TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Daniel Smirnov 164056IAPB

THE DEVELOPMENT OF WEB-BASED GRAPHICAL USER INTERFACE FOR THE ORIGINAL EQUIPMENT MANUFACTURER (LDI INNOVATION)

Bachelor’s thesis

Supervisor: Innokenti Sobolev
Ph.D,
Eduard Petlenkov
Professor

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Daniel Smirnov 164056IAPB

VEEBIPÕHISE GRAAFILISE KASTUAJALIIDESE ARENDUS ORIGINAALSEADMETE TOOTJALE (LDI INNOVATION)

bakalaurusetöö

Juhendaja:	Innokenti Sobolev
	Ph.D,
	Eduard Petlenkov
	Professor

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Daniel Smirnov

20.05.2019

Abstract

The main goal of this thesis is to develop a web-based graphical user interface for SFS Cube™ instrument [1] developed and manufactured by LDI Innovation. LDI Innovation is a company that develops innovative new products and services based on accumulated knowledge, experience and intellectual property of the founders. The company provides various Engeneering services including embedded and PC (Personal Computer) software development for control, real-time asquisition and data processing.

The SFS Cube™ is designed as a compact, fast scanning spectrometer to operate in laboratory and in industrial environment based on the measurement of the Spectral Fluorescence Signature (SFS) of liquid, powder or solid samples [1]. The PC side software for SFS Cube must be able to communicate with the device and to visualize the information received from the SFS Cube™.

The aim of this project is to develop the convenient user interface for communication, configuration and management of the device. As a result, an appropriate solution has been found and implemented. A working web-based user interface has been presented. The developed graphical user interface is currently in use by customer.

This thesis is written in English and contains 34 pages of text, including 6 chapters, 25 figures.

Annotatsioon

Veebipõhise graafilise kastuajaliidese arendus originaalseadmete tootjale (LDI Innovation)

Käesoleva bakalaureusetöö peamiseks eesmärgiks on arendada veebipõhine graafiline kasutajaliides SFS Cube™ seadmele, mida arendab ja toodab LDI Innovation OÜ. LDI Innovation OÜ on Eesti ettevõte, mis arendab innovatiivseid uusi seadmeid ja teenuseid, mis põhinevad ettevõtte asutajate aastakümnete pikkusel teadustööl ning tekkinud kogemusel ja intellektuaalomandil. Ettevõte pakub erinevaid insenerlahendusi, mille hulka kuuluvad sisseehitatud tarkvara ning arvutitarkvara arendamine seadmete juhtimiseks, reaalajaliseks andmete hõiveks ning andmetöötluseks.

SFS Cube™ seade on konstrueeritud kompaktse ning kiire spektrofluoromeetrina kasutamiseks laboratooriumis ning tööstuslikus keskkonnas. Seade põhineb vedelate, tahkete ning pulbriliste proovide Spektraalsete Fluorestsentsi Sõrmejälgede (SFS) mõõtmise meetodil. Seadme arvutitarkvara peab olema võimeline seadmega kahesuunaliselt andmeid vahetama ning SFS Cube'ilt saadud andmeid visualiseerima.

Käesoleva projekti eesmärgiks oli arendada mugav kasutajaliides seadmega kommunikeerimiseks, seadme konfigureerimiseks ning selle haldamiseks, pidades arendusel silmas, et graafilist kasutajaliidest oleks ettevõttel tulevikus võimalik lihtsasti ja modulaarselt uuendada.

Käesolevas töös selgitatakse, kuidas *WebSocket*'i abil toimub interaktiivsete andmete visualiseerimine. Arendamiseks on kasutatud vahendeid nagu *IDE PyCharm, Chrome DevTools* ja *Git*. Peamiseks keeleks oli valitud Python. *Front-end*'i jaoks kasutati CSS, HTML, JavaScript ning Flask tehnoloogiaid. Töö käigus disainiti kasutajaliides.

Arendustegevuse tulemusel on leitud ja disainitud sobilik lahendus ning töötavat veebipõhist kasutajaliidest on käesolevas töös esitletud. Välja arendatud graafiline kasutajaliides on juba reaalselt kasutuses.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 34 leheküljel, 6 peatükki, 25 joonist.

List of abbreviations and terms

C++	Programming language
Callback	Executable code that is passing argument to other code
CLI	Command Line Interpreter
Cookies	Piece of data that is stored on a user side
CSS	Cascading Style Sheets
DevTools	Development tools
Google Chrome	Cross-platform web browser developed by Google
HTML	HyperText Markup Language
IDE PyCharm	Integrated development environment
JavaScript	The programming language of HTML and Web
JSON	Lightweight data-interchange format
OEM	Original equipment manufacturer
PC	Personal computer
Python	Programming language
RWD	Responsive web design [2]
SFS	Spectral Fluorescence Signature Analyzer
WebSocket	Connection between browser and server. Used for data exchange

Table of contents

List of figures

1 Introduction

The main purpose of the thesis was to develop a web-based user interface for the OEM (Original Equipment Manufacturer) LDI Innovation. User Interface gives interactions between the users and machines. To develop the user-friendly interface, we must understand and see from the different ways how the current program will be used and for which kind of people it is specialized. Nowadays, LDI-Innovation develops and manufactures the unique spectral analyzers based on the technology of Spectral Fluorescence Signatures (SFS). The devices have the interface that was initially developed in the year 2005. It uses the framework that is written in C++ language for device control, data acquisition, analysis and visualization. While the interface has been substantially upgraded since that, its methodology is already obsolete. Comparing with currently available software solutions this interface does not match with the up-to-date requirements of the company. Also, previously made interface could only work using the executable file, so there is no opportunity to open it using installed in every computer browser. Technologies are going on and the actuality of web-based interfaces rises every day.

2 About the company

LDI Innovation is a company that keeps focus on innovative developments of new products and services based on accumulated knowledges, experience and intellectual property of the founders. It has a rich historical background in innovative technologies of developing sensors that detect the organic pollution in water and on the ground, pollution identification in liquid, powder or solid samples and remote pollution detection and identification.

Company is operating with novel Photonics, Information and Communication technologies in industrial, environmental and security domains. Products which are developed by the company includes photonics sensors and systems providing real-time and online sensing solutions for industries, environmental protection, medicine and bio-chemical security and safety.

