

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kelpe** Jméno: **Artem** Osobní číslo: **437859**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávací 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:

System for creating scenarios for F-Tester platform

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: **07.01.2020**

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



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of computers science**

Bachelor's Thesis

F-Tester Platform Scenario Creation System

Artem Kelpé

Software Engineering and Technology

January 2020

Supervisor: Ing. Zbyněk Kocur, Ph.D.

/ 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, January 06, 2020

.....

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

1 Introduction	1
1.1 Project goals	1
2 Background	3
2.1 Network flow	3
2.2 F-Tester Platform.....	3
2.2.1 Technologies.....	3
2.3 Integration with F-Tester.....	4
3 Design	5
3.1 Existing solutions	5
3.2 Requirements.....	5
3.2.1 Functional requirements ...	5
3.2.2 Non-functional re- quirements	5
3.3 Use-case diagram	6
3.4 Frontend design requirements ...	6
4 Implementation	9
4.1 Application architecture	9
4.2 Technology stack.....	10
4.2.1 Backend	10
4.2.2 Frontend.....	10
4.3 Data transformation	10
4.4 Graphical User Interface.....	14
4.5 Backend integration.....	17
4.6 Implementation Issues	18
5 Testing	19
5.1 Test data	19
5.2 Production data.....	20
5.3 Usability testing	20
5.3.1 Target group	21
5.3.2 Conditions of testing	21
5.3.3 Preparation	21
5.3.4 Answers of participants..	24
5.3.5 Analyse of results	25
6 Installation & Deployment guide	27
6.1 Prerequisites	27
6.2 Manual installation	27
7 Conclusion	29
7.1 Project goals fulfilment	29
7.2 Future development.....	29
8 Appendix	31
8.1 Contents of attached archive ..	31
8.2 Abbreviations	32
References	33

Tables /

4.1. Decision table for frontend frameworks	10
--	----

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

3.1 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.

3.2 Requirements

3.2.1 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

3.2.2 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 \geq 72, Firefox \geq 65, Safari \geq 12, Opera \geq 58, Edge \geq 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

