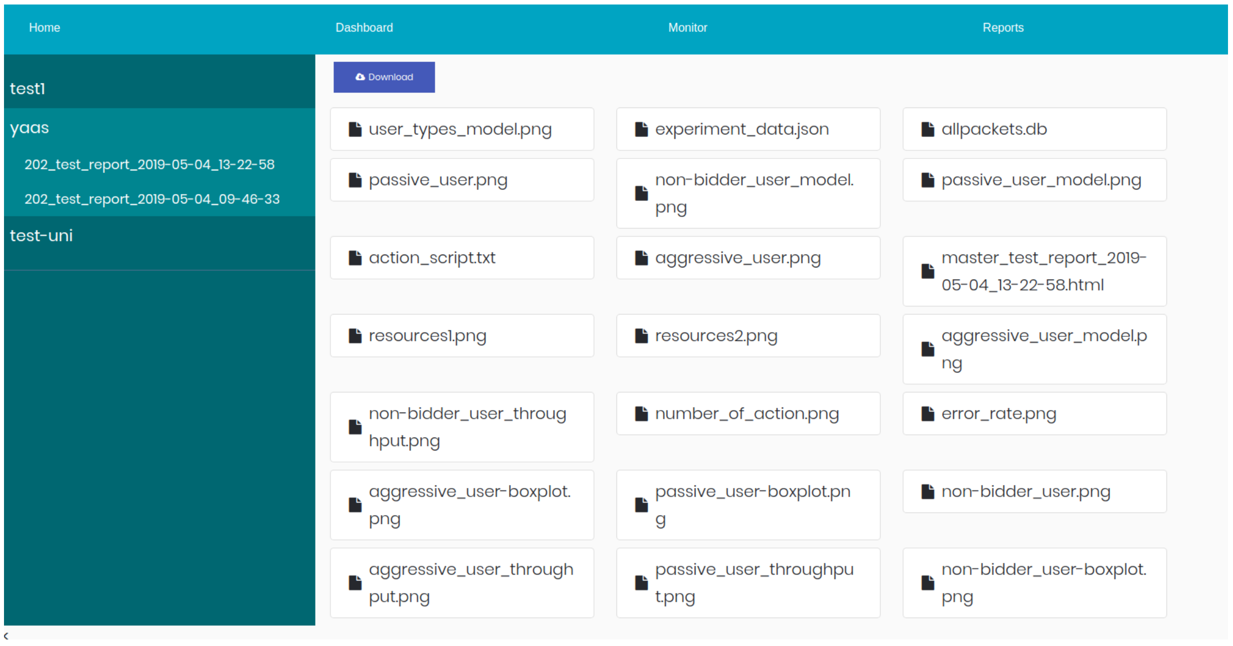


Slave: Slave1
Target User: 67
Throughput: 1
Average Response: 0.134 sec
Sent: 24481
Received: 128162
Time: 43 sec

Figure 22: Monitor Panel

In the report section, the reports generated by the MBPeT tool are provided to the user. The graphs which were already shown in the monitoring section can also be found here. Other type of information can also be found in this section, like tables and graphs which show additional data. As an example, a table is present in this report which displays if the tests passed or failed based on the average/max response time threshold breach for each studied action. There are other files which display information about the system stats, SUT network and disk utilization, physical memory usage or CPU usage from each slave node.

The user is provided with the possibility of downloading the whole test report. The reports are stored until the user deletes them, which consists of another feature that was implemented. The user can view the list of all files present in one test report directory and open them in the browser. The following picture showcases how the report section looks like in the web application, while the next one shows the case when the user clicks on one of the files.



test1
yaas
202_test_report_2019-05-04_13-22-58
202_test_report_2019-05-04_09-46-33
test-uni

user_types_model.png	experiment_data.json	allpackets.db
passive_user.png	non-bidder_user_model.png	passive_user_model.png
action_script.txt	aggressive_user.png	master_test_report_2019-05-04_13-22-58.html
resources1.png	resources2.png	aggressive_user_model.png
non-bidder_user_throughput.png	number_of_action.png	error_rate.png
aggressive_user-boxplot.png	passive_user-boxplot.png	non-bidder_user.png
aggressive_user_throughput.png	passive_user_throughput.png	non-bidder_user-boxplot.png

Figure 23: Reports Panel

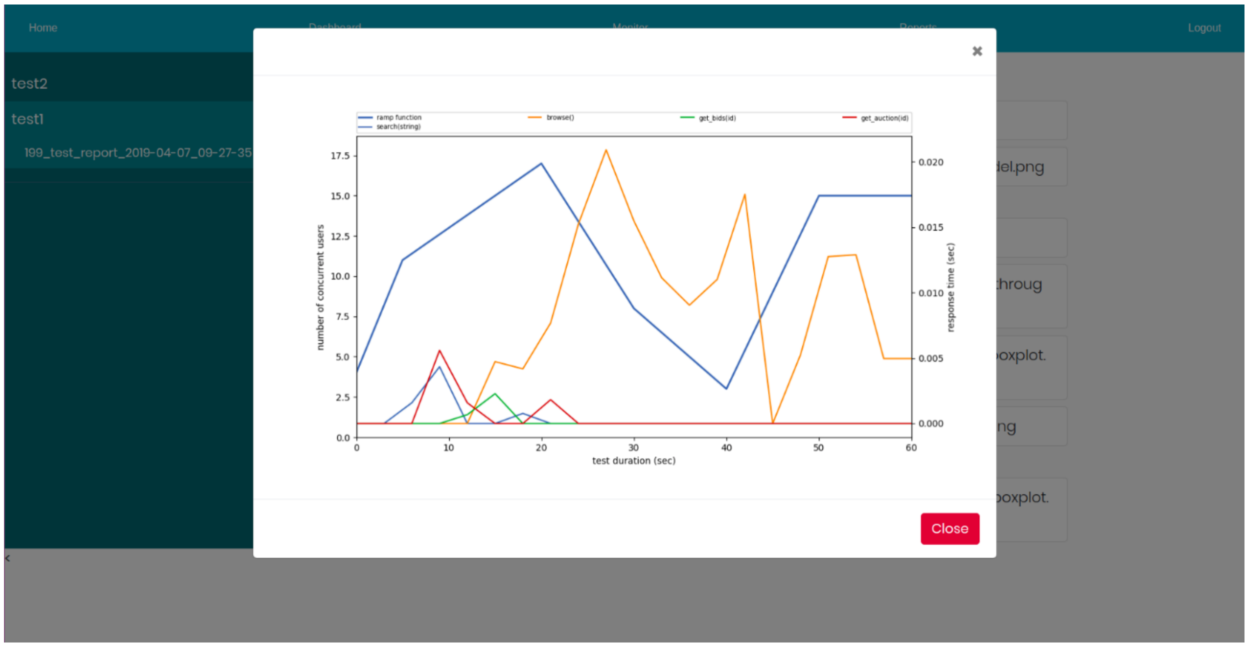


Figure 24: Example of a Report File

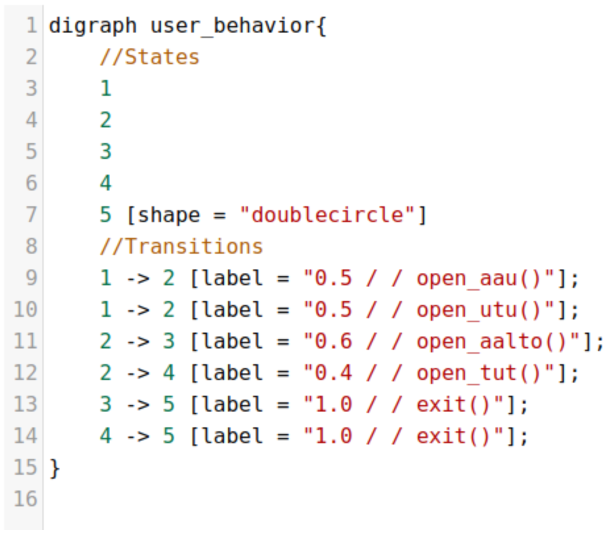
6.2.3 Additional Features and JS Libraries

In this section, we will introduce the additional features which enabled the software functionalities' enhancement. The main reason for developing them is related to the usability requirement, to take into account the user experience and make it as smooth as possible. Together with the features' description, information about the used libraries to implement them is provided. All the following libraries are implemented in JavaScript, so a further explanation on how JS contributes to a better implementation will follow. These components can be found in specific panels:

Online Code Editor

The online code editor is the feature that provides additional functionality, empowering the user to upload files from a remote computer and if there are changes required, the modification can be directly done in the system using the editor. We started from this component as it is used in all panels. It is firstly used in Models to define the user models expressed in DOT language. Then it is also used in the Test Configuration by providing the input parameters for the test execution using a python file "settings.py". The last application is in the Adapters, where the user has to define the adapters using XML and/or python language.

The picture below displays how the online editor for defining the user models looks like. After making the proper changes to the code, the DOT script will be translated into a drawn / graphical model thanks to another JS library.