Historically, all software developments in the company were done on C++ language. Current products in company have the interface that was developed using framework that is written on C++ language. The Python language was chosen for the further developments and upgrades of the products.

The company wants to use novel technologies of software development. Comparing with currently available software solutions the interface does not match the requirements of the company, such as easy integrative software, independent from the executable windows file and resizable, customizable GUI design. Nowadays, the company is focused on using technologies that will allow in future integrating developed user interface to other products.

3 Requirements

3.1 Company's requirements to the user interface

Product in this project is modern device for the spectral fluorescence signature analyzer named SFS-Cube™. Company's main requirement was to develop web-based graphical user interface for the application that would make easier for the user to communicate with the device. The design must be understandable, intuitive and at the same time must contain enough information about the current state of the SFS Cube™ and about the measurements.

The current project was to develop a web-based user interface that should be easily updated and integrated into other projects of the company. The developed application should allow creating further projects on its base. The methodology of development was chosen mutually with the head of company developers.

Requirements:

- Develop a web-based user interface by using nowadays technologies
- Use technologies that allow easily to integrate the developed user interface into the other projects of the company
- Develop a design that can be easily updated and must be intuitive
- Use Python language for development along with CSS (Cascading Style Sheets), HTML (HyperText Markup Language) and JavaScript

3.1.1 Functional requirements

- Real-time device operation control and status visualization
- Convenient measurement data visualization with customizable plots
- Resizable and customizable GUI layout

3.1.2 Non-functional requirements

- Multiplatform web-based interface. User interface should work in such systems as: Linux, Macintosh, Windows

4 State of the art

4.1 Possible solutions

This paragraph describes the requirements suggested by the company and some of the possible approaches are reviewed.

4.1.1 Web development

Nowadays, the popularity of Web developments is rising very fast. The usage of web-based interfaces allows to avoid the necessity of installation of the big sized files on the computer. A web-based interface is compatible with any device's platform and the only requirement is to have the appropriate browser. Being able to work on the Web-related code elements and systems became a very important skill for the up-to-date development of the software products. The benefit of web-based development is that there is a huge selection of tools, frameworks and extensions, that can be used [3].

Web Applications are dynamic web sites combined with server-side programming which provide user interactions, connecting to them to backend databases and generating results to browser [4].

It was decided to implement the application in the form of a web site, since it contributes to the ease of portability and accessibility for users, because of the widespread use of web browsers.

4.1.2 Python as programming language

The Python language was chosen, because it is an interpreted, interactive, object-oriented programming language. It provides high-level data structures such as list and associative arrays (called dictionaries), dynamic typing and dynamic binding, modules, classes, exceptions, automatic memory management. It has a remarkably simple and elegant syntax and yet is a powerful and general-purpose programming language [5]. Python has a very extensive set of machine learning libraries [7]. The compilation is not needed because of its dynamically-typed nature and reuse of code written in other languages is possible. For these reasons and due to the abundance of well written, well documented, scientific Python libraries, Python has become very popular in scientific communities [8].

The development on Python takes less time than development on programming language C++. The reasons why Python development takes less time are:

- Supports Garbage Collection. When objects are not needed Python automatically reclaims memory from them
- Runnable through interpreter
- Dynamically typed language
- Language comes with a massive set on inbuilt standard libraries [6]

Python is selected as a core language in SFS Cube™ software project. It has several strong web-development frameworks such as Flask, thus it is the best candidate for frontend software part development as well.

4.1.3 Library choice

The main problem of choosing the framework was to find the most suitable libraries that will cooperate with backend and at the same time will not restrict both parts of the project. As a result, Flask web framework was chosen for the user interface. Python library – Bokeh was chosen for generating plots, which allow for the rendering of the most important types of graph and they are all rendered in line with the HTML web standard.

4.1.4 Bokeh

Bokeh is an interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of versatile graphics, and to extend this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications [9]. In current project Bokeh is used as an independent server and it is starting as a separate server. The reason why Bokeh is used as a separate server is that it affords many novel and powerful capabilities, such as:

- UI widgets and plot selections driving computations and plot updates.
- Intelligent server-side downsampling of large datasets.
- Streaming data automatically updating plots.

- Plot and dashboard publishing for wider audiences [9].

4.1.5 Flask

The reason why Flask was chosen is that it is a lightweight Web framework written in Python [10]. It allows easily and comfortable start developing user interfaces. Due to the fact, that by company was required to create flexible and integrative web-based user interface we suggested to choose pure CSS, HTML and JavaScript for developing. The Flask that is responsible for generating and delivering pages to the client.

Flask pros:

- Routing URL
- Integration with Jinja2, which is one of the most powerful Python template engine
- Pluggable session management
- Interactive web-based debugger
- Flexible application configuration managements [10]

4.1.6 HTML/CSS/JavaScript

Modern web applications cannot avoid the use of HTML, CSS and JavaScript [11]. Was used to take technologies, which lie in a web-based development. HTML is responsible for the structure of pages using markup [12] and CSS for the style of the elements which are used in HTML. JavaScript is most commonly used in web browsers, and, in that context, the general-purpose core is extended with objects that allow scripts to interact with the user, control the web browser, and alter the document content that appears within the web browser window [13] that is the reason why JavaScript is used in the development.

4.2 Development tools

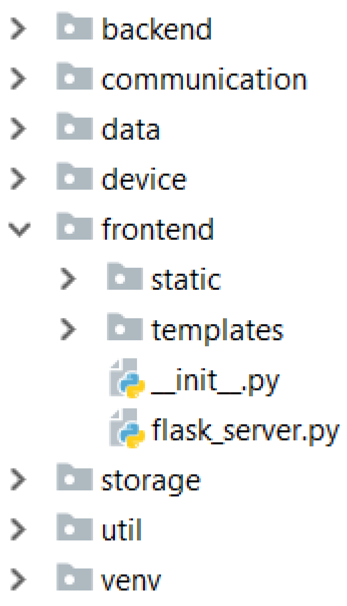
During developments was used *IDE* (Integrated Development Environment) *PyCharm* and *Google Chrome's* built-in *DevTools* [14]. As a tool for exchanging code with other developers in our team was used *Git*.

5 Development of Graphical User Interface

In this paragraph reviews the creation of graphical user interface, project structure and methods, which were used while developing the application.