3.3 Use-case diagram

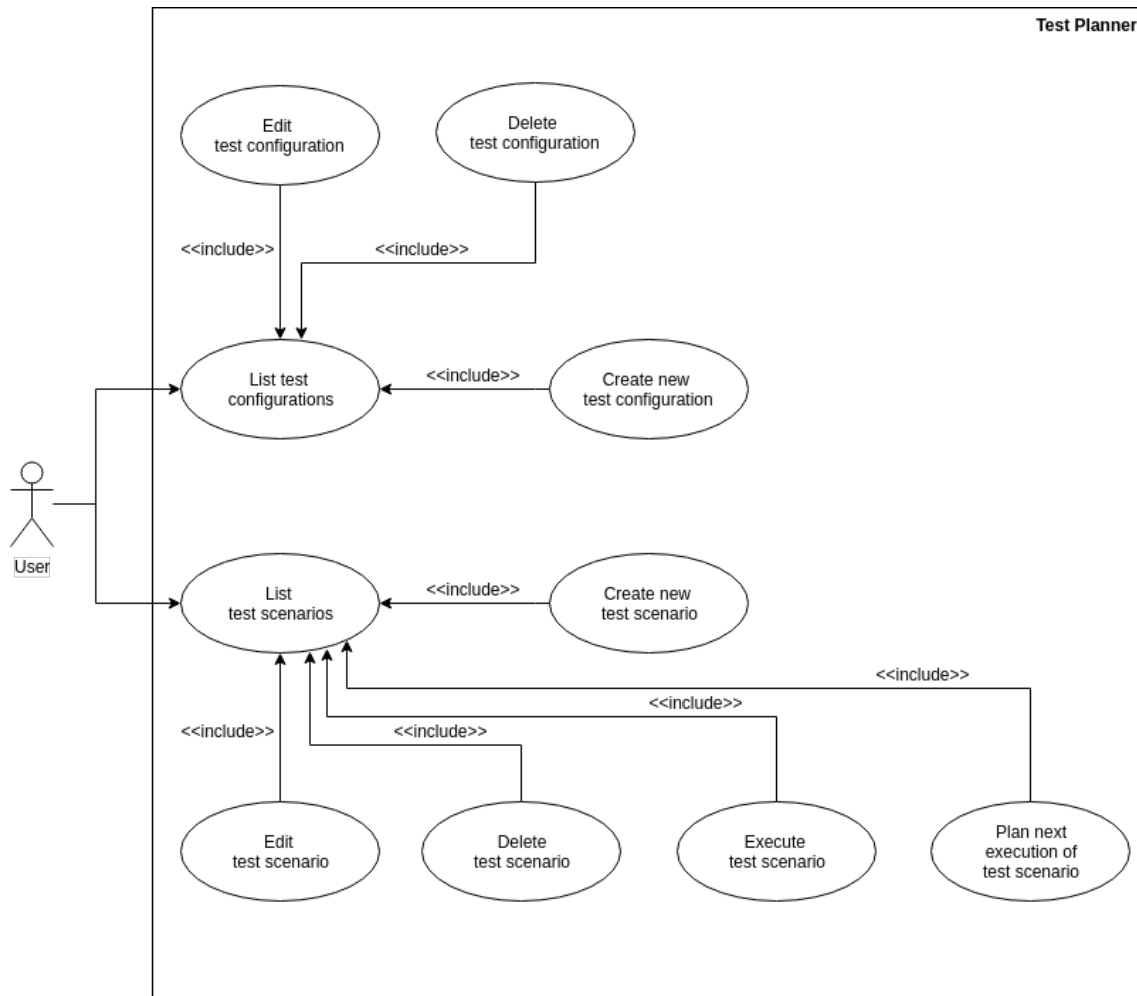


Figure 3.1. Schematic description of application architecture

3.4 Frontend design requirements

As Test Planner tool is planned to be a replacement of manual type of creating scenarios in the console, it has to be user-friendly and easy to use system - these terms are usually used as synonyms of UX and UI. UX or user experience is the process of manipulating of user behaviour that covers all aspects of the person's experience with the system including industrial design graphics, the interface, the physical and the manual interaction.[5] UI or user interface is the part of human-computer interaction. Generally the goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals. One of the main tasks of this project is to implement and test frontend part of Test Planner to be simple, convenient to use and similar to the other modules of F-Tester. The whole style of system is inspired by LuCI default minimalistic design, so for implementing frontend of Test Planner was chosen Bootstrap library, which provides different ready to use components. Design is

described in detail in section 4.4, more information about usability testing can be found in section 5.3

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:

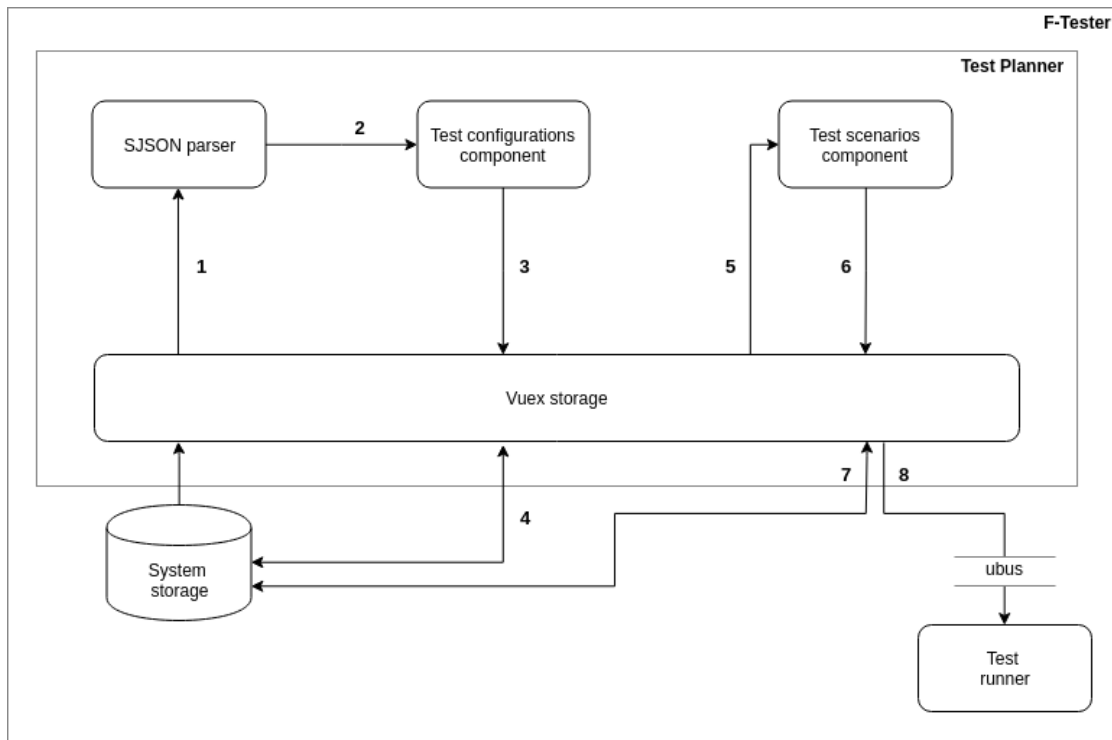


Figure 4.1. Schematic description of application architecture

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 SJJSON will look like this:

CREATE NEW TEST

Test type:

Test configuration name

Program

Target host

Count of icmp packets to send

Interval between send packets [s]

Packet size [bytes]

SAVE TEST

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

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 [6]. 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

OVERVIEW SCENARIOS TESTS

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.





Name	Test type	Parameters	
config2	Iperf Test TCP	program: iperf3 smth-P: 14 smth-numParam: 6 target_ip: 10.0.11.15 getServOutput: false target_port: 5190 mode: time smth-J: true smth-i: true	 
eights	ping	program: ping count: 5 print_timestamp: false target: 8.8.8.8 packet_size: 56 interval: 1	 

Figure 4.4. Screenshot of test configurations preview

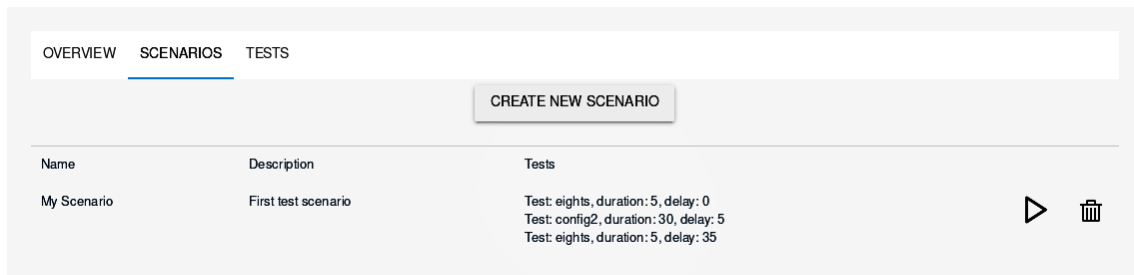


Figure 4.5. Screenshot of test scenarios preview

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.

The screenshot shows a web interface with three tabs: OVERVIEW, SCENARIOS, and TESTS. The TESTS tab is active. A button labeled 'CREATE NEW TEST' is positioned above a form. The form contains the following fields:

- Test type: ping
- Test configuration name: eights
- Program: ping
- Target host: 8.8.8.8
- Count of icmp packets to send: 5
- Interval between send packets [s]: 1
- Packet size [bytes]: 56
- Print timestamp at the begin of line:

A 'SAVE TEST' button is located at the bottom of the form.

Figure 4.6. Screenshot of form for creating test configuration

OVERVIEW SCENARIOS TESTS

CREATE NEW SCENARIO

Scenario name: My Scenario

Scenario description: First test scenario

Tests:

Test: eights

Duration: 5

Delay after scenario start: 0

Test: config2

Duration: 30

Delay after scenario start: 5

Test: eights

Duration: 5

Delay after scenario start: 35

ADD TEST

SAVE SCENARIO

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 to 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

