TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Eduard Egils Looga 164033IAPB

THE DEVELOPMENT OF SPECTRAL FLUORESCENCE SIGNATURE ANALYSER SOFTWARE

Bachelor's thesis

Supervisor: Innokenti Sobolev
Ph.D.
Eduard Petlenkov
Professor

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogiateaduskond

Eduard Egils Looga 164033IAPB

SPEKTRAALSE FLUORESTSENTSI SIGNATUURI ANALÜSAATORI TARKVARA ARENDAMINE

bakalaureusetöö

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: Eduard Egils Looga

20.05.2019

Abstract

Ecological problems are some of the most important problems we face at the moment. One of those is quality of water ecology. Water is getting polluted by chemicals and human wastes. Different types of pollutions require different types of treatments. Process of distinguishing pollution types consumes time and resources.

LDI Innovation is a Research & Development and Engineering Company that provides Engineering services which includes prototyping, development and verification of various instruments and tools. It offers real-time and on-line sensing solutions in industrial, environmental and security domains which could save resources and time that otherwise would be spent for secondary processes like sample pre-treatment.

SFS Cube™ is a compact scanning spectrometer designed to measure the Spectral Fluorescence Signature of liquid, powder and solid samples in environmental and industrial applications. Important part of SFS Cube™ is the client-side application used for instrument control, visualisation and data analysis.

The main objective of this thesis is to develop a back-end part of the application for collecting and visualising data from the SFS Cube™ using the most up-to-date software development techniques.

The result of this thesis is a standalone Python application, that communicates with SFS Cube™, parses and stores received data and is able to visualise stored measurement data.

This thesis is written in English and is 32 pages long, including 5 chapters and 6 figures.

Annotatsioon

Spektraalse fluorestsentsi signatuuri analüsaatori tarkvara arendamine

Looduskeskkonna reostus on üks olulisematest probleemidest, mille ees inimkond seisab. Meie mered ning teised veekogud on reostatud erinevat tüüpi plastikute, naftapõhiste kemikaalide, mürkide ning liigväetamisest tuleneva eutrofeerumise tulemustega, mida tuleks korrigeerida erinevate meetmetega. Et neid erinevat tüüpi plastikuid, naftasaadusi ja muid looduslikke ning tehislikke reostusaineid kindlaks teha, tuleb tüüpiliselt võtta veeproovid ning analüüsida neid laborites. Tüüpiliselt on proovide analüüsimise protsess aeganõudev ning ressursimahukas.

LDI Innovation on teadus- ja arendustegevusele ning insenertehniliste lahenduste loomisele keskendunud ettevõte, mille teenuste hulka kuulub mitmete optika- ja fotoonikaseadmete prototüüpimine, arendamine ning katsetamine. Ettevõte pakub reaalajaliselt mõõteandmeid pakkuvaid sensorlahendusi tööstus-, keskkonnahoiu ning julgeoleku sektorites, mis võimaldavad kokku hoida aega ning vahendeid, mis muidu võiks kuluda teisejärgulistele protsessidele nagu proovide eeltöötlemine.

SFS Cube™ on kompaktne spektrofluoromeeter, millega mõõdetakse Spektraalseid Fluorestsents Signatuuri(SFS) vedelikes, pulbrites ning tahketes proovides põhiliselt tööstus- ning julgeolekurakendustes. Oluline osa SFS Cube™ seadmest on kliendipoolne rakendus, millega seadet juhitakse, sealseid mõõteandmeid visualiseeritakse ning analüüsitakse. Käesoleva bakalaureusetöö eesmärgiks oli arendada SFS Cube™ seadme tarkvara funktsionaalne osa, kasutades selleks tänapäevast lähenemist andmete kogumisel ning visualiseerimisel.

Käesoleva bakalaureusetöö tulemuseks on Pythoni rakendus, mis võimaldab SFS Cube™ seadmest andmeid vastu võtta, sorteerida, salvestada ning visualiseerida.