5.1 Project Structure

The whole project consists of several logical modules, such as: logical device (SFS Cube™), Backend, Database, Analysis and GUI, but within this thesis we will review only GUI and partially backend part of the web application. The Figure 1 demonstrates the file structure of the project, where each of the system components is in a separate subdirectory. The web server with the client part is in the frontend directory; This component is central to this thesis.



```
> backend
> communication
> data
> device
v frontend
    > static
    > templates
      __init__.py
      flask_server.py
> storage
> util
> venv
```

Figure 1. Project file structure

5.1.1 Project Isolation

In order, to ensure the independence of the project from the Python system interpreter and globally installed libraries, we use the *virtualenv* tool, with which you can create an isolated environment with the required version of the interpreter and libraries inside the project directory, thereby avoiding possible conflicts due to versions, since the application may be dependent on specific library versions. The *virtualenv* tool is designed as a *CLI* (Command Line Interpreter) module, and makes it easy to switch between virtual environments, activating them transparently for applications [15]. On the Figure 1, we can see the *venv* subdirectory containing all the dependencies of the project, and after activating the virtual environment, all Python packages will be installed locally in this directory.

5.1.2 Web application architecture

Web application consists of two main parts, which are implemented as interacted subsystems on the separate servers:

- Page server (Flask), which generates and returns final html pages to the client. It is the first level server which interacts with the client.

- The server based on Bokeh is responsible for creating plot components for graphs and delivering them to the server pages as inline HTML elements; later, it acts as a point for synchronization of graphs using the *WebSocket* protocol.

5.1.3 Multi-threading

Traditionally, to scale server applications in Python, a multi-process approach is used. Due to the fact, that whole application (server and client side) is intended for local work on the client machine, we use a simple multi-thread approach for reducing process management costs, because we have only one process. The multithreading library is lightweight, shares memory, responsible for responsive user interface and is used well for input/output bound applications [16]. In our case it means that both servers will run in the separate threads. Therefore, for example, server's start and shutdown can be simply performed. It can be seen on the Figure 2.

```python
from core import BOKEH_SERVER, WEB_SERVER
BOKEH_SERVER.start()
WEB_SERVER.start()
```

Figure 2. Starting Bokeh and Flask servers

5.1.4 Configuration and Initialization

Configuration is going on the initialization of the servers. In this case configuration is done as a simple Python code. Figure 3 shows the ports which are given for the current servers.

```
WEB_PORT = 5000
BOKEH_SERVER_PORT = 5001
```

Figure 3. Configuration of the servers

Thanks to multi-thread approach server setup is simple (Figure 2).

5.1.5 Jinja2 templating

In projects for dynamic pages generation on the server-side template engines is a common way which reduces the size of the source code of the pages and achieve high reuse of the composite code, which is repeated. Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django’s templates [17], and in this project the choice was made in favor of Jinja2.

In the process of templating is going the placement of HTML-elements generated by the Bokeh server. Since by default Jinja2 escapes HTML special characters, we must use the safe filter (Figure 4), which disables escaping and the “trusted” string will be output as is.

```
<div class="rightcolumn">
    <div class="row">
        <div class="column">
            <div class="data">
                {{ bk_plot | safe }}
            </div>
        </div>
    </div>
</div>
```

Figure 4. Bokeh element integration into HTML using safe filter

5.2 Design of the interface

Graphical User Interface design took a big part of a development, because it should be flexible and integrative, using pure CSS, HTML and JavaScript. The developing of design consists of several parts:

- The understanding what does client wants to get
- The developing of the prototype and showing of it to the client
- Finding optimal solution in CSS and HTML to avoid the excess code lines