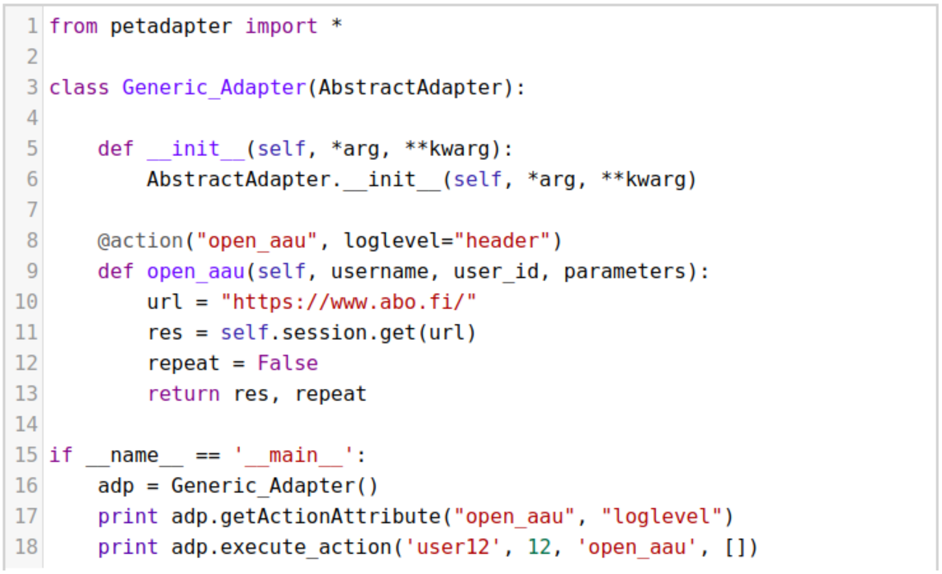


```
digraph user_behavior{
    //States
    1
    2
    3
    4
    5 [shape = "doublecircle"]
    //Transitions
    1 -> 2 [label = "0.5 / / open_aau()"];
    1 -> 2 [label = "0.5 / / open_utu()"];
    2 -> 3 [label = "0.6 / / open_aalto()"];
    2 -> 4 [label = "0.4 / / open_tut()"];
    3 -> 5 [label = "1.0 / / exit()"];
    4 -> 5 [label = "1.0 / / exit()"];
}
```

Figure 25: User Models Online Editor

The library that was used for the implementation is **CodeMirror** whose main function, as described in its official documentation [46], is "a versatile text editor implemented in JavaScript for the browser". It was chosen not only because it supports a large number of programming languages (more than 100) among which the ones needed for this system, but also provides additional features like a real code editor. Some of these features are: autocompletion, code folding, Vim / Emacs / Sublime Text bindings, custom scrollbars, bracket and tag matching, configurable keybindings, programmable gutters, indentation, line numbers and many other methods. [46] The documentation is also an advantage as it provides clear information on how to integrate the editor in the project and work with it.

Another example of the editor's usage is showing in the Figure 26. It is the case of an adapter written in Python language, where one action named "open_aau" is translated into a real operation: making a request to the website "https://www.abo.fi/".



```python
from petadapter import *

class Generic_Adapter(AbstractAdapter):

    def __init__(self, *arg, **kwarg):
        AbstractAdapter.__init__(self, *arg, **kwarg)

    @action("open_aau", loglevel="header")
    def open_aau(self, username, user_id, parameters):
        url = "https://www.abo.fi/"
        res = self.session.get(url)
        repeat = False
        return res, repeat

if __name__ == '__main__':
    adp = Generic_Adapter()
    print adp.getActionAttribute("open_aau", "loglevel")
    print adp.execute_action('user12', 12, 'open_aau', [])
```

Figure 26: Adapters Online Editor

Interactive user models drawing

This feature is related to the possibility of drawing interactively the user PTA models. As already known, the models consist of nodes, edges and labels that can be specified in the code using the DOT language. This component allows users to draw the models by adding, editing and removing nodes, edges and labels. The harmonization between the code in the online editor and the drawn graph is taken into consideration.

A JavaScript library named **vis.js** [47] was used for the implementation. It contains different components like: Network, DataSet, Timeline, Graph3d and Graph2d meaning that it includes many functionalities. Our interest is towards the first component, Network, which is a visualization library for displaying and modifying networks composed by nodes and edges. [48]

Integrating the JS libraries in a web application is really easy. It is only required to include the link in the HTML file, as below:

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vis/4.20.1/vis.min.js"></script>

The library offers the advantage of handling model manipulations in various ways. Not only the basic operations like the ones mentioned above, but there are also options for the styling of the models. Since many were out of the project scope, the focus was kept on the main features. The following picture displays the result we took after using the library with one of the defined user models.

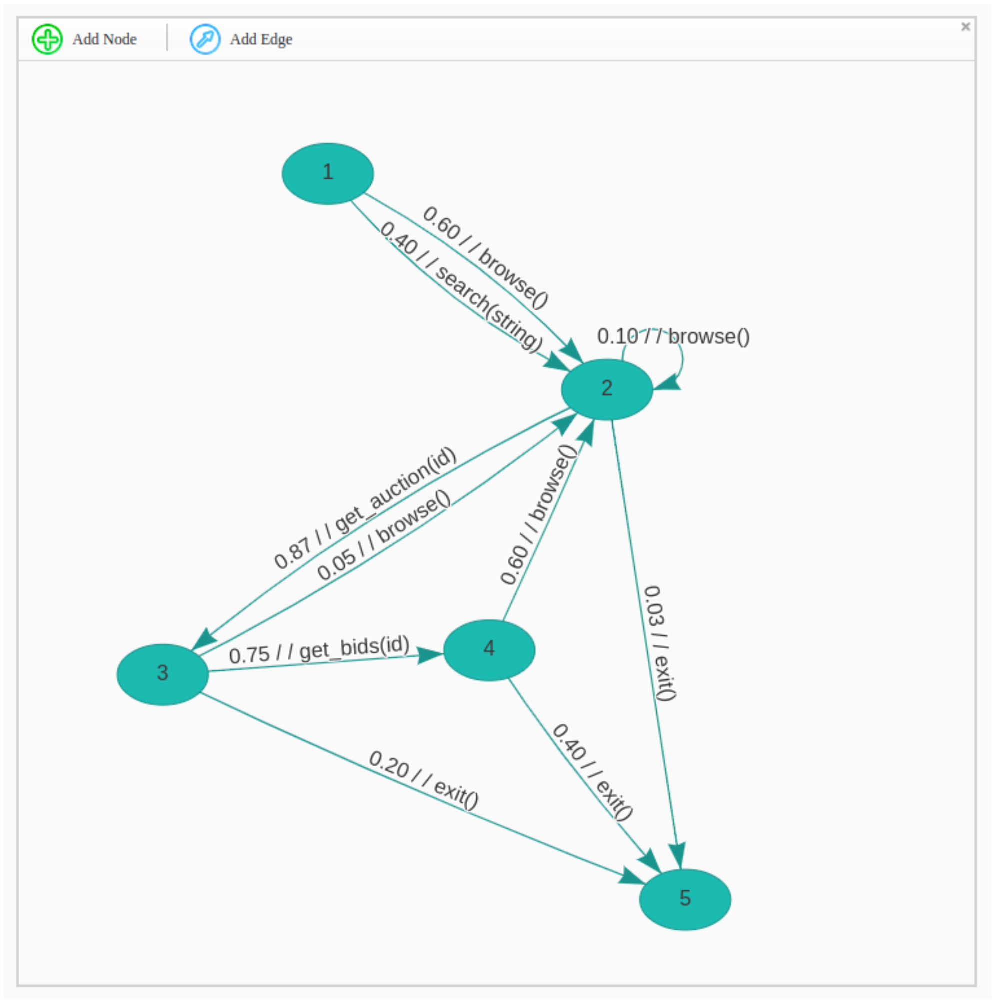


Figure 27: User Model Example

It is important to mention another big advantage of the library; the fact that it supports data in the DOT language. With the method *vis.network.convertDot,* the model represented using the DOT language, can be easily converted into its graphical version with nodes, edges and labels. In this way, the harmonization between the code and the drawn model is easily achieved.

Interactive charts

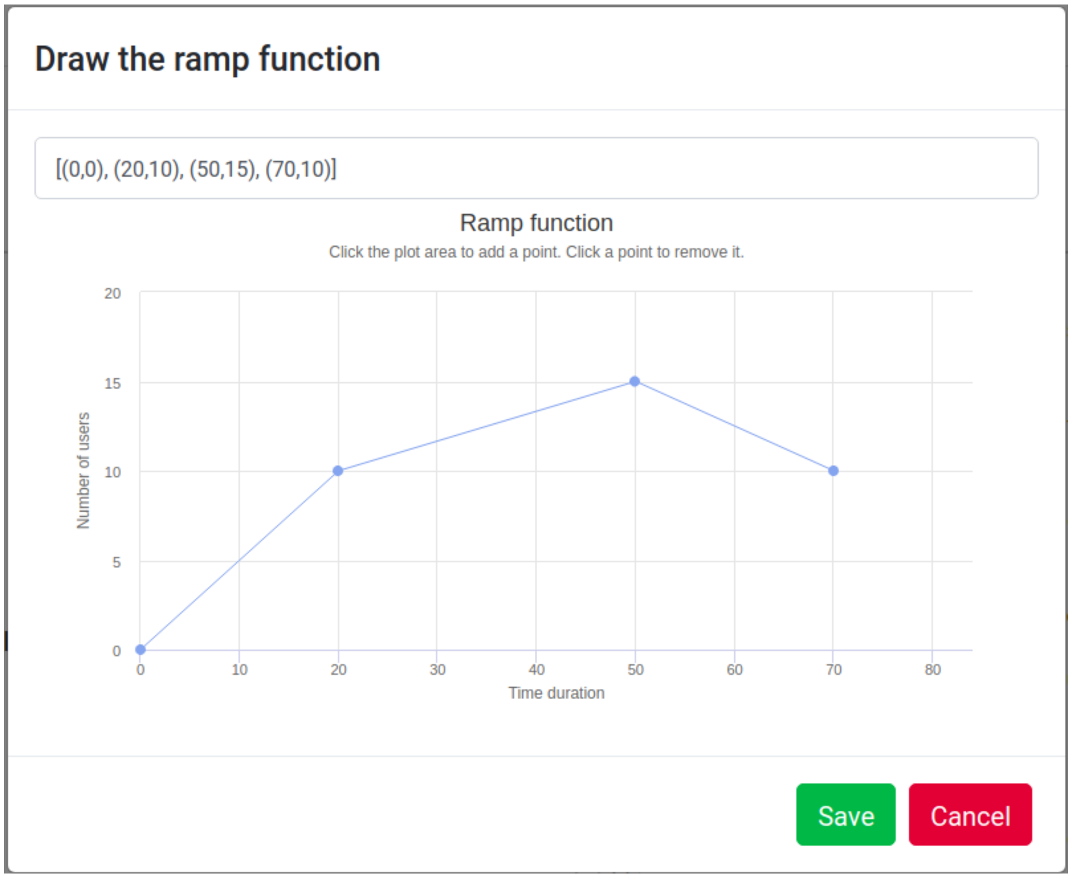
This feature allows charts to be interactive and it is applied to all of them. There are two main usages of this component which will be presented below.