Rakenduse arendus hõlmas seadmega ühendatud arvutisse kohaliku serveri loomist, kasutades laialt levinud ja toetatud Python programmeerimiskeele võimalusi. Rakenduse käitamine läbi veebibrauseri muudab selle arvuti või nutiseadme operatsioonisüsteemist sõltumatuks lahenduseks ning Pythoni laialdane kasutamine nii programmeerijate, inseneride kui teadlaste hulgas tähendab väga suure hulga komplekssete teekide olemasolu, mis võimaldab seadmele arendada üpris efektiivselt keerulise analüüsimooduli hilisemas etapis. Arendatud rakendus juba võimaldab SFS Cube™ seadme kõiki mõõtmisparameetreid (ergastuslainepikkused, emissioonilainepikkused, fotokordisti võimendus, signaali keskmistamine jne) muuta ning eelnevalt defineeritud mõõtmisaknaid valida. Samuti võimaldab rakendus mõõdetud SFS spektreid visualiseerida nii kahe- kui kolmemõõtmeliste piltidena. Visualiseerimisel on valitud üheselt loetav värviskaala fluorestsentsi intensiivsuste kuvamiseks mõõtepunktides. Seade salvestab andmed lihtsasti kasutatavasse SQLite andmebaasi mõõtepunktide (ergastuslainepikkus, emissioonilainepikkus, intensiivsus) kaupa. Andmebaasis on võimalik mõõteandmeid erinevate projektide ning erinevate kasutajate vahel jagada ning andmebaasis on salvestatud kõik mõõtmist puudutavad andmed.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 32 leheküljel, 5 peatükki, 6 joonist.

List of abbreviations and terms

AP	Access Point
API	Application Programming Interface
C++	Programming language
GUI	Graphical User Interface
HTML	Hypertext Markup Language
OEM	Original Equipment Manufacturer
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
Python	Programming language
SFS	Spectral Fluorescence Signature
SQL	Structured Query Language
TCP	Transmission Control Protocol
UI	User Interface
Web-Based application	Client–server computer program which the client runs in a web browser

Table of contents

List of figures

1 Introduction

1.1 Problem Statement

Nowadays quality of water ecology is one of the most important problems humanity faces. Humanity pollutes water with chemicals and wastes causing ecological disasters. Marine life could be affected by those pollutions and lead to an aggressive mutation. For example, algae could mutate in such a way to start producing toxins. Toxins affect water sources and their inhabitants, consequently affecting us.

Example of such an outbreak could be an accident near the coast of Norway in May 2019 which killed a numerous population of salmon fish [1]. If pollution was addressed earlier the fish population could be saved.

However, different treatment procedures are required by different types of pollutions. Also, any kind of solution is preceded by problem analysis. This process could consume considerable amount of resources and time.

SFS Cube™ by LDI Innovation is a compact diagnostic instrument that is designed for fast sample analysis by measuring Spectral Fluorescence Signature (SFS) of liquid, powder or solid matter. It could be used not only in environmental, but also to industrial, scientific, medical and law enforcement applications. SFS Cube™ attempts to reduce time and resources spent on pollution research.

The core part of the end-user software for SFS Cube™ was written in the year 2005. It was developed using C++ programming language with Microsoft Foundation Classes framework. Later C++ classes were refactored multiple times using different C++ frameworks.

Due to the fact that C++ is a low-level programming language, old code base is relatively large [2]. This limits the development capabilities of new measuring procedures, visualization and analysis algorithms for the data acquisition, storage and

management which in turn hampers the rapid prototyping ability of original equipment manufacturers such as LDI Innovation.

1.2 Aim

The primary goal of the thesis is to develop a backend for the application that would provide control and communication interface with SFS Cube™, visualize recorded SFS spectra and store the collected data in SQL database.

The secondary goal is to research the capabilities of modern software development frameworks which will reduce time consumption for subsequent software development in LDI Innovation.

1.3 Company requirements

LDI Innovation is OEM that follows modern software and hardware development standards. Thus, SFS Cube™ software has following requirements:

- Ability to run on most modern operating systems such as Windows 10, macOS and some Linux distributions
- Be scalable. Software must be able to work with one as well as many users.
- Be modular. Since LDI Innovation offers many solutions in different applications, functional parts of the software must be easily removed and/or replaced.
- Store measurement data. Software must be able receive information from SFS Cube™ and store it permanently.
- Communicate with SFS Cube™. Software must be able to establish connection with SFS Cube™ and allow user to control the device.
- Upgradable. Future development of new functional parts must not require changes in current software.
- Detachable. Software must be able to run in server mode, meaning that it is possible to run the software and connect to it from the local and/or global network.

1.4 Outline

This thesis is organised in following way. Chapter 2 contains pre-production analysis. Chapter 3 contains database development description and structure overview. Chapter 4 contains back-end development description and back-end software overview.

2 Development process

Development process was started with analysis of current version of software, its source code, methods and frameworks used. Also, important factor for selecting new frameworks and development methodology was performance. New software must perform as fast as old or better. Furthermore, new software must be easily scalable and upgradable. New software functionality must be expandable without the need to change any of the previously written code.

2.1 Old software overview

Old software was developed in the year 2005. The software was developed using C++ programming language (with Microsoft Foundation Classes framework). Software went through several iterations, some of those expanded functionality, while others reduced. Currently old software has following functionality:

- Data reception.
- Data storage.
- Data visualisation.