The required design was developed. We were using the nesting of elements for creating the base HTML template (Figure 5). The nesting of elements means that we can position elements in any way and the whole design template will adapt and resize automatically.

```
{% block content %}
    <div class="base-body">
        <div class="leftcolumn">
            <div class="row">
                <div class="column">
                    <div class="data"></div>
                </div>
            </div>
            <div class="row">
                <div class="column">
                    <div class="data"></div>
                </div>
            </div>
        </div>
        <div class="rightcolumn">
            <div class="row">
                <div class="column">
                    <div class="data"></div>
                </div>
            </div>
            <div class="row">
                <div class="column">
                    <div class="data"></div>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

Figure 5. Jinja2 HTML template

5.2.1 User flow

Login page is the first page which user sees. On Figure 6 we can see the design of the login page, which is flexible and looks the same on different screen resolutions.

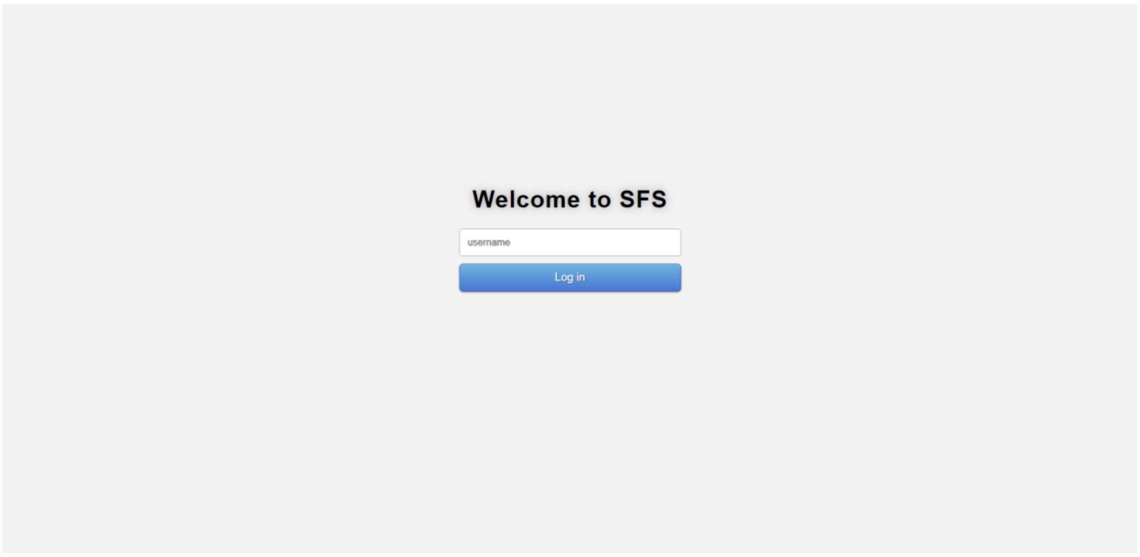
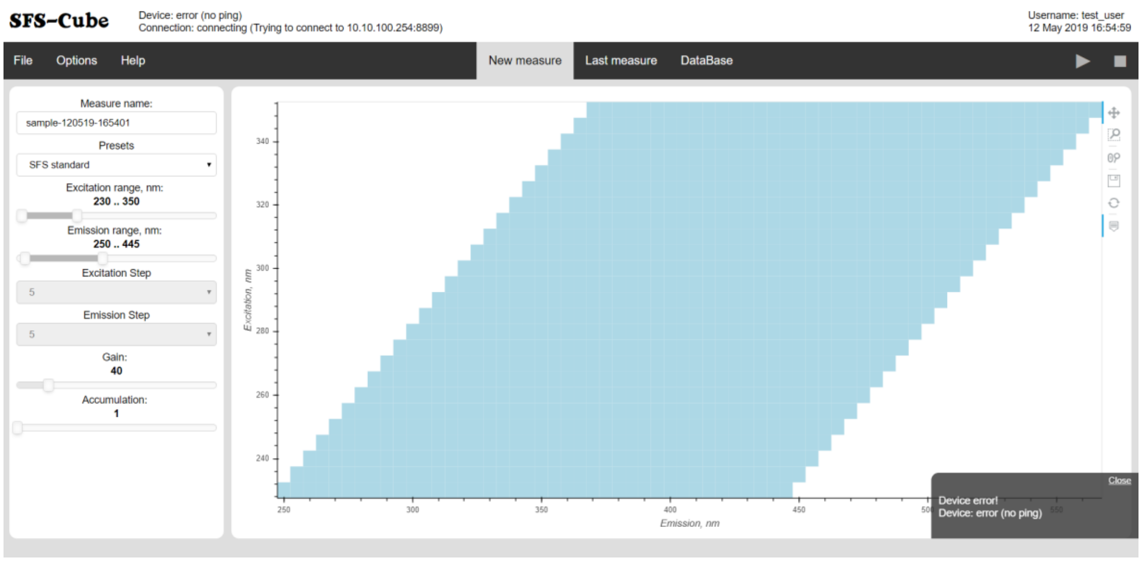


Figure 6. First page of the application, login page

After the authorization, client is going to another page (Figure 7) where we can see displaying of current day, month, year and time, which is realized by using JavaScript (Figure 10).


SFS–Cube

Device: error (no ping)
Connection: connecting (Trying to connect to 10.10.100.254:8899)

Username: test_user
12 May 2019 16:54:59

Measure name: sample-120519-165401
Presets: SFS standard
Excitation range, nm: 230 .. 350
Emission range, nm: 250 .. 445
Excitation Step: 5
Emission Step: 5
Gain: 40
Accumulation: 1

Device error!
Device: error (no ping)

Figure 7. New measure page

On the left side of the screen we can see the control elements, such as range sliders, input box, select box and sliders (Figure 8). These elements were created on Bokeh side. The creation of this elements is shown on Figure 11.

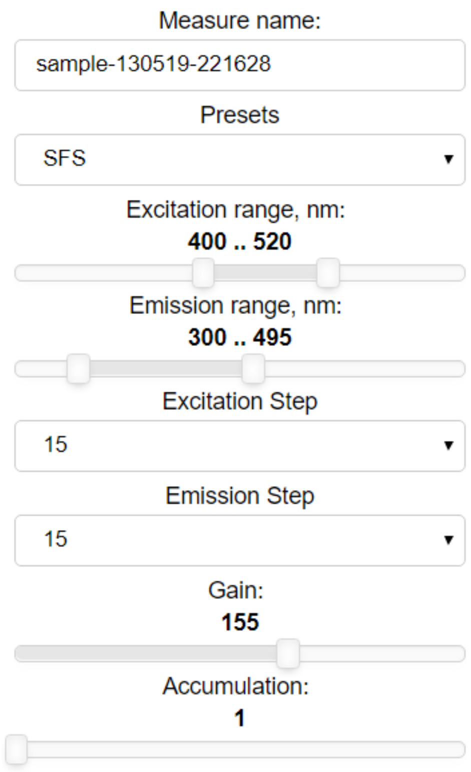
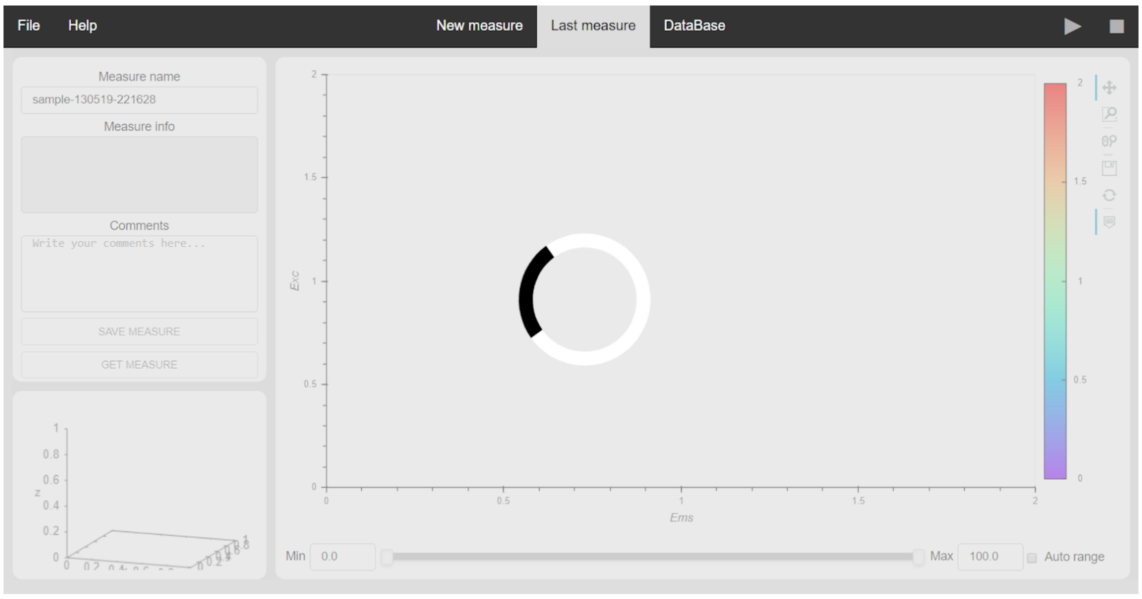

Measure name:
sample-130519-221628
Presets
SFS
Excitation range, nm:
400 .. 520
Emission range, nm:
300 .. 495
Excitation Step
15
Emission Step
15
Gain:
155
Accumulation:
1

Figure 8. Control elements display

While the loading of elements is not completed the animated pre-loader (Figure 9) is running on the page with disabling the elements on the current page. This functionality is implemented only on last measure page, because only there can be elements which are taking time to occur in the window.



Measure name: sample-130519-221628
Min 0.0 Max 100.0

Figure 9. Pre-loader example

5.2.2 Elements style customization

Function on Figure 10 sets to HTML element the current date and time. That was one of the requirements from the clients, because this functionality allow people to manage the time which is going during the processes. For example, for the client is comfortable when he can follow the time while using this user interface.