The first chart can be seen when specifying the ramp function in the test configuration step. The ramp function displays how many users will be achieved during the test duration. It can be defined using an array of tuples (pairs):

[(0,0), (20,10), (50,15), (70,10)]

where each pair is composed by two numbers. The first one defines the test duration and the second shows the target number of users. For example (50,15) indicates that, during the test execution, at 50 seconds the number of concurrent users will be 15. This array of tuples will then be converted to a line chart and the opposite. The user can visualize the ramp given as an array, but he/she can also define it by adding or removing points directly in the chart. This composes one of the improvements which makes it more user-friendly and easier to be used.

The library used for this functionality is called **HighCharts**, a JavaScript library which provides a variety of interactive dynamic charts and operations over the charts [49]. The library provides the flexibility of interacting with the JavaScript clicking event through one of its parameters. This is used to handle the adding and removing points by detecting the coordinates of the selected point. If the clicked point is not in the sequence, then it is added and the other way around. The update of the textual ramp function is done at the same time. Picture 28 shows how this implementation looks like.



[(0,0), (20,10), (50,15), (70,10)]

Figure 28: Graphical Ramp Function

The other functionality is related to the conversion of the textual ramp to the graphical one which is done by formatting and manipulating string. Sanity checks are added to confirm that the syntax and order time are correct.

The second occasion when the library is used to implement interactive charts is in the monitoring phase. As seen in the previous section, different charts are displayed in the monitoring panel. They show information about KPIs that are being observed during the test execution. Depending on the KPIs and the actions defined by the user in the test configuration step, various number of graphs are presented. JavaScript eases the manipulation of dynamic graphs.

Since monitoring should be in real-time, the graphs need to be updated in periodic times by receiving the latest values. This is achieved via the aforementioned library, HighCharts, which facilitates the insertion of new coordinates in the chart(s) at every specified interval. The library provides two methods "addSeries" and "addPoint", which are used correspondingly for creating a new series (a line chart) and adding data to a specific series. The picture X showed how this implementation was achieved in the monitoring panel.

Target Response Time

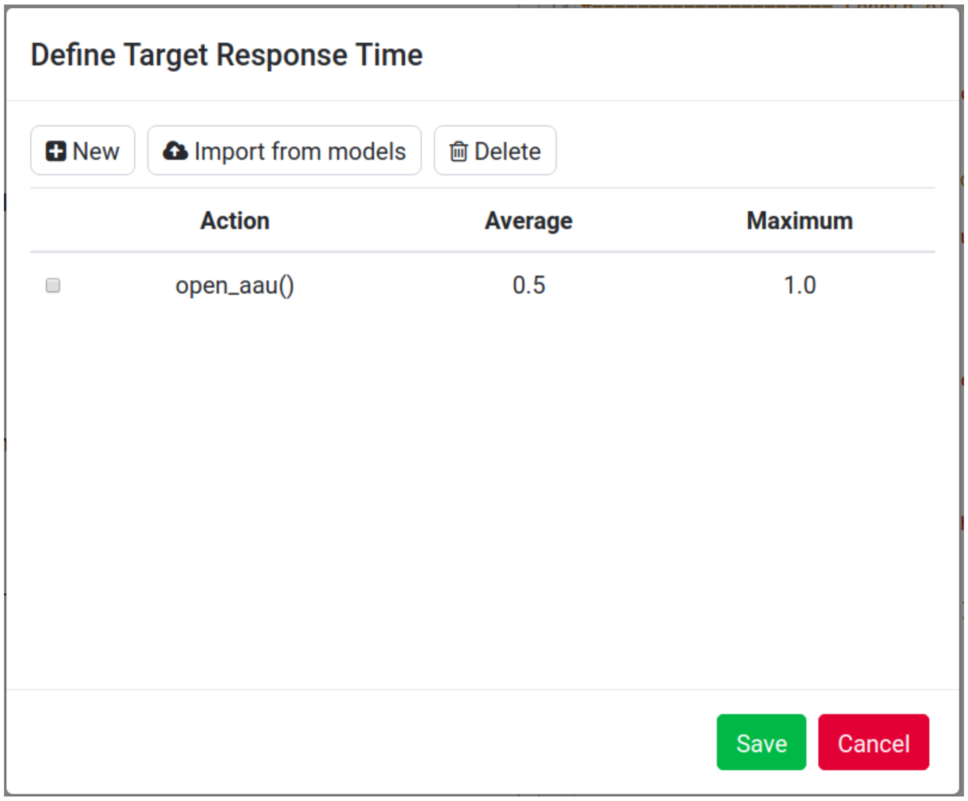
The following feature does not include any special JavaScript library, but it is important to be mentioned, since it specifies the SUT's actions needed for the final report of this study. Firstly, a brief explanation of the Target Response Time will be introduced.

It defines the target response time for the actions by specifying the max and average response time we expect when the test is executed. At the end, the target value that was previously specified will be compared with the real response time. In case it exceeds the real response time, the moment of the excess is registered, and the test is considered as failing. At the same time, the number of concurrent users that were active during that time will be registered and shown in the report.

The parameter is represented as in the following format:

```
{
        'action1()': {'max': 1.0, 'average': 0.5},
        'action2()': {'max': 0.7, 'average': 0.4},
        ….
}
```

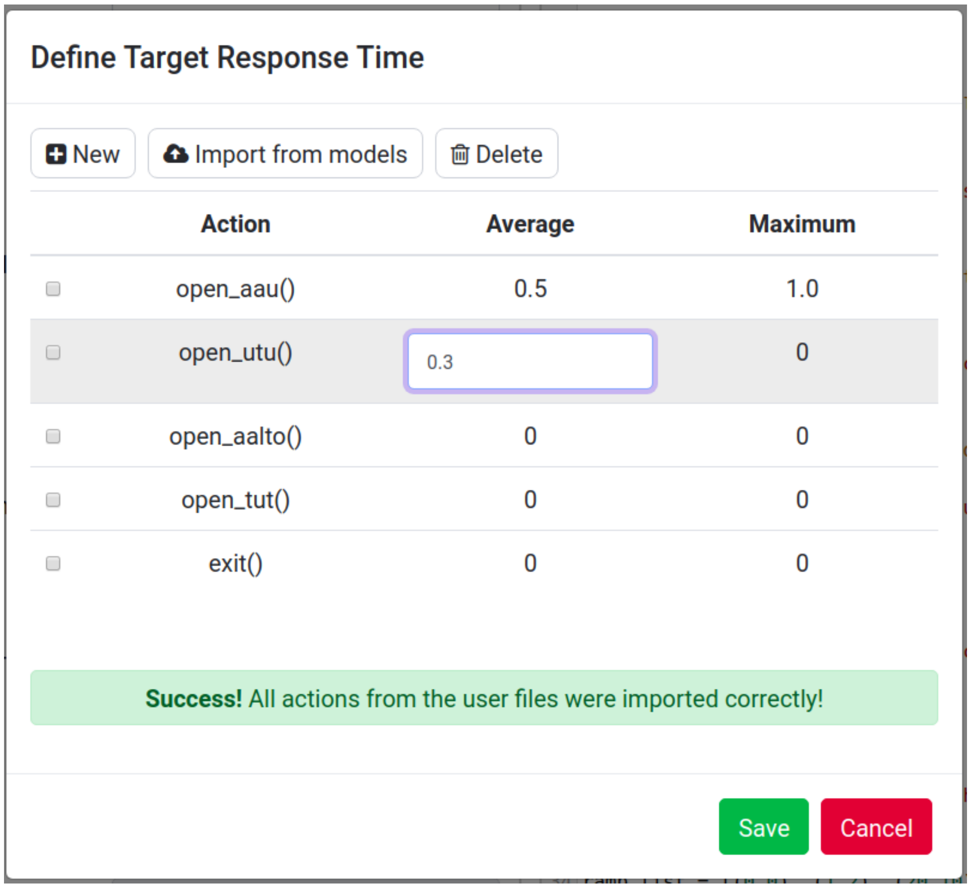
Referring to the above example, the average target time that is desired to be achieved for the action1 is 0.5 and the maximum allowed is 1.0. After the operation of string manipulation is completed to the textual target response time, it is now possible to convert it, split into the action name, average and maximum value and display as in the picture below.