Old software is able to receive messages from the SFS Cube™ device, parse them and act accordingly. Current software works in master-slave communication mode where SFS Cube™ acts as a slave device and client Personal Computer (PC) acts as master machine. Master-slave relation means that device does not request data from the master machine but receives commands and executes them.

Old software was not designed to work with databases. The software used a file system. For each measurement and analysis result it creates a new file. For each new project it creates a new folder. Thus, old software was inefficient with space and data management. It also made data transfer more complicated.

Old software visualised data using Graphical User Interface (GUI) elements written in C++ by LDI Innovation development team.

2.2 Methodology selection

Before development main programming languages were compared. Those were: Java, Python and C++. [3] Both C++ and Java would perform better that Python in speed. Java requires installed Java Virtual Machine and the client machine and has to be compiled for each Operating System (OS) separately. C++ also is compiled language. It requires compilation for individual OS. Also, C++ has less data visualisation libraries available that Python.

Python was chosen as the main programming language to achieve desired results, because it is an interpreted, high-level general language with big number of standard libraries written by the members of a global software community.

Python programming language will be used for backend development along with scientific libraries NumPy (for computation) [4] and Pandas (data structuring) [5]and SQLite database system.

2.3 Development organisation

Development was divided into several phases that start one after another. Those phases are: database development, plot engine development and API development.

Phases structured as follows:

- Additional feature implementation (for previous part)
- Possible solution research and further analysis
- Main part development
- Testing and bug fixing

Code development process was managed via Git version control system. Also, weekly project meetings were conducted.

3 Database development

First phase is database development. For data storage SQLite database engine was chosen. It is a light-weight, zero-configuration database [6]. Compared to using filesystem or some of the most used database engines, like PostgreSQL, MongoDB and MySQL, it is faster [7], single-file [8]and more space efficient [5].

SQLite writes data to single file. This help improve the speed and space consumption. Single-file structure allows for greater portability [6].

Also, SQLite does not require server or any other running application to operate. It is operable via reading and writing to the file. Since SQLite does not require server to be running it means that SQLite database requires no configuration and prior set up. This saves development time, that can be used for more detailed structure analysis.

3.1 Database tables

3.1.1 Users

Table for storing usernames and their settings. User settings are Python instance variables of *User* class instances exported with default Python *__dict__*and *str* methods. *__dict__* method represents all instance variables as a Python dictionary and *str* converts Python dictionary into *String* type variable. User settings are stored using this method so we could easily read user settings from database into Python object using Python built-in method *eval*. It converts *String* variable into Python code and runs it. This introduces security risk. Someone could upload malicious code into database. However, this does not concern us in current application, because database sharing between companies is highly discouraged and software checks database and rebuilds or recreates the database if needed.

3.1.2 Measurements

Tables for storing measurement info and raw data. This table has following columns:

- Measurement id. A unique measurement identifier.
- Measurement name. Human-readable measurement name.
- Device configuration hash. Device configuration hash is computed from device configuration parameters. It is required because even with the same samples each device would measure differently. There could be no two or more identical devices. Thus, device configuration, which stores not only device configuration, but also device information, which is required for analysis and error troubleshooting.
- Processed data. Stores processed spectrum data that could further be used for plotting and analysis.
- Raw data. Raw packet that came from the device.
- Measurement parameters. Parameters that were used during measurement. These are the parameters that users can change. Device settings are stored separately.
- Measurement timestamp. Formatted measurement starting time.
- Measurement comment. User commentary.
- Measurement flags. A string of bits. Each bit indicates whether any of the initial measurement parameters were changed by the device. Device is capable of making decisions whether to adjust some parameters or leave them as they arrived.

3.1.3 Projects

Table for storing project names, their owners and time they were created. Projects group different data types under one name. This helps user categorize different measurements and other data types.

3.1.4 Types

Table for storing data type classifiers. This table is used for determining project component type. Projects could include several different types of components.

3.1.5 Project components

Table for storing measurement-project relations. Each row of this table must be unique, consisting of project id, component id and component type. Component type is important for making a decision which table to take further information from.

3.1.6 Device configurations

Table for storing device configurations. This table holds device configuration hashes that were computed from device configuration parameters, device configuration names (it is supposed that in the future it might prove important to assign human-readable names to device configurations) and device configuration parameters, that hold device information as well. As mentioned previously device configurations are required for error troubleshooting and data analysis.

3.2 Database abstraction

To create an abstraction level from directly issuing SQL queries in other program files *storage* package was created.

Storage package consists of following files:

3.2.1 Generic

This file contains the class that directly addresses the database by issuing SQL queries. This class uses built-in Python package *sqlite*. This package allows communication with SQLite database engine. This is class is the lowest abstraction layer from the database engine.

