CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of computers science

Bachelor's Thesis

System for creating scenarios for F-Tester platform

Artem Kelpe
Software Engineering and Technology

May 2019
Supervisor: Ing. Zbyněk Kocur, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kelpe** Jméno: **Artem** Osobní číslo: **437859**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro tvorbu scénářů platformy F-Tester

Název bakalářské práce anglicky:

F-Tester Platform Scenario Creation System

Pokyny pro vypracování:

Navrhněte systém pro tvorbu testovacích scénářů pro platformu F-Tester. Systém umožní tvorbu definic jednotlivých testů, ale také z nich složených scénářů. Implementujte modul do serverové části F-Testeru, taktéž navrhněte a implementujte modul do uživatelského rozhraní F-Testeru. Důraz zaměřte na uživatelskou přívětivost tvorby a konfigurace testů a scénářů. Navržený systém otestujte.

Seznam doporučené literatury:

[1] Scott, E.: SPA Design and Architecture: Understanding Single Page Web Applications 1st Edition. Manning Publications, 2015. 275 stran. ISBN: 1-61729-243-5.
[2] Podpůrné materiály platformy FLOWTESTER. Dostupné na: https://flowtester.fel.cvut.cz/ [on-line]
[3] Studijní materiály dostupné na https://openwrt.org/ [on-line]

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Zbyněk Kocur, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Zbyněk Kocur, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

/ Declaration

I hereby declare that the present bachelor's thesis was composed by myself and that the work contained herein is my own. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the Methodological Guideline on Ethical Principles for College Final Work Preparation.

Prague, May 24, 2019

....

Abstrakt / Abstract

F-Lab je platforma pro testování vlastností a schopností počítačových sítí. F-Tester je součástí F-Lab, která je zodpovědná za měření parametrů komunikace mezi zařízeními založenými na protokolu TCP/IP. Test Planner je modul pro F-Tester, který umožňuje vytvářet testovací konfigurace a kombinovat je do scénářů, které může F-Tester provádět. Tento dokument obsahuje analýzu návrhu Test Planneru a popis jeho implementace.

Klíčová slova: Počítačová síť, NGN, NGA, F-Tester, OpenWRT, Lua, LuCI, VueJS

F-Lab is the platform for testing properties and abilities of computer networks. F-Tester is a part of F-Lab that is responsible for measuring parameters of communication between devices based on TCP/IP protocol. Test Planner is the module for F-Tester that allows to create test configurations and combine them into scenarios that can be executed by F-Tester. This document contains design analysis of Test Planner as well as its implementation description.

Keywords: Network, F-Tester, NGN, NGA, OpenWRT, Lua, LuCI, VueJS

/ Contents

Tables /

Chapter 1
Introduction

Networking as the part of IT science is the still growing and fast developing sphere, new technologies, protocols and specifications appear every day. Their development and introduction lead to new requirements both for hardware and for software to meet.

One of the possible ways of networks development is transitioning to new standards called NGA/NGN (Next-Generation Access/Next-Generation Network). This standard defines network with packet commutation that can be used with different broadband transport technologies, where service functions do not depend on transport technologies. NGN is more about software rather than hardware, so it can work with already existing hardware infrastructures [1]. Some major telecom companies such as China Telecom, Bulgarian Telecommunication Company, KPN (Netherland) and others are already implementing this standard.

There are some test and criteria that can determine if the selected network can be considered as the Next-Generation one. F-Lab, which is a product for measuring network parameters, is one of such tools. In order to prepare and execute test scenarios of such measurements, it needs a special module, which can add the new way of interaction between the user and the system. Aim of this project is to develop such a module, which is called Test Planner, and integrate it into the existing system.

This document gives an insight into the F-Tester project (chapter 2), describes the design and requirements for Test Planner module (chapter 3), its implementation details (chapter 4), testing (chapter 5), process of deployment on working F-Tester instance (chapter 6) and conclusion with analysis of fulfilment of project goals (chapter 7).

1.1 Project goals

The goals of this thesis are the following:

- Review F-Lab and F-Tester platforms
- Analyze way of development and integration of custom modules for them
- Define requirements for Test Planner
- Implement Test Planner module
- Test the module on specified use cases

Chapter 2
Background

This chapter is dedicated to the technologies and platform upon which Test Planner is built or that are connected with it.

2.1 Network flow

Network flow (also known as traffic flow or packet flow) is a sequence of packets that goes through a computer network. Different RFC describe flow in different ways:
RFC 2722: For the purpose of traffic flow measurement we define the concept of a traffic flow, which is like an artificial logical equivalent to a call or connection [2].
RFC 3697: A flow is a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow. A flow could consist of all packets in a specific transport connection or a media stream. However, a flow is not necessarily 1:1 mapped to a transport connection [3].
RFC 3917: A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties [4].
Flows are an essential part of network analysis as they provide more useful data about network events than single packets.

2.2 F-Tester Platform

F-Tester (previously named as FlowTester) is a part of a complex solution called **F-Lab** that is being developed on the Department of Telecommunication Engineering (Faculty of Electrical Engineering, CTU in Prague). F-Lab is a complex testing and simulating system that verifies properties and abilities of data networks. F-Tester is the hardware and software component of this system that allows measuring parameters of communication devices based on the TCP/IP protocol family.

2.2.1 Technologies

F-Tester runs on the Unix-like open-source operation system **OpenWrt**. This is a lightweight highly extendible GNU/Linux distribution built from scratch for embedded devices (typically wireless routers). OpenWrt is rather often used to replace stock firmware of routers due to its security, stability and extensibility. In 2016 some developers left the OpenWrt project team and forked original OS and then named it **LEDE** (Linux Embedded Development Environment). After two years projects OpenWrt and LEDE were merged together and since then they exist as a single project under OpenWrt name. During the period of existence LEDE F-Tester project had been based on that operational system.