Define Target Response Time

	Action	Average	Maximum
☐	open_aau()	0.5	1.0

Figure 29: Table of Target Response Time

There are three different actions (create, import and delete) one can perform to the target response time using the graphical option. The most challenging operation is the second one, importing actions defined in the user models. The clicking event from JavaScript is linked with the AJAX call to the server to retrieve the models of the test, which are stored in the database. Possible errors are always handled and displayed to the user. After retrieving the correct data from the stored files and removing possible duplicates in the actions list, the table columns are filled via jQuery by placing the right information in the proper column. The JS events doubleclick and blur allow the users to change the values as they need. It is shown in the picture X, where the average of action open_utu() is being modified. The update between the table and the textual option happens when clicking the button Save and where all table's fields are converted into the desired format.


Define Target Response Time

	Action	Average	Maximum
☐	open_aau()	0.5	1.0
☐	open_utu()	0.3	0
☐	open_aalto()	0	0
☐	open_tut()	0	0
☐	exit()	0	0

Success! All actions from the user files were imported correctly!

Figure 30: Operations on Target Response Time

6.2.4 Deployment in the Cloud

The steps that were carried out in order to finalize the deployment in the cloud will be explained in this section. Amazon Web Services (AWS) [29] was chosen over other cloud services providers. Among all cloud services that are provided by AWS, the one of our interest is named Amazon Elastic Compute Cloud (Amazon EC2). The definition taken from its official website [50] explains clearly what solutions are provided by Amazon EC2.

> "Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud."

What this system should offer is related to the resizable and scalable compute capacities that Amazon EC2 makes available. In other words, these compute resources are referred as Virtual Machines, and in the case of AWS they are named instances. Virtual machines imitate and have the same functionality as a physical hardware. Using Amazon EC2, several virtual machines can be initiated which only differ from each other by their configurations.

The first and the most important step when launching a VM is to choose the right Amazon Machine Image (AMI). Each virtual machine will be configured with a specific operating system (OS), and this is defined through the AMI. The AMI includes not only the OS one can use for the VM, but also other pre-deployed applications. For example, the AMI used in this project is *Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type*. [51]

The user can also create its own AMI from the ones offered by AWS. The purpose of creating a custom AMI is to ease the process of deploying many times the same application / service when launching a new instance. It is the case of the MBPeT slave nodes that need to be initiated on every test execution. So, in order to avoid deploying the MBPeT slave program on every VM that is launched, an AMI with pre-installed dependencies is created. In this way, all VMs that are created from this specific AMI, will already have the MBPeT slave software deployed in them. This applies and is used also for the MBPeT master software.

Then, the other configuration parameters are out of the scope of the project and most of them have already their default values. The procedure of deploying the system on the cloud is as below:

1. Create an instance and deploy the GUI in it.
2. Create an AMI with the MBPeT master deployed on it.
3. Create an AMI with the MBPeT slave deployed on it.
4. Launch an instance from the first AMI which will function as the master node.
5. Launch instances from the other AMI, representing the slave nodes.

The user will be provided with the URL of the GUI which resides on a launched VM. When a test is run, instances containing the MBPeT master and MBPeT slaves will be launched at the same time. Since this happens dynamically from the test execution and not manually using the AWS dashboard, there is the need to use a Python library which facilitates the interaction with the service provider. The library has to offer API endpoints for managing the resources needed. There are many options available, from which Apache Libcloud can be mentioned. It is not unique to a specific cloud provider and supplies a unified API for most of cloud providers [52]. Due to the lack of documentation and missing functionalities, the official library from AWS was chosen instead, named boto3. With few lines code, one can initiate, monitor, delete and manage AWS services, such as the instances in Amazon EC2. [41]

The code snippet below demonstrates an example of creating new instances and which parameters have been defined. The list of parameters that can be specified in the *create_instances()* method is longer than the one shown in the example. The number of nodes is limited to the mandatory ones, since they are sufficient for fulfilling the project's requirements. Explanations of the parameters can be found after the example.

```
ec2 = boto3.resource('ec2')
ec2.create_instances(
  ImageId='ami-0438fc535358deba1',
  MinCount=1,
  MaxCount=1,
  InstanceType='t2.micro',
  KeyName='JoanaAWS',
)
```

The first parameter is ImageId, which is the AMI's ID we have created with the MBPeT master already deployed in. MaxCount determines the number of instances to be launched. In this example, the number of instances is 1, representing the master node. In order to launch a new instance, one needs to define its type. Amazon EC2 offers a wide selection and each type offers a unique combination of CPU, storage, memory etc. The instance of type t2.micro is chosen over other options, which is one of the smallest one, but more than enough for the system. The last parameter is KeyName which is the public key that the user creates. It will be used as a way of authentication when accessing the launched resources.

The next step before starting the test execution is to retrieve the required information about the instances. With the method *describe_instances(),* one can identify if instances are already in running state, since it usually takes some time until they are ready. Once they are initiated correctly, the data which will allow the communication between the instances should be retrieved. Each instance is provided with a public DNS, a private DNS, a public IP and a private IP. In order to make the master and slave nodes run inside corresponding instances, their public DNS-s are needed to access them using SSH. There is no need for more information to execute the master program, while for executing the slaves, the private IP of the master node is also needed. It is necessary as a specification of the MBPeT tool. In this way, the master node recognizes that the slaves are ready and listening.

The following commands are examples used to run respectively the master and the slave nodes:

1. *"ssh –o -i 'JoanaAWS.pem' ubuntu@ec2-52-90-90-15.compute-1.amazonaws.com 'source myenv/bin/activate; cd mbpet_master; python run.py ../yaas-test 3 -s' "*
2. *"ssh –o -i 'JoanaAWS.pem' ubuntu@ec2-34-16-66-89.compute-1.amazonaws.com 'source myenv/bin/activate; cd mbpet-slave; python run.py 172.23.34.45' "*

After the test starts running, the monitoring in real-time can be visualized in the monitor panel of the GUI. A request is made to the appropriate URL, which provides with the necessary information. The URL is based on the public DNS of the master node or its public IP. Both solutions can be used and work the same way. An example of this URL is: *ec2-52-90-90-15.compute-1.amazonaws.com:8888/update.*

When the test execution is ended, all instances should be terminated. This is achieved by another function provided from the boto3 library, *terminate_instances().* It takes an argument which is an array containing the instance ids, we want to terminate.

7. Evaluation

After the implementation came to an end, it was fundamental to evaluate if all stated functionalities were implemented correctly. The purpose of this chapter is to describe the case study that was used for the system's evaluation. The test setup and the various experiments that were undertaken will be described. Finally, comparison between the results and discussions about the findings can be found at the end of the chapter.

7.1 Case Study

YAAS (yet another auction site) is the web service selected to perform as the SUT of the system. The implementation was achieved using the Django web framework. Its main functionality is to provide a platform where registered users can buy and sell different products. It also allows non-registered users to check auctions, what bids have been placed on them and what their current price is.

Despite the existing web application, it also provides a RESTful web service which will be the focus of the evaluation. The APIs are mainly related to browsing the list of auctions, bids made on a specific auction, retrieving the information of an auction etc. The complete list of APIs, including description of each operation, is presented below.

1. *browse()* - Returns the list of all auctions.
2. *search(name)* - Returns the list of auctions matching the defined name as a parameter.
3. *get_auction(id)* - Returns the auction with the given id.
4. *get_bids(id)* - Returns all bids placed to the auction with the specified id.
5. *bid(id, price, username, password)* - The user with the defined credentials bids to the auction with the indicated id.

In the test setup, explanations on how testing of the mentioned APIs was achieved can be found.

7.2 Test Setup

As explained in the previous chapter, MBPeT tool, GUI and the SUT have been deployed on Amazon Web Services, using the Amazon EC2 instances. MBPeT tool has been divided into two types of instances, one containing the master node and the other type including the slave node. GUI and YAAS has been deployed beforehand the test execution, while MBPeT tool is always deployed only during the test execution. The master and the slave nodes are ready for deployment in two different AMIs as previously described. When a test is run, the EC2 instances are launched from the corresponding AMIs, one from the master AMI and the specified number of slaves from the slave AMI.

The same VM configurations have been used for the deployment of all nodes. Below, we can check some of the relevant parameters taken from [28] that were chosen.

1. AMI - Ubuntu Server 16.04 LTS (HVM), SSD Volume Type
2. Instance Type – t2.micro which includes 1 virtual CPU, clock speed 2.5 GHz, 1 GB memory, 64-bit and 32-bit Intel Xeon Family processor. The experiments will reveal if the instance type affected in some way the requirements' accomplishment.
3. Storage – size 8 GB, volume type General Purpose SSD.
4. Security Group which is a set of firewall rules that control the instance traffic [36]. Additional rules can be added to allow the incoming or the outbound traffic from a specific instance. Besides the default ones, we added the port 22 for allowing SSH into the instance and the port 8888 since it is needed to access the real-time monitoring.