3.2.2 Project

This file contains Python class *Project*. This class is on the highest abstraction level from the database engine. Project class is used in other parts of application. Instances of this class are created in *Generic* database class when corresponding methods are called.

3.2.3 SFS

This file contains all the classes that represent data structures specific to SFS Cube™ device, such as *SFSMeasure* and *SFSConfig*.

SFSMeasure class represents measurements that are recorded with SFS Cube™. It is responsible for raw data parsing. Raw data arrives from the device and after it was assembled from bytes and packages it gets transferred to the class' constructor method. Thus, there are two options for creating *SFSMeasure* instance: either by supplying

constructor with raw data or assigning values manually. Latter method is used for creating an empty measurement for testing and visual purposes. Need for empty measurement may arise when, for example, device was just booted and does not hold any real measurements yet.

3.2.4 User

This file contains *UserParams* class. This class represents user data structure. When user logs in, an instance of this class is created. Class instance stores any parameter that user can change on any given page. This allows us to persist values between the pages and between user logins (program runs).

4 Backend development

Next phase of development is core backend part development, namely development of code that would provide device management and measured data visualisation and storage.

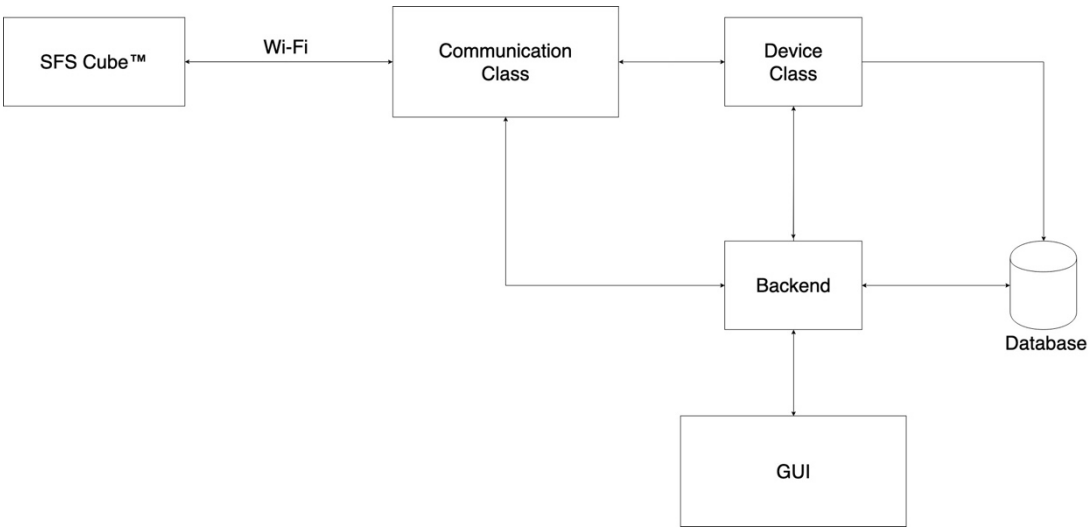

Figure 1. Data-flow diagram

4.1 Communication

Device management and data transfer is implemented through wireless communication namely Wi-Fi. SFS Cube™ has a serial-to-wireless access point station which converts serial stream to Transmission Control Protocol (TCP) packets and vice versa. PC client while connected to SFS Cube™ Access Point (AP) can receive and send serial data using OEM communication protocol.

The *Connection* class is based on proprietary Application layer communication protocol in terms of Open Systems Interconnection (OSI). Its purpose is to combine streaming binary data coming from SFS Cube™ into packets and send them to other parts of software. The *Connection* class also handles outgoing data for device configuration and control.

Python class *SFSDevice* is a logical representation of the SFS Cube™ device in the software. When *SFSDevice* class receives bytes from *Communication* class object it parses them using C structures. To use C structures in Python CStruct [7] package was used. *SFSDevice* converts bytes into one of the special message types and then performs necessary actions, based on the type of message. *SFSDevice* does not alter or process data in any way. *SFSDevice* class' role is to receive bytes and convert them into Python objects which will be further used for user interactions. Another role of *SFSDevice* class instances is to receive commands from other application parts, convert them into bytes and pass to *Communication* class instance for further delivery to an actual SFS Cube™ device. Message types and actions to further perform that *SFSDevice* class can understand are following:

- Heartbeat. This is a message that contains device current information and state. At any given time, device could be actively measuring, calibrating or idling. Heartbeat message is also an indicator of healthy connection between user machine and SFS Cube™ device. Heartbeat message must be received periodically at specific, predetermined intervals (in current software it is set to 2 seconds). If heartbeat message is not received, *SFSDevice* class instance will wait for half a period more and then attempt to reconnect to the physical device. At the same time, *SFSDevice* class will update its status attribute resulting in error message on the client-side.
- Configuration. This message contains current device configuration, that is stored in volatile memory of the SFS Cube™ device. Upon receiving configuration message *SFSDevice* class instance updates its config attribute.
- Measurement. This message contains information about the last measurement that is stored in SFS Cube™ volatile memory. SFS Cube™ device stores its most recent measurement. SFS Cube™ device is set to send measurement information upon request or when it has finished measuring process. Measurement message also contains user parameters (setting that end-user can change), device flags (bit string, that represents changes or correction of user parameters made by the SFS Cube™ device) and current device configuration. Upon receiving *Measurement* message *SFSDevice* updates its *last_measure* attribute value.