When configuring a traditional Unix system the user has to fill a big amount of text configuration files most of which have different syntax. Some services have to be configured via executing the special command with different parameters. Instead of this OpenWRT has created **UCI** (Unified Configuration Interface) which allows managing the most of the system parameters via unified syntax of files and command line parameters. Instead of storing different configuration files in different locations of filesystem, UCI stores everything under */etc/config* path, for example, configuration of network interfaces which is located at */etc/network/interfaces* can be changed by editing */etc/config/network*, or Samba/CIFS can be configured by */etc/config/samba* instead of typical */etc/samba/smb.conf*. While running init.d initialization scripts, UCI simply parses concrete */etc/config* files and overrides original configuration files. UCI files can be modified not only by manual editing text files, but also with the help of command line utility *uci*. Moreover, that files are also modifiable via various programming API (like Shell, Lua and C), which is the way how different interfaces like LuCI make changes to the UCI files.

LuCI is a free, clean and extensible web user interface for embedded devices. It uses Lua programming language and splits the interface up into logical parts like models and views, uses object-oriented libraries and templating, what in common ensures better performance, smaller installation size, faster runtimes and simple maintainability. LuCI provides user with another way of editing UCI configuration files by simply using web browser.

2.3 Integration with F-Tester

LuCI provides an easy and well-documented interface for implementing and integrating custom modules. In terms of LuCI single module is a package with MVC structure, which allows extending default functionality of LuCI, where

- **Model** is a necessary part of the module only if it is planned to be used as editor of configuration files. In that case, it consists of files where the developer describes the structure of the configuration file
- **View** is one or more HTML files written in special format that can be recognised by Lua's regex based template processor. Those files can be used both for controlling the module and show some results of command execution.
- **Controller** is de facto list of actions that the module can execute on the backend side. Often actions include editing configuration files, showing statistics or executing scripts. Actions are defined as functions that are executed on a request from the client side.

Process of integration of custom module is described in more details in chapter 6.

Chapter 3
Design

3.4 Existing solutions

As the F-Tester platform is the product of Dept. of Telecommunication Engineering at FEE and because the problem is rather specific, currently there is no any kind of alternative software that can be used as the part of F-Tester for scenarios production. The only option how to create them is to manually write test scenarios in the text editor and then send them to the test runner module.

3.5 Requirements

3.5.1 Functional requirements

Functional requirement is such a requirement that describes behaviour or specification of system, i.e. what the system should do. Test Planner have to be able to accomplish next functionality:
FRQ1 Create new test configuration
FRQ2 View list of test configurations
FRQ3 Edit existing test configuration
FRQ4 Delete existing test configuration
FRQ5 Create new test scenario
FRQ6 View list of test scenarios
FRQ7 Run existing test scenario
FRQ8 Plan next run of existing scenario at predefined time
FRQ9 Edit existing test scenario
FRQ10 Delete existing test scenario

3.5.2 Non-functional requirements

Non-functional requirements (also known as `quality requirements`) in contrast to functional requirements describe how the system should work. Test Planner have to accomplish next non-functional requirements:
NFRQ1 Application should operate properly in modern browsers (Chrome $\geqslant$ 72, Firefox $\geqslant$ 65, Safari $\geqslant$ 12, Opera $\geqslant$ 58, Edge $\geqslant$ 18)
NFRQ2 Application should be able to validate input data in order not to cause error while running test scenarios
NFRQ3 Application's GUI should have similar look to other system components design

3.6 Use-case diagram

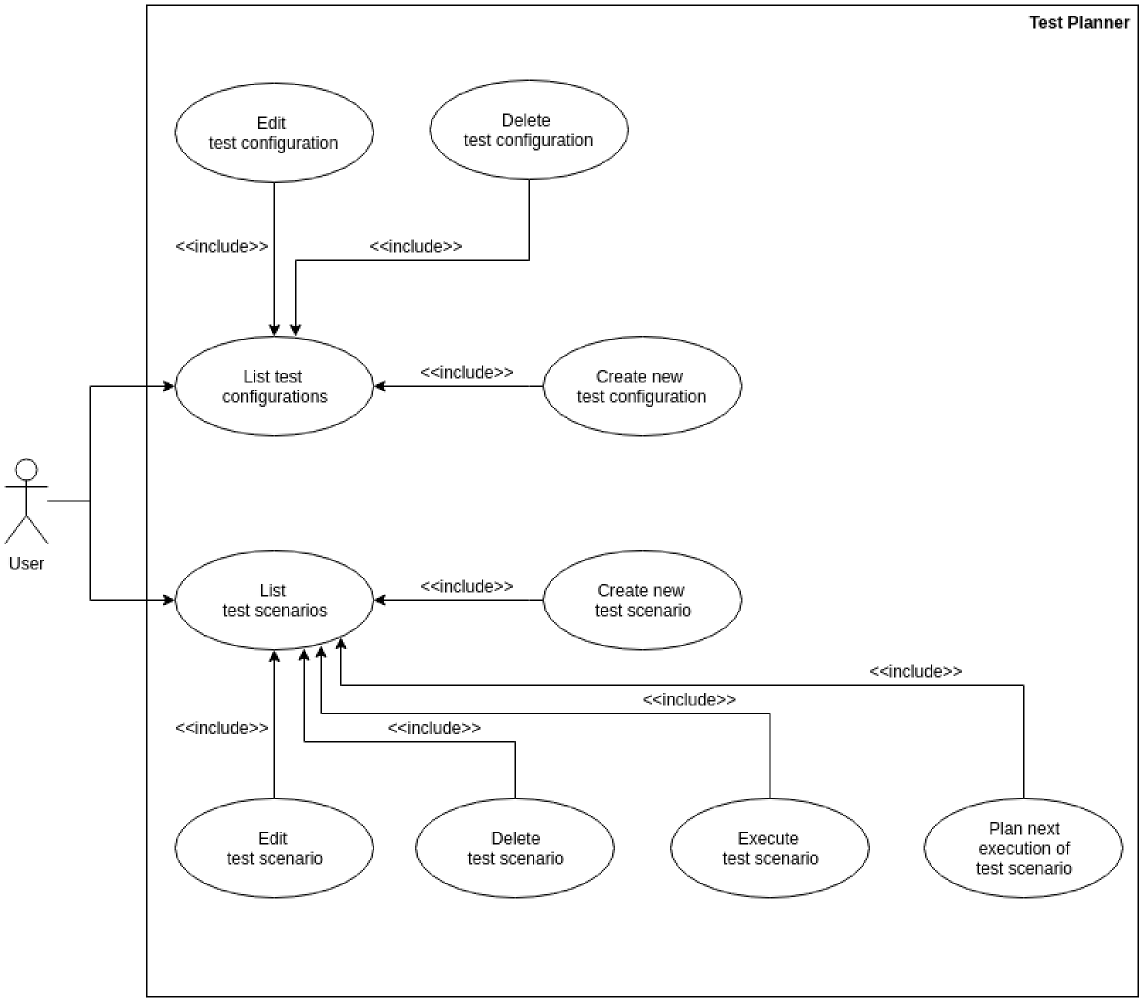

Figure 2.1. Schematic description of application architecture

Chapter 4
Implementation

4.1 Application architecture

Test Planner is a typical web application with backend-frontend architecture and REST API used as an interface for communication between the server and client side. Backend side is used mostly for CRUD operations with data that are stored as JSON files on the system storage. Frontend side consists of Vuex storage (which is the implementation of Flux architecture for VueJS) and components for each entity in this application. Next diagram shows relations between components and typical data flows:

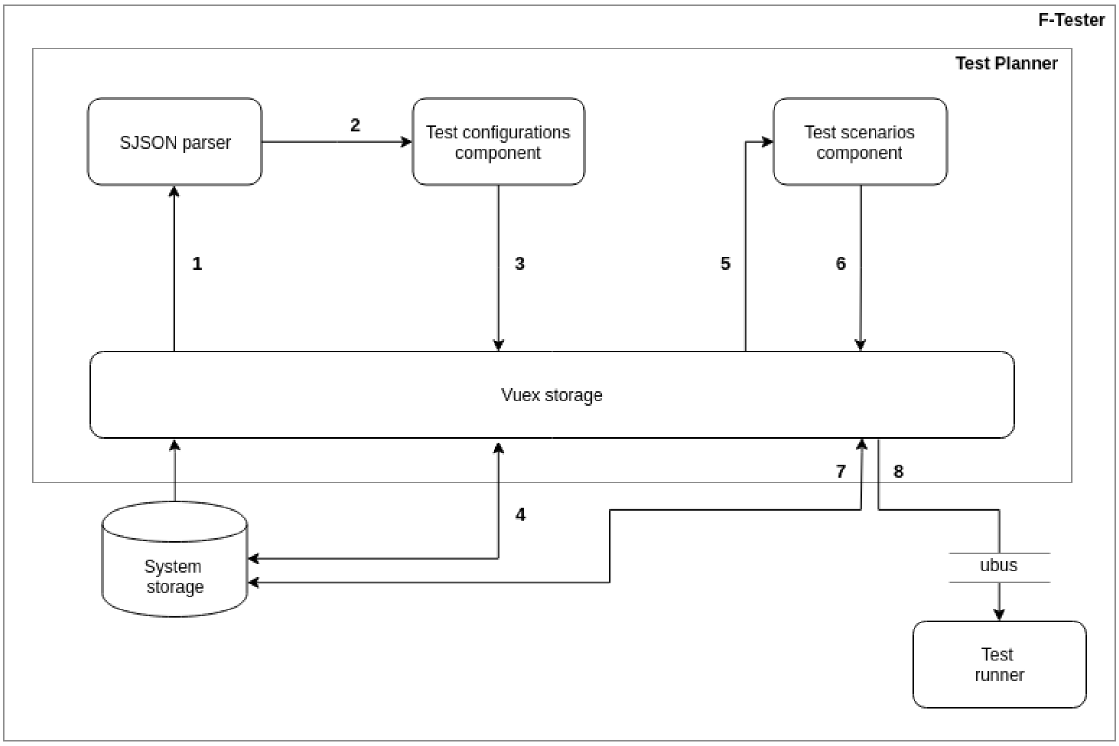

Figure 4.1. Schematic description of application architecture

List of operations and dataflows on the schema:

- 1 - System storage provides application with predefined SJSON files
- 2 - SJSON parser produces web form based on received SJSON file
- 3 - Newly created test configurations are stored in the Vuex storage
- 4 - Vuex storage synchronizes with system storage
- 5 - Vuex storage provides test scenarios component with test configurations
- 6 - Newly created test scenarios from test configurations are stored in Vuex storage
- 7 - Vuex storage synchronizes with system storage
- 8 - Test scenario is sent to test runner component of F-Tester

4.2 Technology stack

This section describes technologies, architectures, programming languages and frameworks that were chosen to implement this project.

4.2.1 Backend

Due to possible limitations of hardware infrastructure and potential isolation from the Internet, any package for LuCI or OpenWRT is expected to use other pre-installed packages and not to download/install new packages as dependencies. So in order to minimise package's size, Lua was selected as the main programming language to implement the backend side of the Test Planner. In terms of this application, backend is used only to create, read, edit and delete entities, and the data is stored as serialized JSON files, so JSONC was used as the library for working with JSON files as it is the standard library for LuCI. LuCI's application server is also used to give back HTML and static files such as Javascript and CSS ones.

4.2.2 Frontend

As the most operations are performed on the frontend side of applications, it was decided to use JavaScript framework instead of the vanilla version. Moreover, GUI of application is expected dynamically refresh its content, for example, successful editing particular item on the web page leads to the view with the preview of all entities, so it is reasonable to use some kind of Flux storage on the client side of application. Next table shows criteria that were used while selecting of frontend framework, and the decision of that criteria, where value "1" or green colour is the best option and value "3" or red color is the worst option from the point of the author.

Framework name	React	VueJS	Angular
Designed for	■ Web apps, SPA	■ Web apps, SPA	■ Web apps
Needed knowledge of languages	■ JavaScript, JSX	■ JavaScript	■ TypeScript
Popularity & knowledgebase	■ 1	■ 2	■ 2
Existing libraries	■ 1	■ 2	■ 2
Easy to start	■ 2	■ 1	■ 3
Standard for Flux-like storage	■ 1	■ 1	■ 2
Already used in whole project	■ 3	■ 1	■ 3

Table 4.1. Decision table for frontend frameworks

Flux-like storage is included in this table as an important criterion as it is a rational and suitable technology to use. Commonly, Flux is the architecture for providing operations with data based on the idea of unidirectional data flow. In VueJS there is an officially recommended and supported Flux implementation that is called Vuex, which is also used in Test Planner implementation. Flux pattern makes process of data handling more organized and ordered, and it also works with framework's reactivity, which allows to create independent components that always present actual data.