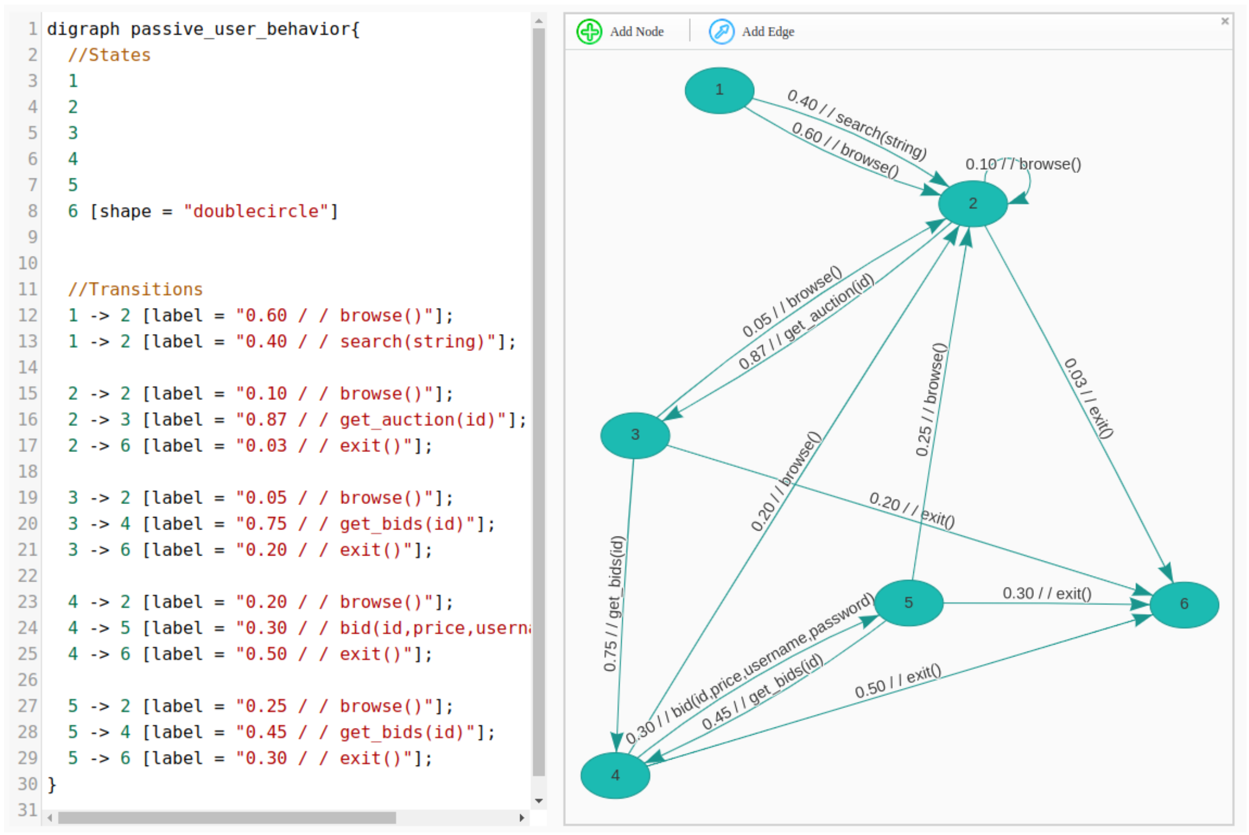
In this section, we will not go into the details of deploying the system, since the explanation has been already clarified before. Instead, the focus will be on the GUI side, when and how the tests are executed and how instances are launched.

7.2.1 GUI Setup

The user models, test configuration and adapters must be defined, in order to be able to execute the tests. At this stage, the evaluation starts from assessing the functional and non-functional requirements of the web application which have been clearly illustrated in Chapter 5. By following all the required steps, it is possible to check if it fulfills all the requirements.

User Models

It was identified from the example in the official documentation of the MBPeT tool that there are three types of users in the YAAS system. [1] These three user models are named: aggressive, passive and non-bidder user. This division has been made according to the frequency of placing bids towards the list of auctions. This can be seen in the graphs representing the models. They display which actions and with what probability each action is undertaken by each type of user. It is obvious by the naming, that the aggressive users are those who frequently bid on auctions, while the non-bidder ones are interested only on searching and browsing auctions and bids. In the figure below, it is shown the achieved result after creating the model for the passive user with the GUI. The passive type includes users bidding less frequently than the aggressive ones. When going from location 4 to 5, the action bid occurs with probability 0.3, while in the aggressive model the probability is 0.5.



```
digraph passive_user_behavior{
  //States
  1
  2
  3
  4
  5
  6 [shape = "doublecircle"]


  //Transitions
  1 -> 2 [label = "0.60 / / browse()"];
  1 -> 2 [label = "0.40 / / search(string)"];

  2 -> 2 [label = "0.10 / / browse()"];
  2 -> 3 [label = "0.87 / / get_auction(id)"];
  2 -> 6 [label = "0.03 / / exit()"];

  3 -> 2 [label = "0.05 / / browse()"];
  3 -> 4 [label = "0.75 / / get_bids(id)"];
  3 -> 6 [label = "0.20 / / exit()"];

  4 -> 2 [label = "0.20 / / browse()"];
  4 -> 5 [label = "0.30 / / bid(id,price,usern
  4 -> 6 [label = "0.50 / / exit()"];

  5 -> 2 [label = "0.25 / / browse()"];
  5 -> 4 [label = "0.45 / / get_bids(id)"];
  5 -> 6 [label = "0.30 / / exit()"];
}
```

Figure 31: Passive User Type Model

Test Configuration

The next step is related to the test configuration. As already mentioned, many of the parameters will be defined at this stage, but the focal point will be only on some specific ones. There is a need to define the IP of the SUT (YAAS software). Since the AWS Free Tier was used for testing, the instances were terminated after running the experiment. This implies that SUT's IP has changed frequently and we will show here one of the IPs used during one experiment. *3.88.84.244* was the public IP of one of the instances which included the SUT.

Another parameter that will change during the experiments is the ramp function. As we previously pointed out, one of the main goals of the tool is test how the SUT reacts to different loads of users. Numerous experiments with different ramps were run and reports on resource utilization of cloud resources can be found below. One example of a ramp function is: [(0,0), (10,50), (20,100), (50,200), (60,300)].

Threshold is another important parameter that needs to be mentioned as it defines when the slave is considered saturated. Different values will be used during the experiments to be able to evaluate the scalability of the tool when slaves are initiated in the cloud.

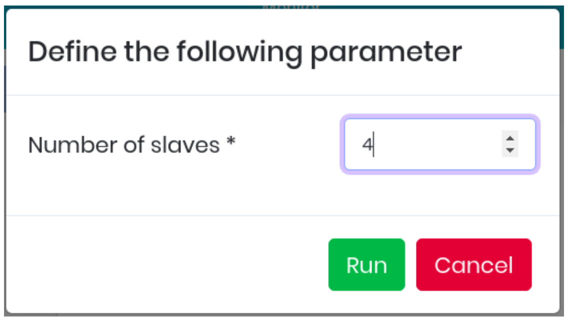
Adapters

The last step before running the test is to fill out the adapters section. We will use the adapters from the YAAS example in the official documentation. Five methods are required in order to make the interpretation of each action into the appropriate API call. The IP is taken from the settings file to construct the right URL for making the calls. The example below presents how the *browse* action is translated into the corresponding API call (HTTP request).

```
@action("browse")
def do_browse(self):
    repeat = False
    url = "http://" + self.settings.ip + '/api/v1/auctions/'
    response = requests.get(url)
    response_result = self.Check_Response(response, "browse")
    status = response_result[0]
    if str(status)[0] == '2':
        data = response.text
        try:
            aucs = json_loads(data)
            self.yaas_auctions = aucs
        except ValueError as error:
            print ("ERROR", error)
            repeat = True
    else:
        repeat = True
    return response, repeat
```

7.2.2 Test Execution

After all test inputs have been correctly defined, the next step is to start the test execution and the nodes initialization. The user is required to provide the number of slave nodes as shown in the picture below.


Define the following parameter

Number of slaves * 4

Figure 32: Selection of the Number of Slaves

When executing the tests locally, the user can run different tests at the same time. This is achieved by providing different ports for the master node process, so there are no conflicts between the processes. With the new solution, every test execution launches new instances. This implies that there is no need to provide a port for the master node process, since they run in different machines. The default port 6666 will be always used.

After the last required parameter (number of slaves) is provided, the test can finally be executed. The code snippet shown in section 6.2.4 is used to launch the master and the slave nodes at the same time. The command used to generate the load in the MBPeT master requires the path to the folder containing the information about the test. This folder resides in the file-system of the GUI node and consists of the user models, adapters and settings files. The files are created when providing the inputs using the web application. Since the MBPeT master and the web application are located in different nodes, there is the necessity of providing the folder to the MBPeT master. The current solution is to copy the folder using SCP (Secure Copy) between the nodes. SCP is the protocol which allows the transfer of files and folders between two hosts. It is related to SSH, since SSH is used for authentication between the communicating hosts. [39]

Since all commands are run from the web application, it was necessary to use a module which allows running shell commands. This was achieved by using the Python module named *subprocess,* which generates new processes, and checks their input/output/error pipes. The class subprocess.Popen(string, ...) is used to call a specific command, which is defined as the first argument. Since some commands can take some time to get executed, like the copying of the files between nodes, it is crucial to check if the process ended executing. This is achieved by using the poll() method from the subprocess.Popen class, which returns None if it is still in execution. [54]

When the initialization is done successfully, the user will be redirected to the monitor panel, where real-time information and charts are published. A folder with all test report files is created in the master node and copied back to the web application node using SCP after the test is executed. In the end, the master and slave instances are terminated and deleted.