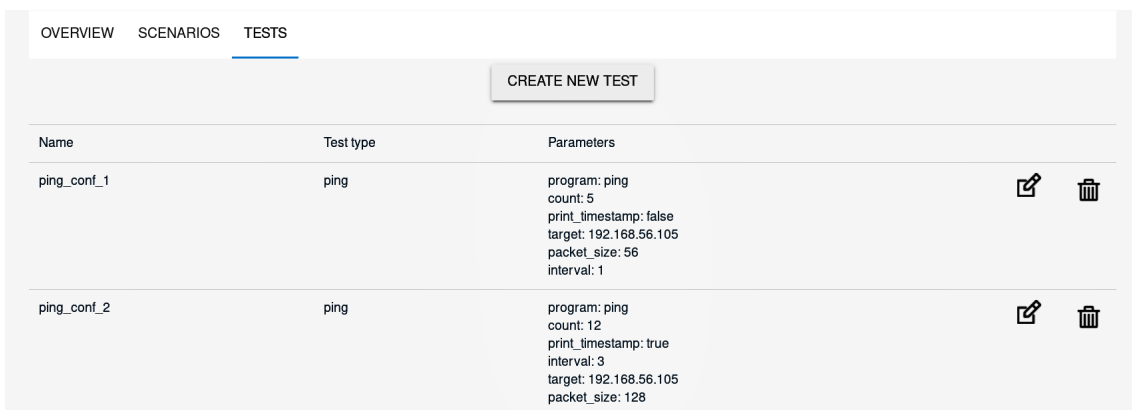
Chapter 5

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:

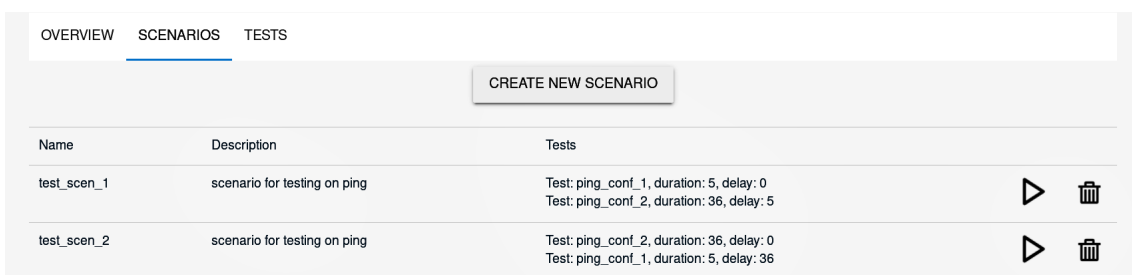


The screenshot shows the 'TESTS' tab in the Test Planner interface. At the top, there are tabs for 'OVERVIEW', 'SCENARIOS', and 'TESTS'. Below the tabs is a 'CREATE NEW TEST' button. The main content is a table with three columns: 'Name', 'Test type', and 'Parameters'. There are two rows of test configurations, each with edit and delete icons on the right.

Name	Test type	Parameters		
ping_conf_1	ping	program: ping count: 5 print_timestamp: false target: 192.168.56.105 packet_size: 56 interval: 1		
ping_conf_2	ping	program: ping count: 12 print_timestamp: true interval: 3 target: 192.168.56.105 packet_size: 128		

Figure 5.1. Screenshot of test configuration

and two test scenarios:



The screenshot shows the 'SCENARIOS' tab in the Test Planner interface. At the top, there are tabs for 'OVERVIEW', 'SCENARIOS', and 'TESTS'. Below the tabs is a 'CREATE NEW SCENARIO' button. The main content is a table with three columns: 'Name', 'Description', and 'Tests'. There are two rows of test scenarios, each with play and delete icons on the right.