As it can be seen from the table, VueJS is the most convenient solution as the frontend JavaScript framework in this case.

4.3 Data transformation

In order to accomplish the task of generating predefined web forms, a special dialect of JSON called SJSON (Special JSON) was created. This kind of object notation was inspired by JSON Schema - technology for annotating and validating JSON objects/documents. The main purpose of SJSON is to provide information about possible structure of the final JSON object, necessity of its attributes and their possible values. Due to that SJSON files Test Planner can dynamically generate web forms for creating test configurations. One SJSON file is mapped to one console application that can be executed in console.

Sample SJSON file contains the name of test type and list of its options:

```
"testTypeName": [testOption1, testOption2, ...]
```

Each option is mapped to one attribute of the console application that will be executed. It provides rules of validating its value:

```
"testOption":{
     "name" : "optName",
     "label": "Option Name",
     "type" : "string" OR "number" OR "checkbox" OR "select",
     "minVal": 0.1,
     "maxVal": 150,
     "options": [selectOption1, selectOption2, ...],
     "multiselect": true OR false,
     "optional" : true OR false,
     "defaultValue": "something",
     "unmodifiable": true OR false,
     "mapping": {mappingScheme}
}
```

where *minVal* a *maxVal* works only for type `number` and *options* and *multiselect* work only for type `select`.

In case of type `select` there should be presented list of possible values:

```
"selectOpt": {
     "label": "Option name",
     "value": "value"
}
```

where *label* is text that will be shown for this option and *value* is value that will be used in test configuration.

The last part of option is mapping object that defines how this value will be used in console application execution:

```
"mappingObject": {
     "type": "main" OR "opt",
     "arg": true OR false,
     "str": "-p",
     "position": "begin" OR "end"
}
```

where *type* describes if this value will be stored as a key-value pair in final test configuration object (`main`) or if it will be added as an option to the string with command that executes application (`opt`). If *arg* is `true`, then value from *str* will be used as an argument for command line, otherwise value of test option will be used without any prefix. Attribute *position* allows defining the order of argument in final string.

For example, SJSON object for execution of program `ping` will look like this:

```
"ping":[
    {
        "name": "program",
        "label": "Program",
        "type": "string",
        "optional": false,
        "defaultValue": "ping",
        "unmodifiable": true,
        "mapping": {
            "type": "main"
        }
    },
    {
        "name": "target",
        "label": "Target host",
        "type": "string",
        "optional": false,
        "mapping":{
            "type": "main"
        }
    },
    {
        "name": "count",
        "label": "Count of icmp packets to send",
        "type": "number",
        "minVal": 1,
        "maxVal": 1000,
        "defaultValue": 5,
        "optional": false,
        "mapping":{
            "type": "opt",
            "arg": true,
            "str": "-c"
        }
    },
    {
        "name": "interval",
        "label": "Interval between send packets [s]",
        "type": "number",
        "minVal": 1,
        "maxVal": 60,
        "defaultValue": 1,
        "optional": false,
        "mapping":{
            "type": "opt",
            "arg": true,
            "str": "-i"
        }
    },
    {
        "name": "packet_size",
        "label": "Packet size [bytes]",
        "type": "number",
```

```
            "minVal": 1,
            "maxVal": 1000,
            "defaultValue": 56,
            "optional": false,
            "mapping":{
                "type": "opt",
                "arg": true,
                "str": "-s"
            }
        }
    ]
```

Web form produced with SJSON will look like this:

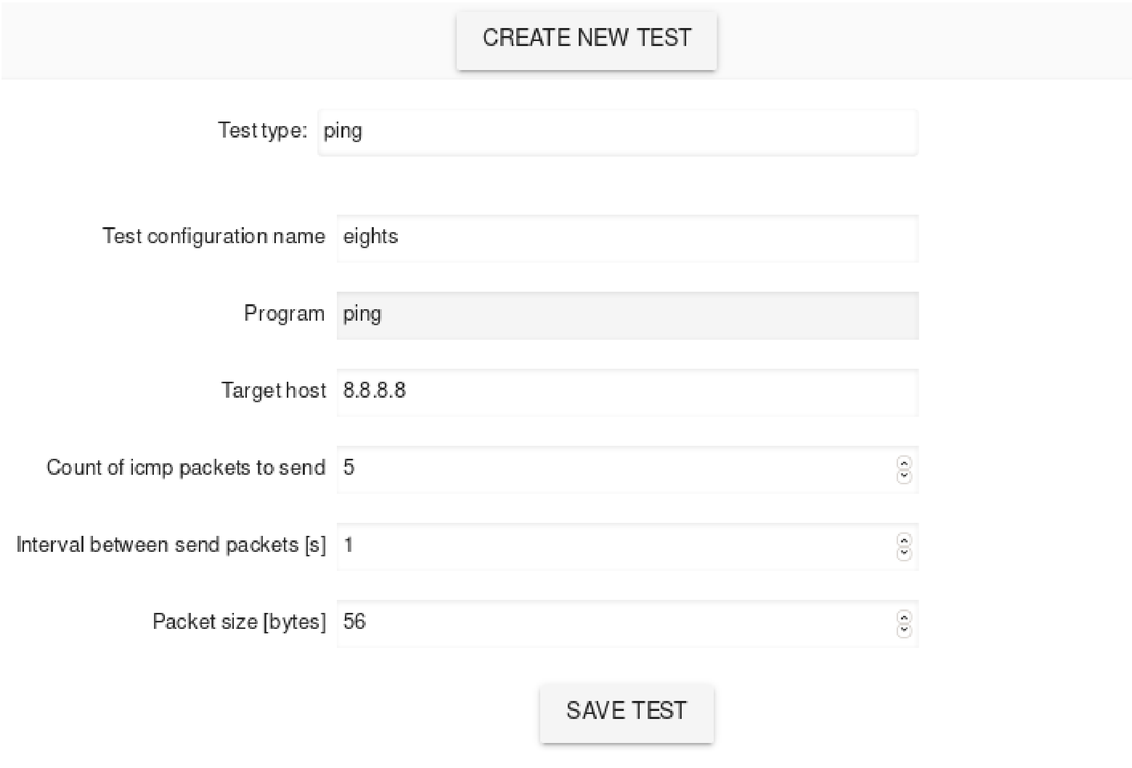

Figure 4.2. Screenshot of web form generated by ping example SJSON

And the final test configuration will contain next implementation:

```
{
    "type": "ping",
    "name": "eights",
    "params": {
        "program": "ping",
        "target": "8.8.8.8",
        "opts": "-c 5 -i 1 -s 56"
    }
}
```

4.4 Graphical User Interface

Frontend part of Test Planner is implemented leaning on basic principles of SPA. SPA (Single Page Application) is a modern pattern of building web application when user once loads an HTML page and then interact with it while page dynamically rewrites its content. Such approach allows to avoid interruptions of UI between loading pages and makes an application to look like a native one [5]. That means that user once download whole web application, all its content and components/graphical elements of UI and after that communicate with the server only in order to send or retrieve some data via backend API. SPA are often built with JavaScript MVC/MVVM frameworks such as React, AngularJS or VueJS.