- Test measurement. This message contains set of coordinates. When SFS Cube™ receives a command to start a test measurement it is set to stream information at specified rate. This mode is used for device calibration and quality control. Upon receiving this type of message *SFSDevice* class instance updates its *test_measure* attribute.

All of the *SFSDevice* class attributes mentioned above are instances of the Python class *StreamingProperty*. *StreamingProperty* is custom utility class that implements Python property [12] functionality. *StreamingProperty* class instances hold *value_* attribute and are set to trigger specific events that they are bound to. To bind *StreamingProperty* class instance to a certain event one would use *StreamingProperty* class method *bind* by suppling it with callback to call when the *value_* attribute is updated and optional arguments that will be passed when *value_* change occur. This class allowed for data pushing instead of polling. *SFSDevice* class instance is able to push updates onto other software parts. This means that user will get more real-time information and resources, that otherwise will be wasted on periodic callbacks, are saved.

4.2 Plotting engine

One of the company requirements to the software was that it must be able to visualise data received from the SFS Cube™ device. For this purpose, several visualisation engines were compared. Among them were Bokeh [13], Matplotlib [14], Plotly [15] and Seaborn [12]. Before this stage it was decided by our front-end developer Daniel Smirnov [17] that front-end will be a web-based application and Flask will be used as serving web engine. Since Seaborn was lacking functionality of embedding into HTML pages, it was discarded as an option. Another requirement was that graphs must be dynamic and interactive. Matplotlib has options to be built into the HTML page. However, resulting plots will be static their data could not be changed without reloading the web page and rendering a new graph. Both Plotly and Bokeh could be embedded into a web page and updated from the backend without page refresh. However, Plotly does not offer as much customisability as Bokeh. Most of the Bokeh functionality could be overwritten or custom functionality and modules could be developed and integrated into Bokeh. Bokeh also offers vast amount of User Interface (UI) elements aside from graphs. Most of the front-end controls are Bokeh controls.

To further enhance Bokeh plotting options Holoviews [18] library was used. Holoviews is an open-source library that allows developer to visualize their data by describing and annotating it. Holoviews also offers some plotting options that are not present in Bokeh library.

4.3 Backend software structure

Core backend software part is a server that hosts Bokeh applications. Class that manages Bokeh applications is called *BokehServerThread*.

BokehServerThread inherits from class *Thread* of the Python package *threading* [19]. This is necessary to be able to run *BokehServerThread* class instance as a separate thread. By running Bokeh in separate thread from the other software we are able to communicate both with user and the SFS Cube™ device asynchronously from running tasks in backend.

BokehServerThread class constructor must be supplied with *Connection* class instance, *SFSDevice* class instance and *Database* class instance. Also, it optionally could be supplied with port to listen to. If not supplied *BokehServerThread* class instance will use port 5000 by default. This port was chosen because there are no systems that use this port for communication by default.

When *BokehServerThread* class constructor is supplied with enough objects it will proceed to creating Bokeh *Applications* for each page. Pages' contents and functionality were agreed upon with the front-end developer and the company.

Bokeh *Application* classes are supplied with *CustomHandler* instances. *CustomHandler* is a class that inherits from Bokeh class *Handler*. *Handlers* are responsible for reacting to certain server events. When one of the following events occur, server passes the event to an *Application* instance. *Application* class instance attempts to call corresponding method of *Handler* class instances that it has. The possible server events that might happen on a server and eventually are passed to *Handler* class instances are:

- Server load. This event calls *on_server_loaded* method. This event occurs when Bokeh server is started. This event handling is not implemented in

CustomHandler class since no special actions are required from the backend when Bokeh server is started.