```
function date_time(id) {
    date = new Date;
    year = date.getFullYear();
    d = date.getDate();
    day = date.getDay();
    h = date.getHours();
    var month = [Jan", "Feb", Mar", "Apr",
                 "May", "Jun",
                 "Jul", "Aug", "Sep",
                 "Oct", "Nov", "Dec"];
    months = month[date.getMonth()];
    if (d < 10) d = "0" + d;
    if (h < 10) h = "0" + h;
    m = date.getMinutes();
    if (m < 10) m = "0" + m;
    s = date.getSeconds();
    if (s < 10) s = "0" + s;
    result = d + ' ' + months + ' ' + year + ' ' + h + ':' + m + ':' + s;
    document.getElementById(id).innerHTML = result;
    setTimeout('date_time("' + id + '");', '1000');
    return true;
}
```

Figure 10. Date and time realization on pages using JavaScript bases

On Figure 7 on the left side of the screen we can see the controls elements, which were done on Bokeh server side. Figure 11 displays the creation of the controls. The method *widgetbox* [18] allows us to group the elements on Bokeh server side. We can see that was used *sizing_mode='stetch_both'* that means that component is completely responsive, independently in width and height, and will occupy all the available horizontal and vertical space, even if this changes the aspect ratio of the component [19]. The *css_classes* which are given allows us to customize the displaying of the elements (Figure 11).

```
preset_controls = widgetbox(meas_name_text_input,
                            preset_dropdown,
                            ex_slider,
                            em_slider,
                            ex_step_dropdown,
                            em_step_dropdown,
                            gain_slider,
                            accumulation_slider,
                            sizing_mode='stretch_both',
                            css_classes=['col-flex-panel',
                                         'stretch-controls'])
```

Figure 11. Creating controls on Bokeh

The developing Responsive Web Design is one of the main goals of my thesis. RWD (Responsive Web Design) is a web development approach that creates dynamic changes to the appearance of the web application [2]. Every element, which is generated on Bokeh server side has its own CSS style, that means to create responsible elements we need to rewrite the given CSS of the elements. The Figure 12 shows how does control elements looks without CSS customization.

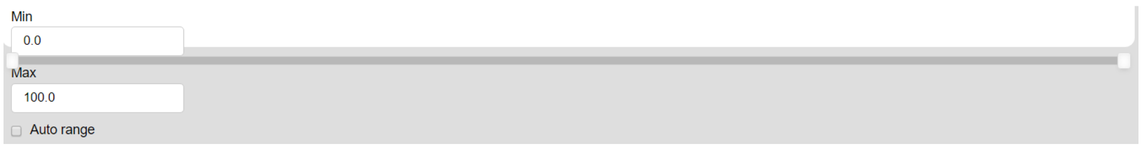


Figure 12. The Bokeh elements design before the customization

To customize elements was chosen Flexible Layout Module [20] that allows to create the flexible elements, which will resize automatically and seems appropriate (Figure 13).

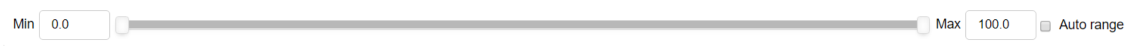


Figure 13. The Bokeh elements design after customization

For example, on Figure 14 we can see how was removed the title CSS place from the Bokeh elements.

```
.no-title-slider label, .no-title-slider .bk-slider-value {
    display: none !important;
}
```

Figure 14. CSS customization example

5.3 Request flow

In this paragraph step-by-step process of handling request from the user will be fully described and how does client agent (browser etc.), Bokeh and Flask servers are interacting with each other.

5.3.1 User management

Almost all pages of the project need basic user identification. Whole application works locally on the user's side and there is no need for a full implementation of the mechanism for managing sessions and security of the users. We can identify users just by login.

Authorization will be stored as a record in cookies; To check this information, it is reasonable to register a *callback* by *Flask.before_request* [21], which is called every time before starting the processing of a request. This *callback* reads the login from cookies and if it is not set, the server returns redirect to the login page (Figure 15). Otherwise, the username will be stored in a global variable, and the query will continue. Some pages do not need authorization, so being on one of them, the check does not occur.

```
@staticmethod
@app.before_request
def before_request():
    global username
    username = request.cookies.get('username')
    if request.path in ('/', '/login', '/static/logIn.css'):
        return
    if not username:
        return redirect('/login')
```

Figure 15. User identification handle

The login handler, represented in Figure 16, is responsible for the login page. It performs two functions depending on the HTTP method: if it is a GET method, it simply returns the login page; if it is a POST method, it expects a completed form with a username, and if this data is available, identification is complete. The server returns redirection to the default page and sets the username in the cookie. Such a combination of two methods (GET and POST) in one handler is a common practice in Flask, in case the code is small.

```
@staticmethod
@app.route("/login", methods=['GET', 'POST'])
def login():
   if request.method == 'GET':
       return render_template('login.html')

   username = request.form.get('username')

   if not username:
       return redirect('/login')

   response = redirect(url_for('new_meas_page'))
   response.set_cookie('username', username)
   return response
```

Figure 16. Login handler

After the work, the user has the opportunity to log out. As a result, it will go to the address */logout*, and the server will clear the user name. Figure 17 shows the handler that is responsible for this.

```
@staticmethod
@app.route("/logout")
def logout():
   response = redirect('/login')
   response.set_cookie('username', '')
   return response
```

Figure 17. Logout handler

5.3.2 The beginning of handling

If the client was successfully identified, the execution of the request proceeds to a pattern matching to select one of the handlers registered by the *Flask.route* function. This process, also called routing, is performed inside the Flask framework. Figure 18 shows the binding of the handler to the path.