Name	Description	Tests		
test_scen_1	scenario for testing on ping	Test: ping_conf_1, duration: 5, delay: 0 Test: ping_conf_2, duration: 36, delay: 5		
test_scen_2	scenario for testing on ping	Test: ping_conf_2, duration: 36, delay: 0 Test: ping_conf_1, duration: 5, delay: 36		

Figure 5.2. Screenshot of test scenarios

5.2 Production data

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

Name	Test type	Parameters		
flowping_test_conf_1	Flowping	program: flowping port: 2424 hostname: 192.168.56.110 outputCSV: true filename: /opt/testdata/file1 count: 5 busyMode: false		
flowping_test_conf_2	Flowping	program: flowping port: 2428 hostname: 192.168.56.110 outputCSV: false busyMode: true count: 125 filename: /opt/testdata/file2		
iper_test_conf_1	iperf Test TCP	program: iperf3 target_ip: 192.168.56.95 parallel: 2 getServOutput: true target_port: 3489 interval: 3 jsonOutput: false mode: time		
iper_test_conf_2	iperf Test TCP	program: iperf3 getServOutput: true jsonOutput: true target_port: 3489 interval: 2 mode: time parallel: 4 target_ip: 192.168.56.95		

Figure 5.3. Screenshot of test configurations

Name	Description	Tests		
test_scen_3	Test scenario for prod testing	Test: flowping_test_conf_1, duration: 20, delay: 0 Test: iper_test_conf_1, duration: 50, delay: 20		
test_scen_4	Test scenario for prod testing	Test: flowping_test_conf_2, duration: 15, delay: 0 Test: iper_test_conf_2, duration: 30, delay: 15		
test_scen_5	Test scenario for prod testing	Test: flowping_test_conf_1, duration: 20, delay: 0 Test: iper_test_conf_1, duration: 50, delay: 20 Test: flowping_test_conf_2, duration: 15, delay: 70 Test: iper_test_conf_2, duration: 30, delay: 100		

Figure 5.4. Screenshot of test scenarios

5.3 Usability testing

Usability testing is a special technique that allows to evaluate the convenience of usability of the system. During usability testing system is presented to the participants, then they try to operate with it and finally participants fill the special form, where they describe their experience with the system.

5.3.1 Target group

Before the testing there was predefined target group basing on potential user of system: people of age 20-40 years with technical (IT) education or experienced in the IT for at least 2 years. There are no limitations on gender.

5.3.2 Conditions of testing

Testing was conducted in a special Usability Lab - special room that is designed for such kind of testing. Lab is divided into two parts - for the participant and the moderator. The participant has cards with tasks that he has to perform on the computer while moderator is observing the whole process from the beginning until the end. Moderator also controls the emotions on the participant's face as they also can influence test results. If the participants meets some problems during task execution, he is able to communicate with the moderator - otherwise communication between the sides is not recommended.