To evaluate the fulfillment of the requirements, several experiments will be executed. The main interest is on the scalability of the provided solution and on the correct implementation of all functionalities of the web application. Experiments will use the same test case, but they will differentiate from the values given to the ramp function, the target response time and the number of slave nodes. After appraising that the system works as expected, we will report and discuss about the resource utilization of the cloud resources and other measured KPIs.

7.3 Experiment 1

In the first experiment, the focus is on evaluating that the system is fully functional. It will evaluate if all GUI's features work as expected and if the test is executing correctly by initiating the specified number of nodes.

The ramp function used in this experiment is: *[(0,0), (40,300), (100,500), (200, 500)]*

while the TRT is defined:

{'exit()': {'max': 1.0, 'average': 1.0},
'search(string)': {'max': 6.0, 'average': 3.0},
'browse()': {'max': 8.0, 'average': 4.0},
'bid(id,price,username,password)': {'max': 10.0, 'average': 5.0},
'get_auction(id)': {'max': 4.0, 'average': 2.0}}

It was also specified that the number of slave nodes used is 3 and the threshold set to 70%. Since all instances are launched at the same time, after sending the command of starting the test execution, the time to deploy and initialize a new instance is not dependent to the number of virtual machines being launched. It takes around 5-10 seconds for the instances to start. Connecting to the instances and start the execution of the test does not require more time comparing to running it locally, so performance in this case is not affected. The only execution that slightly decreased the performance is when copying the test input data from the web application to the master node. If there are many files, the latency increases significantly by affecting negatively the non-functional requirement related to the performance of the system. This implies that changes are required to avoid this issue and ways of overcoming it will be discussed in section 7.5.

The instance of type t2.micro was used in this experiment. It provides only one virtual CPU for the launched instance. After running the test, the report provides different types of information. Our interest is mainly on the resource utilization of the cloud resources and how they react to different ramps. One of the sections contains information about the stress level on the slave nodes.

```
Total Disk read bytes: 196.0KiB
Average Disk read bytes: 942.3B/s
Total Disk write bytes: 424.0KiB
Average Disk write bytes: 2.0KiB/s
Total Network sent bytes: 27.7MiB
Average Network sent bytes: 133.3KiB/s
Total Network received bytes: 17.6MiB
Average Network received bytes: 84.4KiB/s
Physical Memory Usage: 33.45 %
Slave1 CPU 0 Usage: 38.03 %
Slave2 CPU 0 Usage: 6.33 %
```

We can check that only 2 slave nodes were used for the test execution, so launching the third instance was unnecessary. To have a visual understanding of the resource utilization, there is a graph which displays it. The slave1 saturation happened when the target number is users increased to 500. The following picture also provides information about the physical memory usage throughout the session.

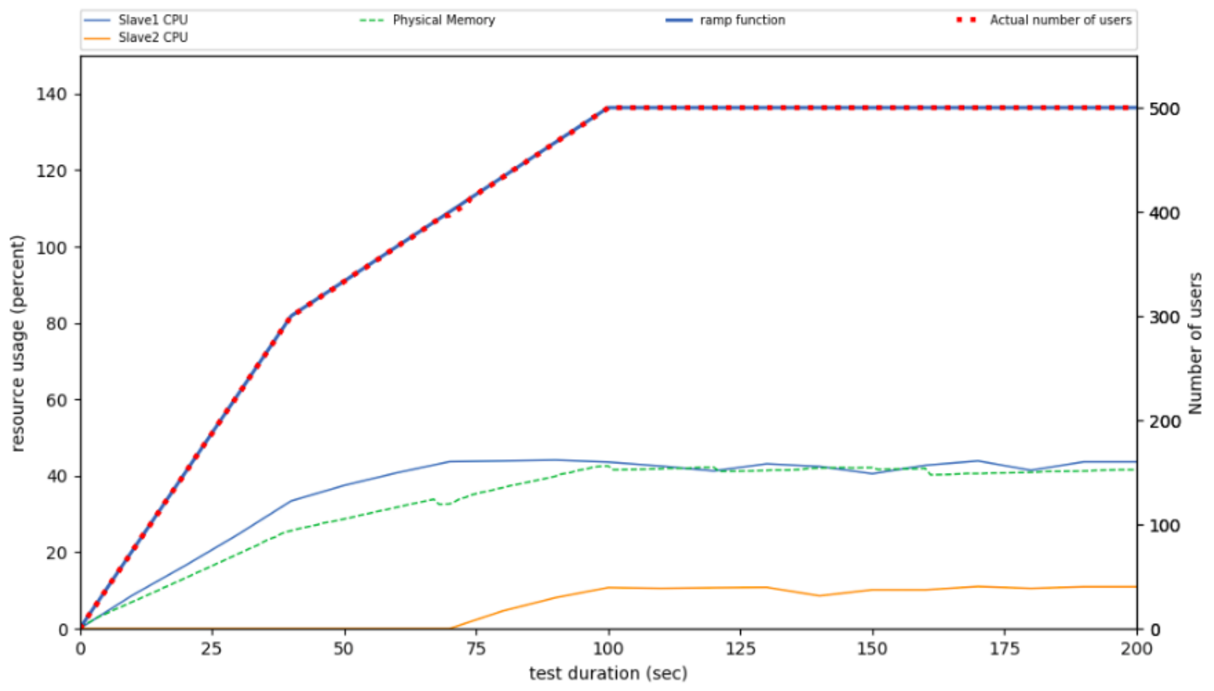


Figure 33: CPU and Memory Usage using t2.micro

If another instance type is used instead of t2.micro, the results about the resource utilization are slightly different. The following experiment runs using t2.xlarge with provides an instance with 4 virtual CPUs. The cost of launching the node based on this type is higher, especially if we launch 3 slave nodes at the same time. Since the instances have 4 CPUs, the system information was different compared to the previous experiment.

Total Disk read bytes: 52.0KiB
Average Disk read bytes: 252.4B/s
Total Disk write bytes: 10.0MiB
Average Disk write bytes: 48.7KiB/s
Total Network sent bytes: 2.2MiB
Average Network sent bytes: 10.6KiB/s
Total Network received bytes: 2.2MiB
Average Network received bytes: 10.6KiB/s
Physical Memory Usage: 2.87 %
Slave1 CPU 0 Usage: 32.15 %
Slave1 CPU 1 Usage: 32.02 %
Slave1 CPU 2 Usage: 31.96 %
Slave1 CPU 3 Usage: 31.64 %

The main difference is the number of used slave nodes. In this case, one slave was enough to run execute the test, since the load is distributed into the virtual CPUs.

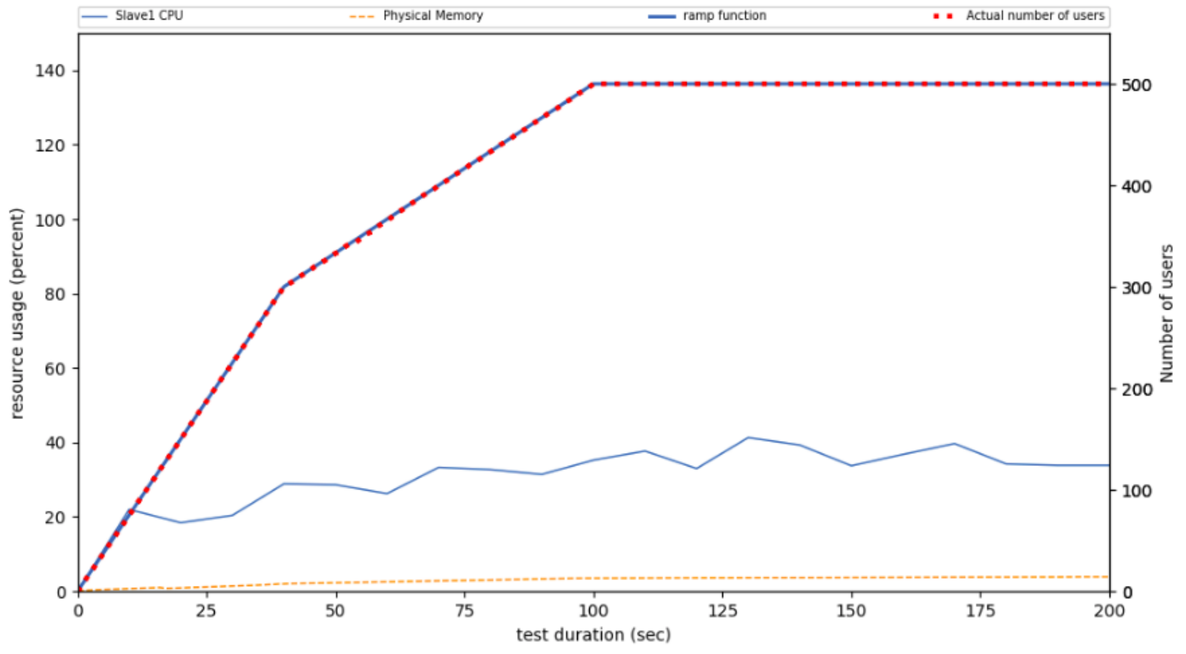


Figure 34: CPU and Memory Usage using t2.xlarge

7.4 Experiment 2

In the second experiment, the focus will be on evaluating if the system performs successfully with a different ramp and a longer test execution. The ramp function is *[(0, 0), (40, 300), (60, 400), (100, 600), (200, 600)],* 2 slave nodes have been used and the threshold was set up to 60%. The Target Response Time was not changed, since it is not the focus of the experiments. The same instance types were used, firstly the t2.micro with one virtual CPU and then t2.xlarge with 4 virtual CPUs.