```
@staticmethod
@app.route("/new_meas")
def new_meas_page():
    allow = {'bk_plot', 'bk_preset_controls'}
    return FlaskServerThread._compile_template("new_meas.html", "New
                                               Measure", allow, 'new')
```

Figure 18. Binding static method new_meas_page with new measure path name

If the target handler is not found, the *HTTP error 404 Not found* is returned, otherwise almost all the handlers start generating a page that is executed by the *_compile_template* function (Figure 20). Before rendering the page template, it requests prepared elements from the Bokeh server using the *_pull_bokeh_roots* (Figure 21) function, which are then used in templating. It also independently requests basic controls, such as the header, main menu (Figure 19) and footer, which should be placed on each page. The process of requesting items is described in detail in section 5.3.3.

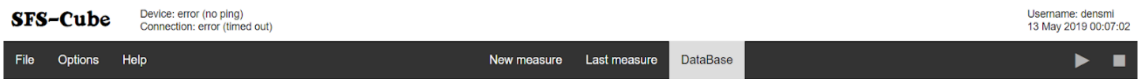


Figure 19. Basic controls.

```
@staticmethod
def _compile_template(body_template, title, allowed, bk_app):
   allow = {'sfs_start_stop_btn', 'device_status', 'progress_bar',
'bk_popup_div'}
   kwargs, script = FlaskServerThread._pull_bokeh_roots(allow, 'ctrl')

   kw, bk_script = FlaskServerThread._pull_bokeh_roots(allowed, bk_app)
   kw['bk_script'] = bk_script

   return render_template(body_template, title=title, username=username,
                          **kwargs, **kw, ftr_bk_script=script)
```

Figure 20. Render templates with creating base element

5.3.3 Elements initialization

The page server (Flask) creates a request to the Bokeh server to create the necessary elements. However, before that, we must create a separate session for each user, in which the bokeh server will manage the life cycle of the created elements. Its unique identifier *(session_id)* is equal to the user's login name and Bokeh's handler name - it allows maintaining a shared view in different browser tabs opened by an individual user.

For the plots creation is responsible the handler, which is implemented on the side of the Bokeh server, whose name is specified in *bk_app*, but its description is beyond the scope of this thesis.

Next, the Bokeh server initializes the internal state of each element using the identifier of the created earlier session and returns them to the page server (Flask) over the network as a special objects. The functionality that is described is shown on Figure 21.

```
@staticmethod
def _pull_bokeh_roots(allowed, bk_app):
    with pull_session(session_id=username + '_' + bk_app,
                      url='http://localhost:5001/' + bk_app) as session:
        bokeh_roots = RenderItem(sessionid=session.id,
                                 roots=session.document.roots,
                                 use_for_title=False)
    log.debug(bokeh_roots)
    script = CUSTOM_JS_FOR_3D_PLOT + \
             script_for_render_items(None, [bokeh_roots],
                                     app_path='/' + bk_app,
                                     absolute_url='http://localhost:5001/'
                                                  + bk_app)
    roots_dict = FlaskServerThread.roots_prepare(bokeh_roots, allowed)
    return roots_dict, script
```

Figure 21. Getting Bokeh elements function

Inside the *_pull_bokeh_roots* function, the auxiliary method *roots_prepare* (Figure 22) is used, which from the container of Bokeh objects *(bokeh_roots)* renders the final HTML code for each element. Since each object is uniquely named, it is convenient to return a dictionary, where the key is the name of the element, and the value is its HTML code. The method also accepts a set of expected names *(allow)*, and if an element with a name that is not in *allow* was detected, a warning is logged, since unexpected elements will later lead to an error during the page generation stage.