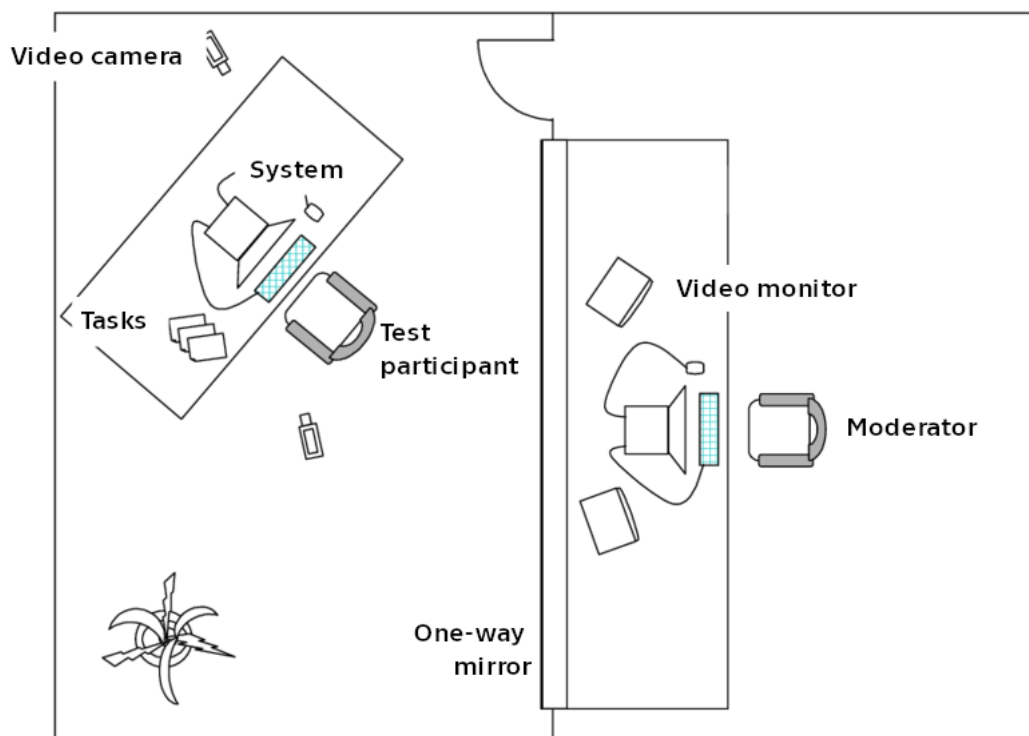


Figure 5.5. Schema of usability lab

Source: <https://www.researchgate.net>

5.3.3 Preparation

Before starting of test, each participant was introduced to the brief description of the system and the whole infrastructure. Moreover, each participant was acquainted with the terminology that is used in the system by next glossary:

```
■ Test configuration - list of parameters that will be used for the
execution of concrete program. Examples of test configurations:
    config_name1: ping -I eth0 -i 2 -s 32 -W 5 192.168.0.5
or
    config_name2: nslookup -query=hinfo -timeout=10 intranet.com

■ Test scenario - list of test configurations in predefined sequence
with time intervals. When test scenario is executed, system starts to
run each application defined in test configuration with configured
parameters and pauses between test configurations. Examples of test
scenarios:
    SCENARIO START
    pause 5sec
    config_name1
    pause 5 sec
    config_name2
    SCENARIO END
or
    SCENARIO START
    config_name3
    pause 20sec
    config_name2
    pause 15sec
    config_name5
    pause 5sec
    SCENARIO END

■ F-Tester - module that hosts Test Planner system and executes test
scenarios
```

All participants had the same list of tasks:

- Task 1 - Create test configuration for application ping with next parameters:
Test configuration name: ping_config_1
Target host: 1.1.1.1
Count of packets: 10
Interval between packets: 2 seconds
Packet size: 56 bytes
Print timestamp on line start: No
- Task 2 - Create test configuration for application ping with next parameters:
Test configuration name: ping_config_2
Target host: internal_nas
Count of packets: 8

Interval between packets: 1 seconds
 Packet size: 32 bytes
 Print timestamp on line start: Yes

- Task 3 - Create test configuration for application `flowping` with next parameters:

Test configuration name: `flowping_config_1`
 Target host: 10.0.20.105
 Target port: 1277
 Count of packets: 20
 Output CSV: Yes
 Busy mode: Yes
 Filename: `/opt/testdata/data.txt`

- Task 4 - Create test configuration for application `flowping` with next parameters:

Test configuration name: `flowping_config_2`
 Target host: 10.0.20.171
 Target port: 8085
 Count of packets: 10
 Output CSV: No
 Busy mode: Yes
 Filename: `/opt/testdata/data.txt`

- Task 5 - Create test scenario with next parameters:

Test scenario name: `Scenario_1`
 Sequence of test configurations:

- ping_config_1, duration: 30, delay: 5
- ping_config_2, duration: 15, delay: 5
- flowping_config_1, duration: 25, delay: 10
- flowping_config_2, duration: 25, delay: 10

- Task 6 - Create test scenario with next parameters:

Test scenario name: `Scenario_2`
 Sequence of test configurations:

- flowping_config_1, duration: 10, delay: 1
- ping_config_2, duration: 30, delay: 10
- flowping_config_2, duration: 20, delay: 10

Card for each task contained not only task description, but also some fields where participant wrote about their experience during the task:

```
[ ... task description ... ]
```

Select an option:

- There was no problem during test
- There was no problem during test, but ...
- There were some problems during test: ...
- I was unable to finish the test because ...

Final card contained next questions:

If you have any comments or remarks, please, write them here: ...
Rate the whole experience of your usage of system from 1 (unable to use)
to 10 (nothing can be improved, system is perfect)

■ 5.3.4 Answers of participants

- Participant 1 - man, 21 years old, bachelor student of CTU in Prague, FIT, currently works as Junior Tester.

Task 1: There was no problem during test.

Task 2: There was no problem during test.