During both tests, the slave nodes were initiated and generated load. The % of used CPUs differ and we will check in the system information below what their values are for each instance type. After executing the test with t2.micro, the system stats section of the report is as below:

Total Disk read bytes: 0.0B
Average Disk read bytes: 0.0B/s
Total Disk write bytes: 476.0KiB
Average Disk write bytes: 1.5KiB/s
Total Network sent bytes: 54.0MiB
Average Network sent bytes: 175.9KiB/s
Total Network received bytes: 34.4MiB

Average Network received bytes: 112.2KiB/s
Physical Memory Usage: 42.26 %
Slave1 CPU 0 Usage: 32.21 %
Slave2 CPU 0 Usage: 26.59 %

To get a better understanding when the second slave started and how many users generated, we refer to the graph which shows the CPU and memory usage.

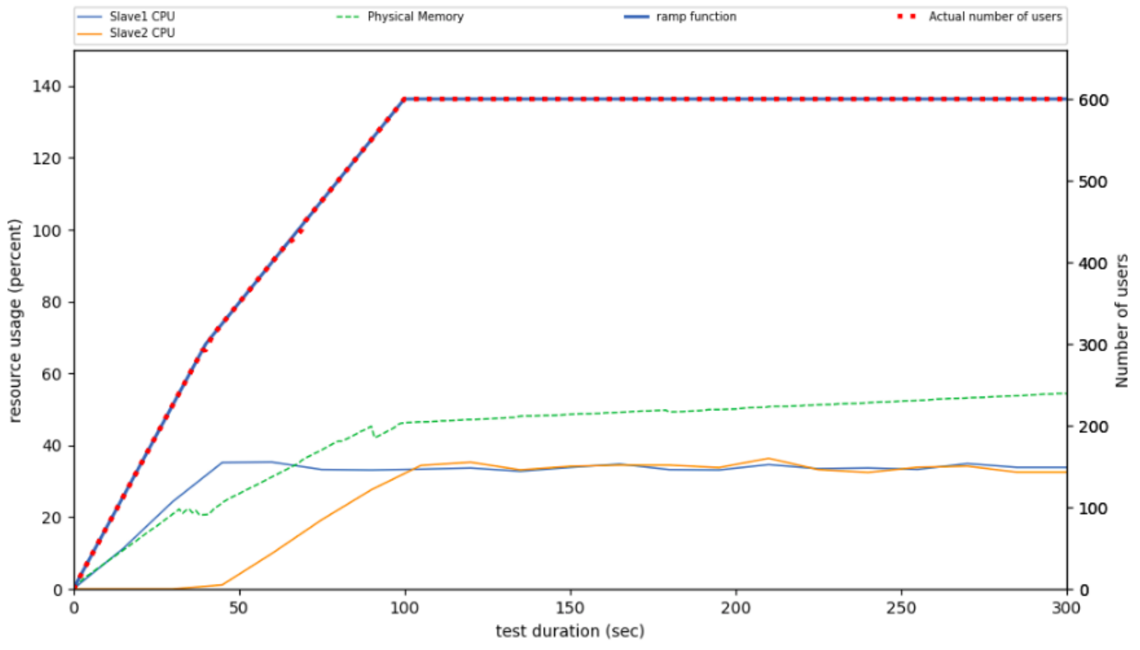


Figure 35: CPU and Memory Usage using t2.micro

It can be checked by the previous values and the graph that the load generation is almost equal in the slave nodes, while the physical memory usage goes up to 42.26%.

The achieved system information when using the t2.xlarge can be seen below and in the picture afterwards. Even though the virtual machines have 4 CPUs, 2 slave node were still used. More than half of the users were generated by the first slave node, which is the main difference from the previous experiment.

Total Disk read bytes: 152.0KiB
Average Disk read bytes: 497.3B/s
Total Disk write bytes: 44.2MiB
Average Disk write bytes: 144.4KiB/s
Total Network sent bytes: 147.6MiB
Average Network sent bytes: 483.0KiB/s
Total Network received bytes: 149.8MiB

Average Network received bytes: 490.0KiB/s
Physical Memory Usage: 3.64 %
Slave1 CPU 0 Usage: 55.67 %
Slave1 CPU 1 Usage: 55.72 %
Slave1 CPU 2 Usage: 55.4 %
Slave1 CPU 3 Usage: 55.53 %
Slave2 CPU 0 Usage: 44.8 %
Slave2 CPU 1 Usage: 44.61 %
Slave2 CPU 2 Usage: 44.53 %
Slave2 CPU 3 Usage: 44.82 %

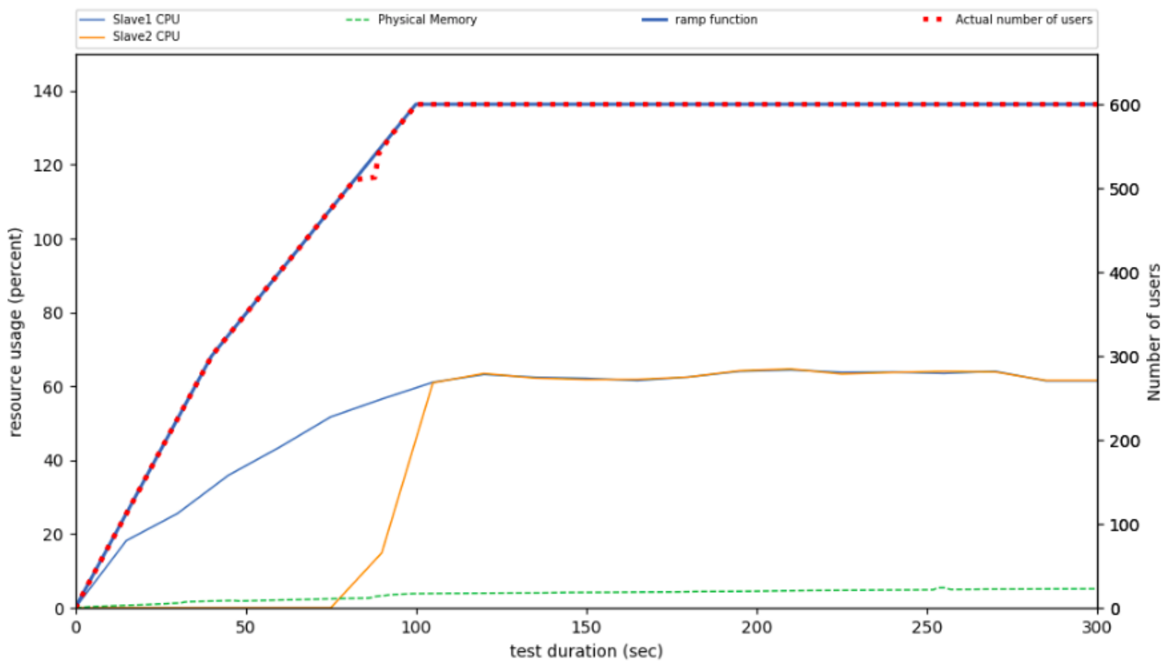


Figure 36: CPU and Memory Usage using t2.xlarge

Deciding the instance type to use as a node in the following system depends on the user's choices. In this experiment, the cost will be higher and the same results can be achieved by the first instance type, which is way cheaper than t2.xlarge.

7.5 Results and discussion

The above experiments let us understand what was correctly implemented and what requires more improvements. The latter one can be considered as future work to make the system fully and precisely functional. The most important result is that the system works as expected. The deployment on the cloud was successful, since the system resides now on the cloud, where components are deployed in separate but connected nodes. The fully featured web application facilitates significantly the user experience. The time spent to provide the test input data is shortened due to the GUI interface and, especially from the interactive features that were added by using JS libraries. It also provides the possibility of monitoring the test execution in real-time, action that is additionally included in the new solution. Executing different tests at the same time was achieved by the fact that every test launches its own instances, so there is no obstacle met.

On the other hand, some flaws were observed during the experiments. In some cases, they mostly affect the performance of the system, and not the functionality. This implies that future improvements are needed in order to provide a better solution.

The first glitch is related to saving costs while initiating the number of slaves. The tool requires that the number of slaves is provided before running the test and all the slaves need to be listening to the master node so that the execution can begin. This indicates that EC2 instances are launched before the test execution, even though not all slaves may be used. Their usage or not depend on the specified threshold or on the test case. Keeping the instances alive costs money, even if they are not used. In order to avoid this, small changes can be implemented on the tool to support the on-the-fly initialization of slave nodes. In this way, instances would be launched only when the threshold of the current slave node is breached, and a new node is required. The only drawback that the change might cause is on the system's performance. Launching a new instance during the test execution increases latency and decreases the performance.

Another improvement is also related to reducing costs. Instances are created and used only for a specific test, but what if two or more tests are run at the same time? Instead of launching new instances, the active ones can be used at the same time by executing the test using another port. This would require some changes in controlling which instances are still running a test before terminating them and specify correctly the port to be used to avoid running the tests to the same TCP port.

The last recommended change is related to the folder created with the input data. SCP was used to copy the folder between the web application and the master node, but this increases unnecessary latency during the test execution and affects negatively the system performance. It is necessary to consider using Amazon S3 as a solution to this problem. It is an object storage service, used to store some amount of data. Instances can then make requests using an API to retrieve the data (input data folder) from the S3 without the need of copying them. [55]

8. Conclusion

The purpose of the thesis was to use the existing MBPeT tool for the implementation of Model-Based Performance Testing as a Service. This solution revokes the user from the need of installing the tool locally. Additionally, a fully featured web application is included in the new solution to provide the users with a more user-friendly way when specifying the test input data. Interactive functionalities were included in the new solution from the usability point of view.