After opening the application page there are 3 tabs that can be browsed:

- `Overview` - information about already executed or planned scenarios executions
- `Scenarios` - list of created test scenarios; forms to create new or edit existing scenarios
- `Tests` - list of created test configurations; forms to create new or edit existing configurations

Figure 4.3. Screenshot of panel with tabs

Lists of scenarios and configurations are presented as simple tables, where each row contains name and description of every single item as well as control elements for editing or executing and deleting them.

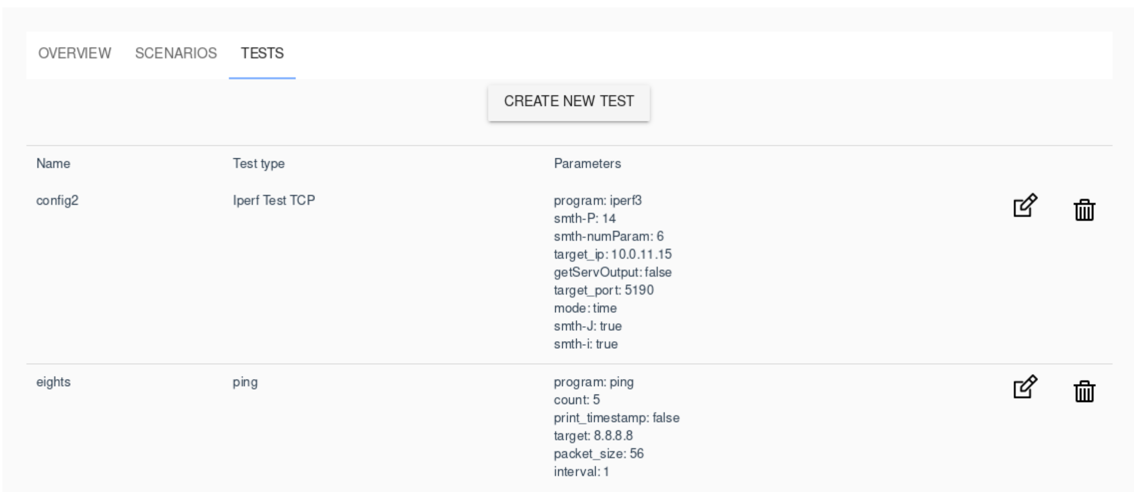

Figure 4.4. Screenshot of test configurations preview

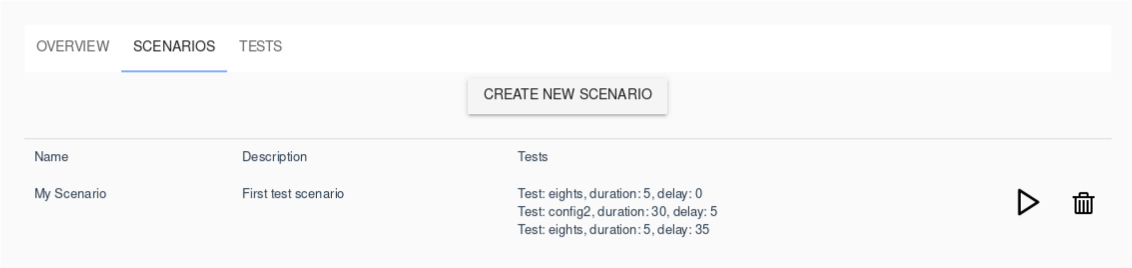

Figure 4.5. Screenshot of test scenarios preview

On each page there is also a button for creating a new item. Clicking on it opens the form where user can build up new test configuration or new test scenario from existing configurations.