- Server unloaded. This event calls *on_server_unloaded* method. This event occurs when Bokeh server is stopped. This event handling is not implemented in class *CustomHandler.* However, it is implemented in its child class *Controls* (this class' role will be described later in this thesis).
- Session created. This event occurs when request that contains never before used session identifier arrives on Bokeh server. This event calls *on_session_created* method. When this method is called *CustomHandler* class instance extracts username from session identifier (in accordance to agreement with front-end developer, session identifiers consist of username and the page name that user is visiting) and attempts to find a username in dictionary object that is owned by Bokeh server. If it succeeds, it loads user parameters from the dictionary and uses them for page rendering. If it does not succeed it creates new instance of *UserParams* class and records it into the dictionary under extracted username.
- Session destroyed. This event is triggered when session is destroyed by the Bokeh server. Sessions are destroyed by reaching the lifetime limit of inactivity. This event calls *on_session_destroyed* method. Method on_session_destroyed sets current *user_params* attribute of *CustomHandler* instance to *None*.

After creation of a session *Application* calls method *modify_document* of its *Handlers*. This method modifies contents of a *Document* that will be served during this session. *Document* is Bokeh class. It creates an entity that hold all the necessary information for graph plotting and/or UI elements rendering.

CustomHandler class has children that inherit its methods and also implement their unique *modify_document* method. Effects of this method are different for each page.

4.3.1 Advanced Configuration.

This page is responsible for entering advanced device configurations and displaying *Test Measurement* plot. This page is also used for calibration.

Advanced configuration page elements are generated by *AdvancedConfigPage* class that inherits from *CustomHandler* class.

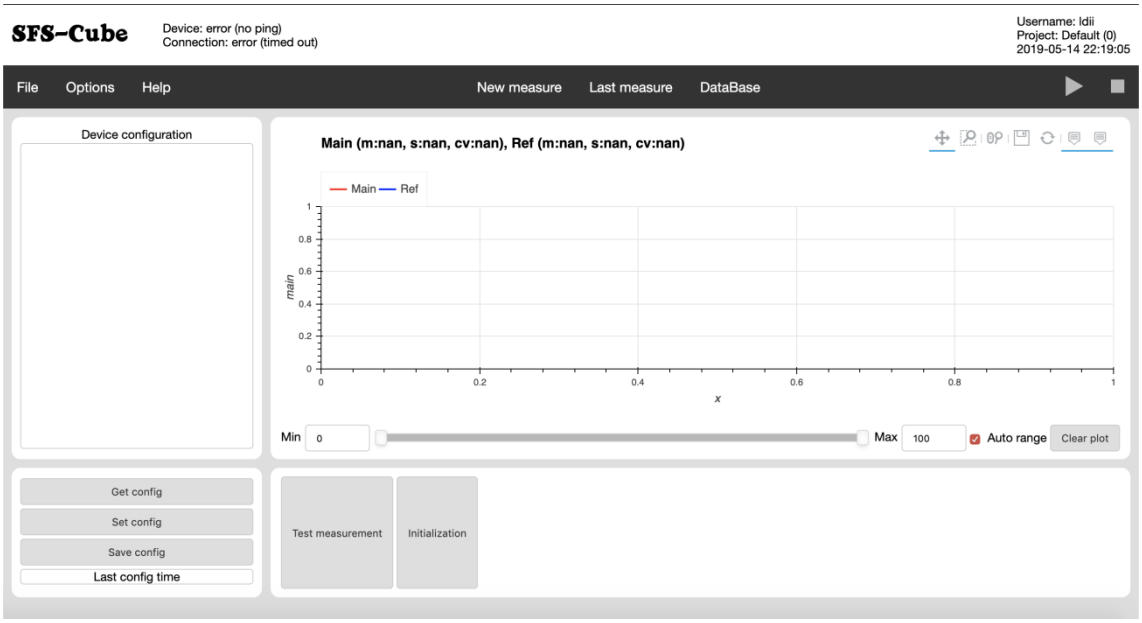

Figure 2. Advanced Configuration Page with rendered elements [19]

4.3.2 Database.

This page is responsible for displaying saved measurements, their information, generate and show their SFS image. This page is also responsible for measurement name and comment changes by user. This page offers user an interface to interact with the database.

Database page elements are generated by *DatabasePage* class that inherits from *CustomHandler* class.