Three different theoretical chapters explained the necessary concepts behind the project. Since it involved implementing a web application, the first chapter explained all terms related to it, going into the details of the used programming languages and discuss why they suit to the system. The following chapter analyzed cloud computing and its service models. This came from the "as a Service" in the thesis topic and goal. The last theoretical part described the software testing notions related to Model-Based Performance Testing. It also gave an overview of the MBPeT tool, its architecture and how it works.

The practical part began with listing the functional and non-functional requirements that the system should fulfill. It continued with illustrating how the system architecture was designed and details about the implementation were given. The theoretical features of the used technologies were evaluated in this chapter and it was confirmed that they eased the implementation part in many aspects.

After analyzing the undertaken steps for the implementation, the last phase included evaluating the provided solutions by running some experiments. They allowed to understand what was successfully achieved according to the requirements and what can be improved as a future work. Small fixes which would benefit the cost of using the service can be undertaken and other stated changes would affect the system performance.

Despite the fixes and changes which will improve the system, the project resulted successful in terms of requirements fulfillments. It provides a user-friendly web application with additional features which simplify the user experience. JavaScript was the perfect choice for satisfying the usability requirement. Security and maintainability are followed strictly and facilitated by using Django as the main technology. The deployment of the system in the cloud and the communication were accomplished using AWS as a cloud provider and Python for the communication between the nodes.

9. References

[1] T. Ahmad, F. Abbors, D. Truscan and I. Porres, “Model-Based Performance Testing Using the MBPeT Tool,” Turku Centre for Computer Science, Turku, 2013.

[2] S. Herbold and A. Hoffmann, “Model-based testing as a service”, International Journal on Software Tools for Technology Transfer, vol. 19, no. 3, pp. 271-279, June 2017.

[3] J. Peleska, “Industrial-Strength Model-Based Testing - State of the Art and Current Challenges,” Journal reference: EPTCS 111, pp. 3-28, 2013.

[4] L. Shklar and R. Rosen, Web Application Architecture Principles, Protocols and Practices, John Wiley & Sons, Ltd, England, 2003.

[5] H. Sh. Oluwatosin, “Client-Server Model,”, IOSR Journal of Computer Engineering, vol. 16, issue 1, pp. 67-71, Feb. 2014.

[6] Client-Side vs. Server-Side: What’s the Difference? [Online] Available: https://www.seguetech.com/client-server-side-code/ [Accessed 5 February 2019]

[7] JavaScript [Online] Available: https://www.tutorialspoint.com/javascript/index.htm [Accessed 1 December 2018]

[8] Uses of JavaScript [Online] Available: https://www.sitesbay.com/javascript/javascript-uses [Accessed 10 February 2019]

[9] jQuery [Online] Available: https://jquery.com/ [Accessed 10 January 2019]

[10] Getting Started – Ajax [Online] Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started [Accessed 10 January 2019]

[11] AJAX Introduction [Online] Available: https://www.w3schools.com/xml/ajax_intro.asp [Accessed 4 February 2019]

[12] Introducing JSON [Online] Available: http://www.json.org/ [Accessed 10 January 2019]

[13] Server-Side Scripting: Back-End Web Development Technology [Online] Available: https://www.upwork.com/hiring/development/server-side-scripting-back-end-web-development-technology/ [Accessed 1 March 2019]

[14] MVC Framework – Introduction [Online] Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm [Accessed 20 December 2018]

[15] Django [Online] Available: https://www.djangoproject.com/start/overview/ [Accessed 1 December 2018]

[16] Django: A Python-Powered Development Framework [Online] Available: https://www.upwork.com/hiring/development/django-programming/ [Accessed 15 January 2019]

[17] Django’s Structure – A Heretic’s Eye View [Online] Available: https://djangobook.com/mdj2-django-structure/ [Accessed 1 December 2018]

[18] P. Srivastava and R. Khan, “A Review Paper on Cloud Computing,” International Journals of Advanced Research in Computer Science and Software Engineering, vol. 8, issue 6, pp. 17-20, June 2018.

[19] P. Mell and T. Grance, The NIST Definition of Cloud Computing, Communications of the ACM. 53. 10.6028/NIST.SP.800-145, September 2011.

[20] What is cloud computing? [Online] Available: https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/ [Accessed 1 February 2019]

[21] Cloud Computing – An Emerging Technology! [Online] Available: https://blog.oureducation.in/cloud-computing-an-emerging-technology/ [Accessed 5 March 2019]

[22] K. Hwang, G.C. Fox, J. J. Dongarra, Distributed and Cloud Computing, Morgan Kaufmann, 2012.

[23] A. Shawish and M. Salama, “Cloud Computing: Paradigms and Technologies,” in F. Xhafa and N. Bessis, Inter-cooperative Collective Intelligence: Techniques and Applications. Springer, pp. 39-67, 2014.

[24] T. Dillon, C. Wu and E. Chang, "Cloud Computing: Issues and Challenges," 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, WA, 2010, pp. 27-33, 2010.

[25] Cloud Service Models (IaaS, PaaS, SaaS) Diagram [Online] Available: https://dachou.github.io/2018/09/28/cloud-service-models.html [Accessed 1 March 2019]

[26] The Pros and Cons of SaaS vs On-premises Deployment [Online] Available: https://smartbridge.com/pros-cons-saas-vs-premises-deployment/ [Accessed 15 February 2019]

[27] J. Gross, “SaaS versus on-premise ERP,”, A Ziff Davis White Paper, January 2012.

[28] SDLC – Waterfall Model [Online] Available: https://www.tutorialspoint.com//sdlc/sdlc_waterfall_model.htm [Accessed 10 February 2019]

[29] SDLC – V-Model [Online] Available: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm [Accessed 10 February 2019]

[30] J. A. Whittaker, "What is software testing? And why is it so hard?," in *IEEE Software*, vol. 17, no. 1, pp. 70-79, Jan.-Feb. 2000.

[31] G. J. Myers, T. Badgett and C. Sandler, The Art of Software Testing, John Wiley & Sons, Inc., Hoboken, New Jersey 2012.

[32] P. Ammann and J. Offutt, Introduction to Software Testing, New York, NY.: Cambridge University Press, 2017.

[33] Certified Tester, Foundation Level Syllabus, International Software Testing Qualifications Boards, 2005

[34] S. Dhawan, N. Kumar and S. Saini, “Model Based Testing Considering Steps, Levels, Tools and Standards of Software Quality,” Journal of Global Research in Computer Science, vol. 2, no. 4, pp.148-151, April 2011.

[35] C. Barna, M. Litoiu, and H. Ghanbari, “Model-based performance testing: NIER track,” in Proceedings of the 33rd Internation Conference on Software Engineering. ICSE ‘11. Waikiki, Honolulu, HI, USA: ACM, pp. 872-875, June 2011.

[36] F. Abbors, T. Ahmad, D. Truscan and I. Porres, “Model-Based Performance Testing of Web Services Using Probabilistic Timed Automata,” in Proceedings of the 2013 10th International Conference on Web Information Systems and Technologies, 2013.

[37] G. Norman, D. Parker and J. Sproston, “Model Checking for Probabilistic Timed Automata,” Formal Methods in System Design, vol. 43, no. 2, pp. 164-190, October 2013.

[38] Graphviz, “The DOT Language,” graphviz, [Online]. Available: http://www.graphviz.org/ [Accessed 10 January 2019].

[39] D. J. Barrett, R. E. Silverman and R. G. Byrnes, SSH, the Secure Shell: The Definitive Guide, O’Reilly Media, Inc., 2005

[40] ITEA2 D-mint project result leaflet: Model-based testing cuts development costs. https://itea3.org/project/result/download/5519/D-, October 2010.

[41] Boto 3 Documentation [Online] Available: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html [Accessed 10 April 2019]

[42] Python – Overview [Online] Available: https://www.tutorialspoint.com/python/python_overview.htm [Accessed 20 December 2018]

[43] SQLite [Online] Available: https://www.sqlite.org/index.html [Accessed 10 December 2018]

[44] Distinctive Features of SQLite [Online] Available: https://www.sqlite.org/different.html [Accessed 23 December 2018]

[45] Amazon Web Services [Online] Available: https://aws.amazon.com/ [Accessed 1 April 2019]

[46] CodeMirror [Online] Avaiable: https://codemirror.net/ [Accessed 15 January 2019]

[47] vis.js [Online] Available: http://visjs.org/ [Accessed 20 December 2018]

[48] vis.js Network [Online] Available: http://visjs.org/docs/network/ [Accessed 20 December 2018]

[49] Highcharts [Online] Available: https://www.highcharts.com/ [Accessed 1 February 2019]

[50] Amazon EC2 [Online] Available: https://aws.amazon.com/ec2/ [Accessed 1 April 2019]

[51] Launch a Virtual Machine [Online] Available: https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/ [Accessed 1 April 2019]

[52] Apache Libcloud [Online] Available: http://libcloud.apache.org/ [Accessed 20 March 2019]

[53] Amazon EC2 Security Groups for Linux Instances [Online] Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html [Accessed 1 April 2019]

[54] subprocess – Subprocess Management [Online] Available: https://docs.python.org/3/library/subprocess.html [Accessed 4 March 2019]

[55] Amazon S3 [Online] Available: https://aws.amazon.com/s3/ [Accessed 15 April 2019]