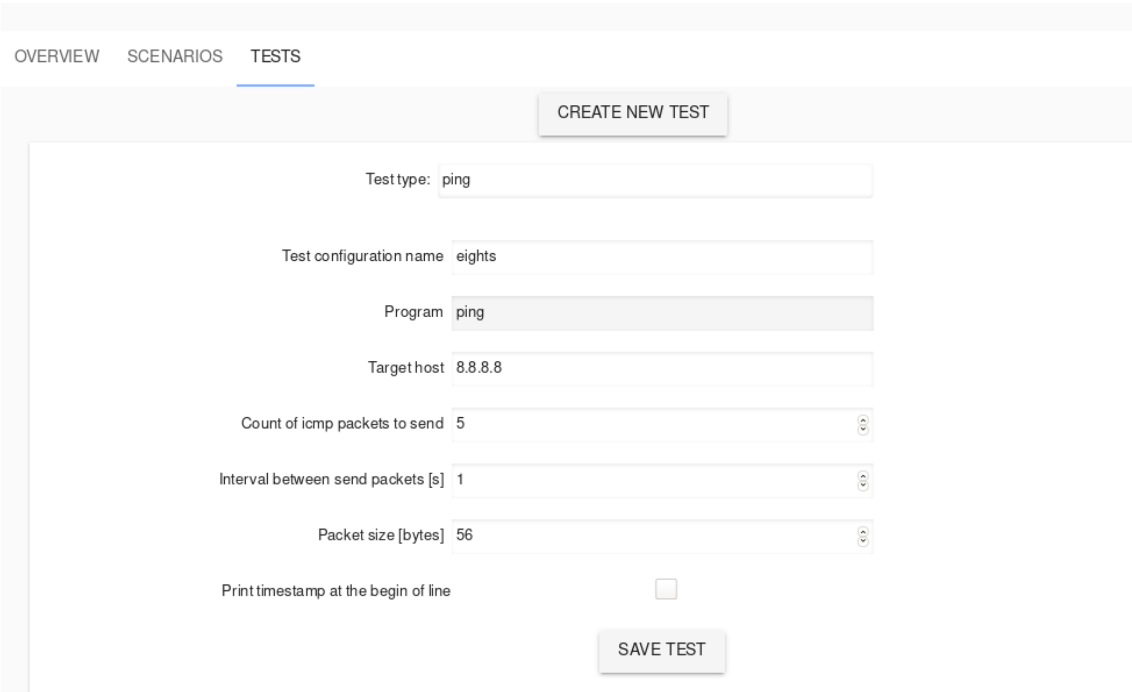

Figure 4.6. Screenshot of form for creating test configuration

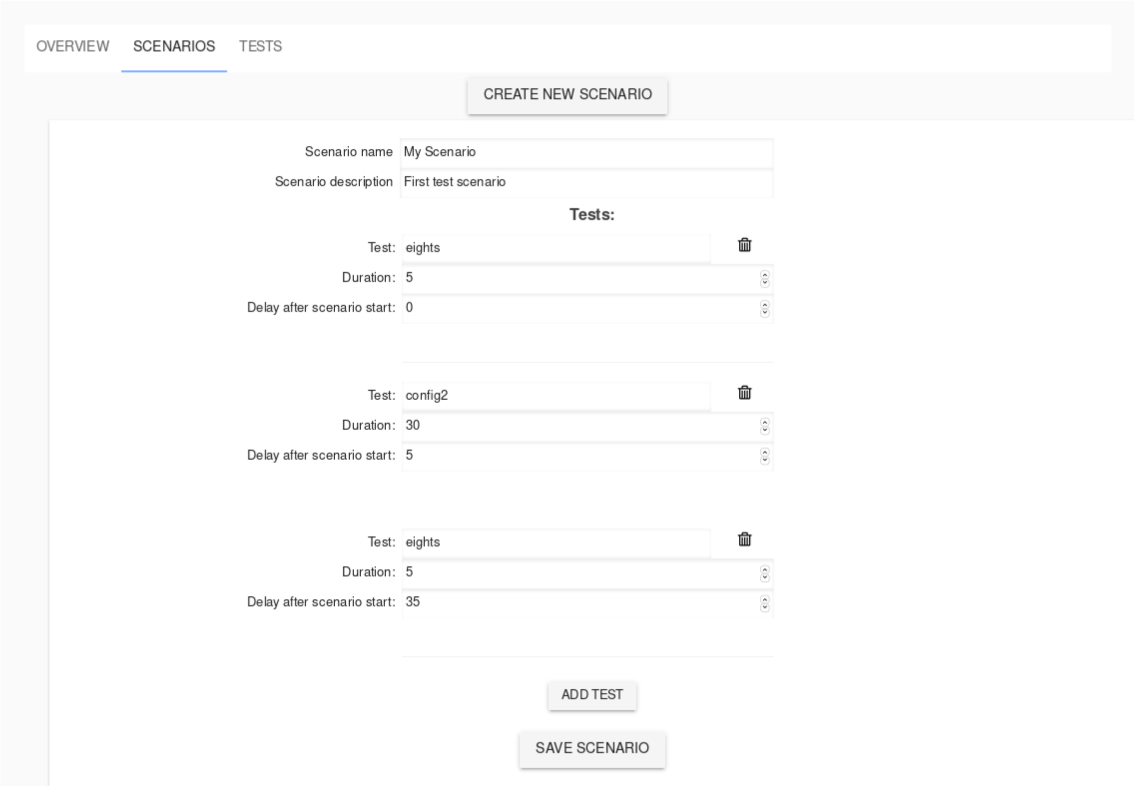

Figure 4.7. Screenshot of form for creating test scenario

Same forms are also used for editing selected items.

4.5 Backend integration

In order to provide communication between different processes, applications and daemons, there was developed a project called `ubus`. Basically, it is a message broker that is integrated into the operational system. The core of ubus is `ubusd` - daemon that provides an interface for registering listeners via Unix sockets and sending messages. Ubus also has a library called `libubus` that can be used in applications for interacting with ubus.

Every daemon registers a unique namespace and set of paths under that namespace in ubus. Every path provides one or more procedures with a predefined list of arguments. Procedure can also return a message after its execution.

F-Tester has its own namespace in ubus:

```
root@F-Tester:~# ubus list
    block
     flowtester
     flowtester_ctl  #            <-- this one
     ftexec
     ftrm
     log
     network
     network.device
     network.interface
     network.interface.loopback
     network.interface.mgmt
     network.interface.net
     network.wireless
     service
     session
     system
     uci
```

List of its procedures can be retrieved this way:

```
root@F-Tester:~# ubus -v list flowtester
'flowtester' @fe68c873
    'flowtester_ctl' @54242849
    "schedule":{}
    "cancel":{"name":"String"}
    "result_list":{}
    "version":{}
    "config_add":{"data":"String"}
    "config_list":{}
    "check_mtu":{"ip":"String"}
    "status":{}
    "result_detail":{"name":"String","results":"String"}
    "config_delete":{"name":"String"}
    "result_delete":{"name":"String"}
```

To execute scenario or plan next scenario execution Test Planner sends it in JSON format encoded with base64 to the procedure `config_add`.

4.6 Implementation Issues

There were some issues during the process of project implementations that were caused by the specificity of platform and technologies.

- Bootstrap

  It was planned to use Bootstrap as the main UI library on the frontend for elements rendering. During its application, it was found that LuCI's GUI is also built with Bootstrap, but with an older version, which is incompatible with the actual one. After loading all content on the page, new Bootstraps overrides old CSS rules and by this breaks LuCI's navigation panel, what makes impossible to go ta any other section of GUI. Finally, another library was chosen to use to prevent conflicts of rules for already existing elements on the page.
- Cascade operations

  Test Planner is prepared for cascade editing - if any test configuration is changed, then the test scenario, where this configuration is used, will produce a test with changed data. But if the same situation will happen with delete operation, then generating test from scenario will finish with an error. Solution is described in section 7.2

Chapter 5
Testing

As the last phase of development process test Test Planner was tested with different sets of data. In both test scenarios application worked properly and ended with success.