Task 3: There was no problem during test, but some fields are not very understandable - it can be a good idea to add description to the parameters.

Task 4: There was no problem during test, but some fields are not very understandable - it can be a good idea to add description to the parameters.

Task 5: There was no problem during test.

Task 6: There was no problem during test.

Notes, remarks: System seems to be user-friendly and easy-to-use, but there are some things that can be improved.

Grade: 8

- Participant 2 - man, 28 years old, no special education, currently works as Network Engineer.

Task 1: There was no problem during test.

Task 2: There was no problem during test.

Task 3: There was no problem during test, but units could be written on the right side of the input box.

Task 4: There was no problem during test, but units could be written on the right side of the input box.

Task 5: There was no problem during test.

Task 6: There was no problem during test.

Notes, remarks: Everything went without any problem, system is ready to be used in production.

Grade: 9

- Participant 3 - woman, 24 years old, graduated bachelor study of CTU in Prague, FEL, currently works as Freelance programmer.

Task 1: There was no problem during test.

Task 2: There was no problem during test.

Task 3: There was no problem during test.

Task 4: There was no problem during test.

Task 5: There was no problem during test.

Task 6: There was no problem during test.

Notes, remarks: Test Planner has minimalistic design, but it seems to be an advantage, as the whole system has almost the same style and design. Anyway, I met no problems during the test.

Grade: 8

- Participant 4 - woman, 29 years old, graduated master study of CTU in Prague, FIT, currently works as Programmer of embedded systems.
 - Task 1: There was no problem during test.
 - Task 2: There was no problem during test.
 - Task 3: There was no problem during test.
 - Task 4: There was no problem during test.
 - Task 5: There was no problem during test.
 - Task 6: There was no problem during test.
 - Notes, remarks: Every task went without any problem.
 - Grade: 9

- Participant 5 - man, 32 years old, no special education, currently works as Frontend programmer.
 - Task 1: There was no problem during test.
 - Task 2: There was no problem during test.
 - Task 3: There was no problem during test.
 - Task 4: There was no problem during test.
 - Task 5: There was no problem during test.
 - Task 6: There was no problem during test.
 - Notes, remarks: System looks good, there is validation for preventing errors, but the system does not look like modern web page, there could be implemented more things to improve UX.
 - Grade: 7

■ 5.3.5 Analyse of results

All test participants were able to pass all tests without any problems. Average grade mark is 8.2, so the system can be claimed as user-friendly and easy to use system with understandable frontend. Though some participants mentioned that there are some elements that can be improved for better user experience, all participants said that they are ready to use such a system in their everyday work routine.

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.

This instruction is written for precompiled frontend project, which is located in the *dist* folder. User can manually compile frontend sources file. In order to compile the project, next commands should be executed in the console while locating in the *frontend* folder, which contains Vue-components, assets, configurational files and needed modules and libraries:

```
npm install
npm build
```

After their execution there will be compiled sources in the *dist* folder. User must be sure to have enough space for downloading dependency modules and packages needed

for compiling process. More info about compiling process via NPM can be found on official documentation here: <https://docs.npmjs.com/>

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 3.2.
- 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 and usability testing
The process of testing Test Planner on test data and production data ended with success. The process and its results described in chapter 5. Another part of testing - Usability testing - is described in detail in section 5.3.

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.

- UI improvements

Participants of the Usability testing mentioned about possible improvements of some frontend elements such as using timeline graph, dynamically showing and hiding of some data etc.

- Tool for creating SJSON files

Such a tool was not a part of the task of this bachelor thesis, but can be a useful part of system and facilitate work routine of the administrators of F-Tester and F-Lab platform.

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 document
thesis.tex	TeX source of this document
glossary.tex	TeX source of glossary section
Technika.pfb	Font for compiling this document
ctustyle2.tex	TeX template that is used in this document

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
UX	■	User experience



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] Merholz, Peter Peter in Conversation with Don Norman About UX and Innovation. Adaptive Path, 2007
- [6] SCOTT, Emmit A. SPA design and architecture: understanding single-page web applications. Shelter Island, NY: Manning, 2016. ISBN 16-172-9243-5