```
@staticmethod
def roots_prepare(bokeh_roots, allow):
    d = {}
    for root in bokeh_roots.roots:
        d[root.name] = div_for_render_item(root)
        if root.name not in allow:
            log.error('Here is some root that is not in allowed or html
                      template.')
    return d
```

Figure 22. Helper for rendering roots

Finally, the resulting dictionary will be passed to the *render_template* [22] function as unpacked key word arguments to provide access to the elements inside the template.

5.3.4 Interactive visualization

After the prepared HTML page is sent to the client, the interaction with the client is still going on. Since the server side of the application connects to scientific devices and is a source of dynamic data, continuous synchronization of elements with the Bokeh server is necessary to support interactive visualization. The Figure 23 [23] demonstrates the principle of this work.

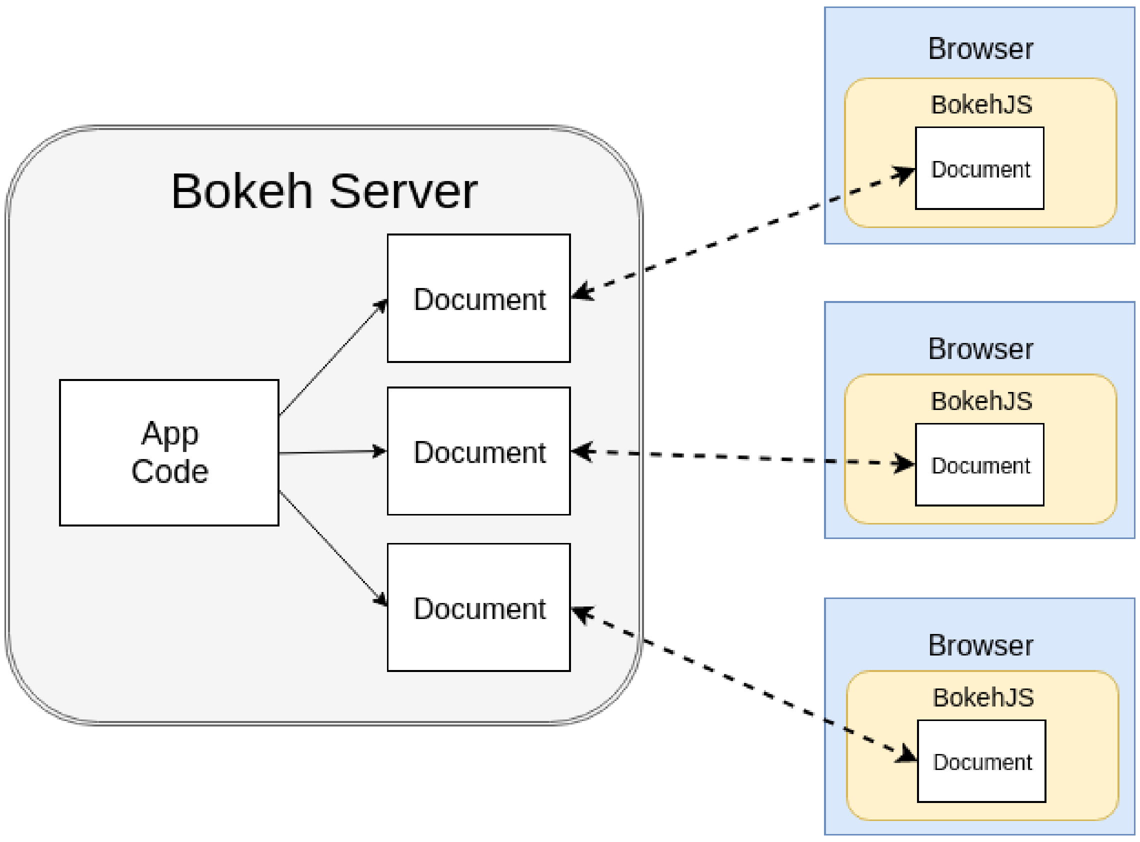
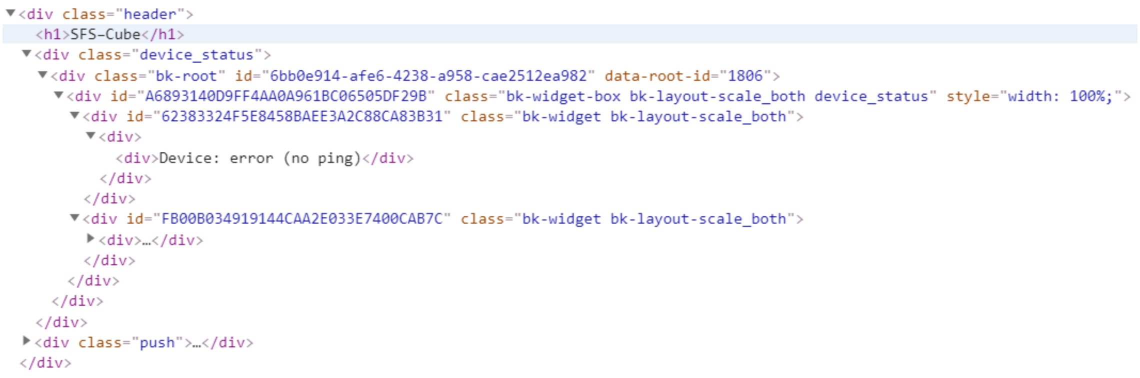


Figure 23. Bokeh server operation scheme

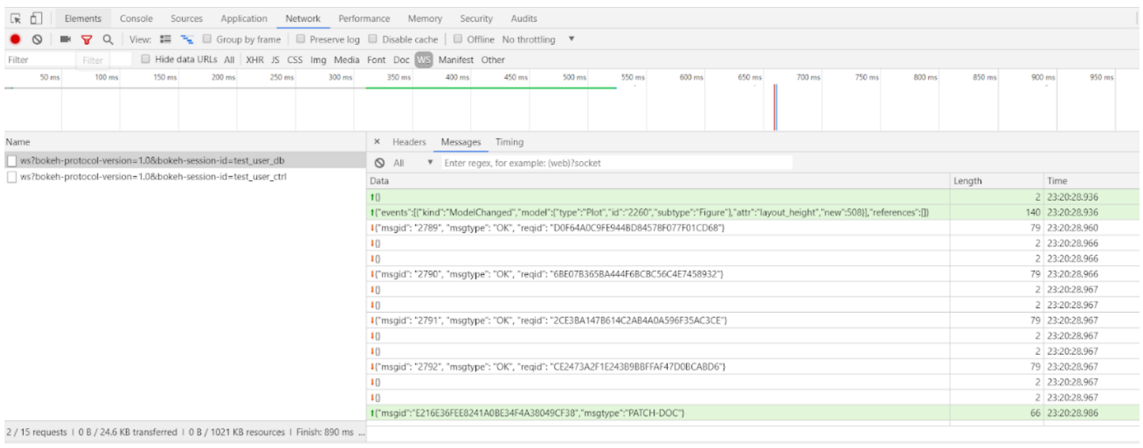
In the previous section 5.3.3, during initialization, to each object is assigned a unique identifier, which is reflected in the HTML code of the element. This identifier is then used when sending events using a *WebSocket* connection. The example is shown on the Figure 24.