5.1 Test data

For this kind of data program `ping` was selected because firstly, it is present in almost every OS (including OpenWRT), and secondly, it has all kind of parameters that are used in Data Transformation component of Test Planner. For testing with `ping` there were created 2 test configurations:

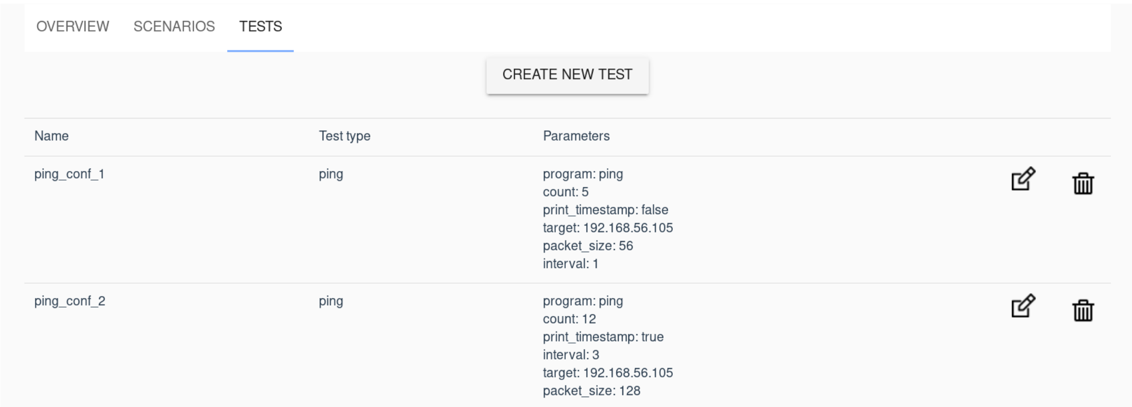

Figure 5.1. Screenshot of test configuration

and two test scenarios:

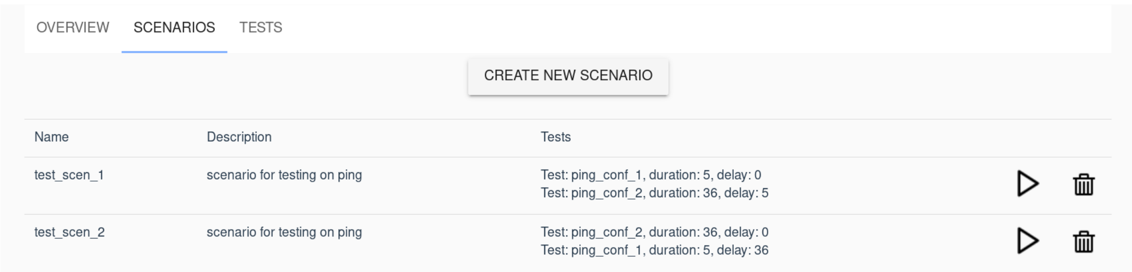

Figure 5.2. Screenshot of test scenarios

5.2 Production data

As the example of production data programs `iperf3` and `flowping` were selected as the ones that are used for network measurements by F-Tester.