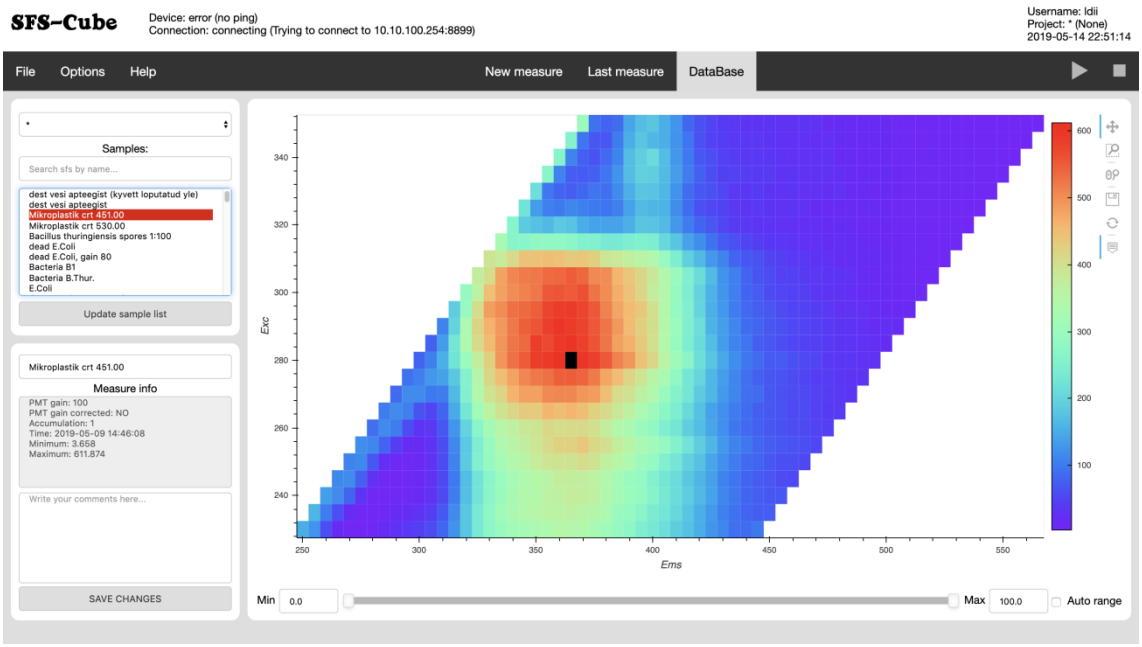

Figure 3. Database Page with rendered elements [19]

4.3.3 Last Measure.

This page is responsible for showing measurement that was received last from the SFS Cube™ device. On this page user is able to interact with SFS projection of the last measurement, give it a name and a comment and save last measurement to the database. On this page user is also able to forcibly request last stored measurement from the device.

Last measure page elements are generated by *LastMeasurePage* class that inherits from *CustomHandler* class.

4.3.4 New Measure.

This page is responsible for receiving desired measurement parameter, such as Emission range and Excitation range, Gain and Accumulation. It also has to project in real-time what resulting measurement window will look like. On this page user also is able to enter measurement name that would be assigned to the measurement and saved to the database.

Specifically, for this this window *Preset* functionality was developed. *Preset* is a shortcut for measurement windows. *Presets* could be static as well as dynamic. Static *presets* are *presets* where coordinate sliders are locked to certain positions and could not

be changed. For example, in the current version of the software SFS Standard and Chlorophyll are fixed *presets*. SFS and Rectangle are dynamic *presets*.

This page is also responsible for generating a path that would be sent to the SFS Cube™ device. A path for traversing the measurement window. As an optimal solution was chosen so called snake path. Snake path traverses every other row in the opposite direction. This path is optimal because it saves time. SFS Cube™ device has to revert motors to their home positions if we traverse every row in the same direction. This requires additional time. By changing to snake path, we reduced measurement time in half. Also snake path is relatively simple to implement. Thus, snake path consumes less client machine resources that other path optimisation algorithms.

New measure page elements are generated by *NewMeasurePage* class that inherits from *CustomHandler* class.