```
▼<div class="header">
   <h1>SFS-Cube</h1>
 ▼<div class="device_status">
   ▼<div class="bk-root" id="6bb0e914-afe6-4238-a958-cae2512ea982" data-root-id="1806">
     ▼<div id="A6893140D9FF4AA0A961BC06505DF29B" class="bk-widget-box bk-layout-scale_both device_status" style="width: 100%;">
       ▼<div id="62383324F5E8458BAEE3A2C88CA83B31" class="bk-widget bk-layout-scale_both">
         ▼<div>
             <div>Device: error (no ping)</div>
           </div>
         </div>
       ▼<div id="FB00B034919144CAA2E033E7400CAB7C" class="bk-widget bk-layout-scale_both">
         ▶<div>…</div>
         </div>
       </div>
     </div>
   </div>
 ▶<div class="push">…</div>
 </div>
```

Figure 24. HTML example of some Bokeh elements and their identification.

The connection is duplex; events are sent in both directions, which allows the server to react in real time to user input and redraw plots when updating the data source on the server side. These messages are transmitted in JSON format and the Figure 25 shows the payload of some events; as we can see, the identifier and other metadata are transmitted.



Elements Console Sources Application Network Performance Memory Security Audits
View: Group by frame Preserve log Disable cache Offline No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
50 ms 100 ms 150 ms 200 ms 250 ms 300 ms 350 ms 400 ms 450 ms 500 ms 550 ms 600 ms 650 ms 700 ms 750 ms 800 ms 850 ms 900 ms 950 ms
Headers Messages Timing
All
Enter regex, for example: (web)?socket

Name
ws?bokeh-protocol-version=1.0&bokeh-session-id=test_user_db
ws?bokeh-protocol-version=1.0&bokeh-session-id=test_user_ctrl

Data	Length	Time
↑{}	2	23:20:28.936
↑{"events":[{"kind":"ModelChanged","model":{"type":"Plot","id":"2260","subtype":"Figure"},"attr":"layout_height","new":508}],"references":[]}	140	23:20:28.936
↓{"msgid": "2789", "msgtype": "OK", "reqid": "D0F64A0C9FE944BD84578F077F01CD68"}	79	23:20:28.960
↓{}	2	23:20:28.966
↓{}	2	23:20:28.966
↓{"msgid": "2790", "msgtype": "OK", "reqid": "6BE07B365BA444F6BCBC56C4E7458932"}	79	23:20:28.966
↓{}	2	23:20:28.967
↓{}	2	23:20:28.967
↓{"msgid": "2791", "msgtype": "OK", "reqid": "2CE3BA147B614C2AB4A0A596F35AC3CE"}	79	23:20:28.967
↓{}	2	23:20:28.967
↓{}	2	23:20:28.967
↓{"msgid": "2792", "msgtype": "OK", "reqid": "CE2473A2F1E243B9BBFFAF47D0BCABD6"}	79	23:20:28.967
↓{}	2	23:20:28.967
↓{}	2	23:20:28.967
↑{"msgid":"E216E36FEE8241A0BE34F4A38049CF38","msgtype":"PATCH-DOC"}	66	23:20:28.986

2 / 15 requests | 0 B / 24.6 KB transferred | 0 B / 1021 KB resources | Finish: 890 ms ...

Figure 25. Log of WebSocket messages in Chrome DevTools

6 Summary

The aim of this thesis was to develop web-based GUI for the original equipment manufacturer (LDI Innovation) that would make it easier for the user to communicate with the device, provide better capabilities for data management, and make the user interface ready for integration of novel software solutions. LDI Innovation is a manufacturer which is engaged in scientific.

First, the Python language was selected as core language along with CSS, HTML and JavaScript. Flask web framework was selected for GUI, because it is lightweight, and it uses Jinja2 template engine. These instruments allow to start developing faster and to save the time for the writting the programm code.

Secondly, the design of the interface and its intuitive placement of the elements were discussed with the head of the company. The expected result was obtained.

In conclusion, the main goal of this thesis was done successfully. The GUI that was developed is currently in use by the clients of LDI Innovation.

References

[1] LDI Innovation, "SFS-Cube - LDI Innovation," [Online]. Available: https://ldi-innovation.com/index.php/sfs-cube/. [Accessed 17 05 2019].

[2] A. Schade, "Responsive Web Design (RWD) and User Experience," *Nielsen Norman Group,* 05 05 2014.

[3] D. Terrence, "26 Tools and Frameworks for HTML-based Desktop and Web App Interfaces," *Visual Studio Magazine,* 25 01 2017.

[4] B. Kohan, "Guide to Web Application Development," [Online]. Available: https://www.comentum.com/guide-to-web-application-development.html. [Accessed 17 05 2019].

[5] M. Sanner, "Python: A Programming Language for Software Integration and Development," *Journal of Molecular Graphics and Modelling,* 1998.

[6] S. Raschka, Python Machine Learning, United Kingdom: Packt Publishing Ltd., 2015.

[7] L. Barnard and M. Mertik, "Usability of Visualization Libraries for Web Browsers," *International Journal of Computer Applications (0975 - 8887),* vol. 121, 2015.

[8] Educba, "Python vs C++," [Online]. Available: https://www.educba.com/python-vs-c-plus-plus/. [Accessed 17 05 2019].

[9] Bokeh Development Team, "Bokeh: Python library for interactive visualization," 2018. [Online]. Available: https://bokeh.pydata.org/en/latest/. [Accessed 17 05 2019].

[10] M. Copperwaite and C. Leifer, Learning Flask Framework, United Kingdom: Packt Publishing Ltd., 2015.

[11] M. Grinberg, Flask Web Development, O'Reilly Media, 2018, p. 12.

[12] Refsnes Data, "Introduction to HTML," 1998. [Online]. Available: https://www.w3schools.com/html/html_intro.asp. [Accessed 17 05 2019].

[13] D. Flanagan, JavaScript: The Definitive Guide, Fifth Edition, O'Reilly Media, 2006, p. 1.

[14] Google, "Chrome DevTools, Tools for Web Developers," [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/. [Accessed 17 05 2019].

[15] I. Bicking, "Virtualenv," 2007. [Online]. Available: https://virtualenv.pypa.io/en/latest/. [Accessed 17 05 2019].

[16] N. Bosco, "Multithreading vs Multiprocessing in Python," 26 11 2017. [Online]. Available: https://blog.usejournal.com/multithreading-vs-multiprocessing-in-python-c7dc88b50b5b. [Accessed 17 05 2019].

[17] A. Ronacher, "Jinja2 Documentation," 2008. [Online]. Available: http://jinja.pocoo.org/docs/2.10/. [Accessed 17 05 2019].

[18] Bokeh Development Team, "Bokeh: Python library for interactive visualization," 2018. [Online]. Available: https://bokeh.pydata.org/en/latest/docs/reference/models/layouts.html#bokeh.models.layouts.WidgetBox. [Accessed 17 05 2019].

[19] Bokeh Development Team, "Bokeh: Python library for interactive visualization," 2018. [Online]. Available:

https://bokeh.pydata.org/en/latest/docs/reference/models/layouts.html#bokeh.models.layouts.LayoutDOM.sizing_mode. [Accessed 17 05 2019].

[20] Refsnes Data, "CSS FlexBox (Flexible Box)," 1998. [Online]. Available: https://www.w3schools.com/css/css3_flexbox.asp. [Accessed 2019 05 17].

[21] Pallets Team, "API-Flask 1.0.2 documentation," 2010. [Online]. Available: http://flask.pocoo.org/docs/1.0/api/. [Accessed 17 05 2019].

[22] Pallets Team, "API-Flask 1.0.2 documentation," 2010. [Online]. Available: http://flask.pocoo.org/docs/1.0/api/#flask.render_template. [Accessed 17 05 2019].

[23] Bokeh Developer Team, "Running a Bokeh Server," 2018. [Online]. Available: https://bokeh.pydata.org/en/latest/docs/user_guide/server.html. [Accessed 2019 05 17].