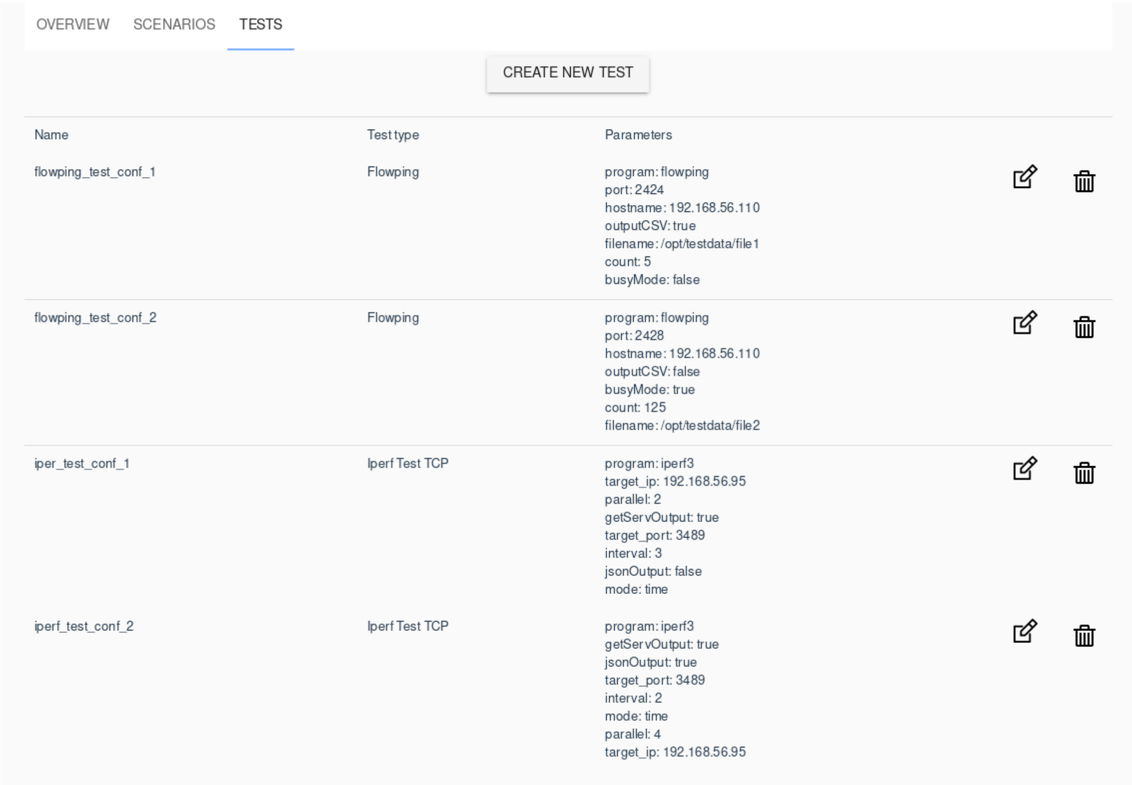

Figure 5.3. Screenshot of test configurations

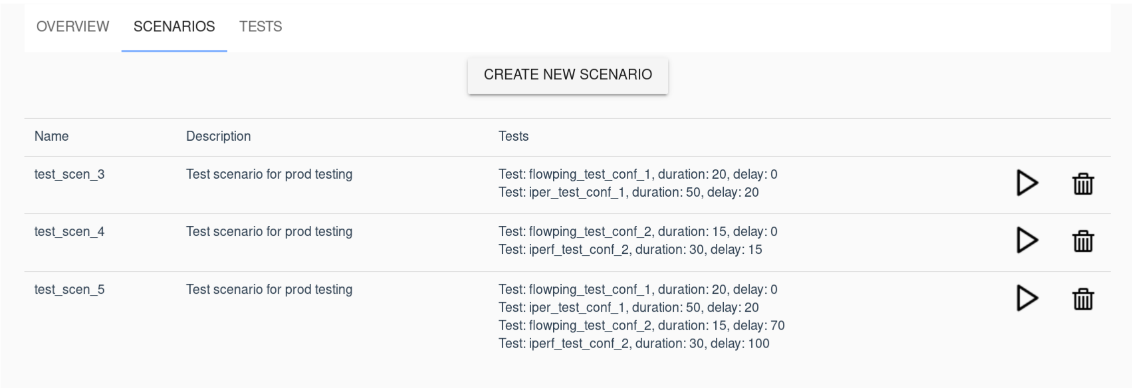

Figure 5.4. Screenshot of test scenarios

Chapter 6
Installation & Deployment guide

6.1 Prerequisites

- Installed `Lua` interpreter
- Installed `LuCI` interface
- Installed `ubus` daemon & library

Test Planner was designed and implemented to be used only as the part of F-Tester, but if any system fits all the conditions, it also can be potentially used there.

In order to make some changes in the frontend part user has to edit source files on a normal computer, build it with `NPM` and then use them normally as it is described in next section.

6.2 Manual installation

Installation steps:

1. Backend - Create endpoints and their handlers
   Create folder `/usr/lib/lua/luci/controller/Test-Planner` and move there all files from *archive*`/backend` folder
2. Frontend - Create static view
   Create folder `/usr/lib/lua/luci/view/Test-Planner/` and copy there *archive*`/frontend/dist/index-vue.htm` file
3. Frontend - Add JavaScript, CSS and images
   Create folder `/www/luci-static/resources/TestPlanner/vue_dist` and copy there all folders *archive*`/frontend/dist/static/`

Note: *archive* is the attached to this thesis archive with files.

Chapter 7
Conclusion

7.1 Project goals fulfilment

- Review F-Lab and F-Tester platforms
  Description of F-Lab, F-Tester and main technologies that are used in F-Tester is presented in section 2.2
- Analyse way of development and integration of custom modules for F-Tester
  There is an officially supported and well-documented way of extending default LuCI's functionality described in section 2.3. Also, section 4.5 contains information about communication between different applications and processes via the default system broker.
- Define requirements for Test Planner
  Functional and non-functional requirements, defined basing on expected functionality of Test Planner, as well as use-case diagram, are mentioned in section 3.5.
- Choose suitable technologies and implement Test Planner module
  Chapter 4 is dedicated to the implementation part of this thesis - planning architecture, selecting technologies and programming.
- Test the module on specified use cases
  The process of testing Test Planner on test data and production data ended with success. The process and its results described in chapter 5

As all the project goals are entirely or at least for the most part fulfilled, the project can be considered a success.

7.2 Future development

As the part of future releases next features are planned to be implemented:

- Overview tab
  Dashboard with history of used scenarios, list of planned scenarios execution in future and graph of network load predicted by parameters of scenarios.
- Extending functionality of data transformation module
  Add more possible data types and values for test schemas, for example, nested objects.
- Reaction on delete of removal test configuration
  Add some kind of warning alert when the user tries to delete the concrete test configuration that is used in at least one test scenario, and notify him that such operation leads to breaking that scenario.

Chapter 8
Appendix

8.1 Contents of attached archive

```
/ ..................................................... Root directory
├── src ................................................ Folder with source files
│   ├── controller
│   │   ├── how_to_create_test_type .............. Manual for creating test types
│   │   ├── module.lua ................................ Script with controller
│   │   ├── scenarios.json ........................ Test scenarios storage
│   │   ├── testTypes.json ............................ Test types storage
│   │   └── tests.json ....................... Test configurations storage
│   └── frontend
│       ├── index-vue.htm ................................ Static view file
│       ├── dist
│       │   └── static .............. Folder with compiled frontend components
│       └── sources .............. Folder with sources of frontend components
└── thesis
    ├── thesis.pdf .............................................. This file
    ├── thesis.tex ................................. TeX source of this file
    └── glossary.tex ....................... TeX source of glossary section
```

8.2 Abbreviations

API	■	Application programming interface
GUI	■	Graphical user interface
JS	■	JavaScript
JSON	■	JavaScript Object Notation
NGA	■	Next-Generation Access
NGN	■	Next-Generation Network
OS	■	Operating system
Regex	■	Regular expression
RFC	■	Requests for comments - formal document drafted by the Internet Engineering Task Force that describes the specifications for a particular technology
SJSON	■	Special JSON - dialect of JSON that was developed by author of this work
UI	■	User interface

References

[1] MAKARENKO, Sergey Ivanovich, Nikolai Nikolaevich CHALENKO and Aleksei Gennad'evich KRYLOV. Next Generation Networks. Systems of Control, Communication and Security. 2016, 2016(1), 84-85. ISSN 2410-9916.

[2] Traffic Flow Measurement: Architecture: RFC 2722 [online]. IETF, 1999 [cit. 2019-05-06]. Available from:
`https://tools.ietf.org/html/rfc2722`

[3] IPv6 Flow Label Specification: RFC 3697 [online]. IETF, 2004 [cit. 2019-05-06]. Available here:
`https://tools.ietf.org/html/rfc3697`

[4] Requirements for IP Flow Information Export (IPFIX): RFC 3917 [online]. IETF, 2004 [cit. 2019-05-06]. Available from:
`https://tools.ietf.org/html/rfc3917`

[5] SCOTT, Emmit A. SPA design and architecture: understanding single-page web applications. Shelter Island, NY: Manning, 2016. ISBN 16-172-9243-5