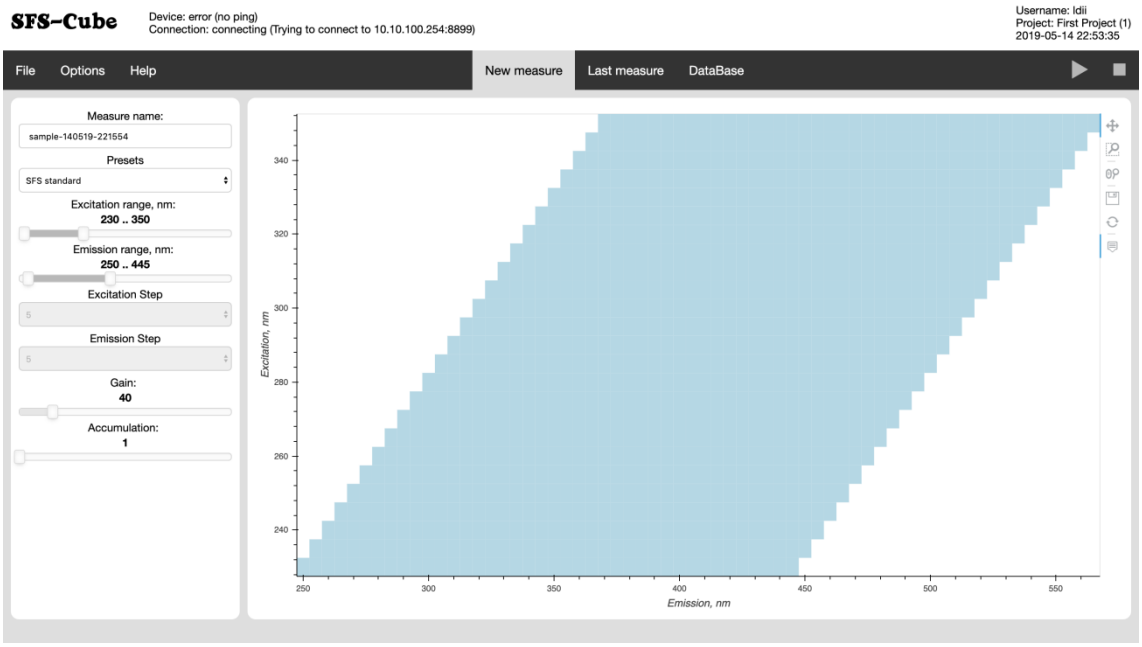

Figure 4. New Measure Page with rendered elements [19]

4.3.5 Projects.

This page presents user with an interface for managing projects.

Project is an entity that would group different data types and different instances of same data type.

In the current version of the software it is possible to only add measurements to projects. However, it is planned for the future versions that user would be able to group *presets* and settings under the project.

Project page elements are generated by *ProjectPage* class that inherits from *CustomHandler* class.

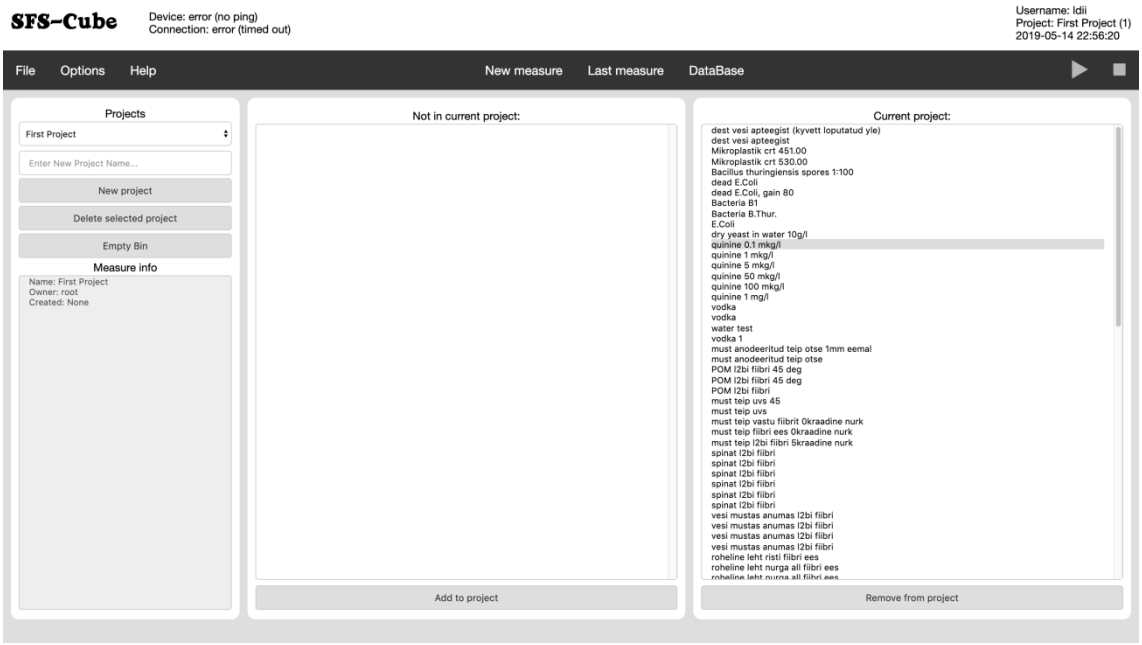

Figure 5. Project Page with rendered elements [19]

4.3.6 Controls

Controls class operates by the same principles like the other page-representing classes described before. However, *Controls* class overwrites *on_session_created* and *on_session_destroyed* methods of its parent class *CustomHandler*.

Controls class is responsible for all elements that occur on every page of the application. Those are: device control buttons Start and Stop, pop-up message card, current session information, device and connection information and measurement progress bar.

Controls class is also responsible for user parameter persistency across the pages and user data management.

Since *Controls* class instance elements are loaded on every page, except when user has logged out of the application, it is supposed that we can determine that user is using (or not using) this software from existence of the session which identifier contains this class' name. Thus, overwritten method *on_session_destroyed* saves user parameters to database so that user parameter was later loaded on the next user login. This leads to a better user experience. Also, if we save user setting to database only once, when he/she logs out we save client machine resources by now overusing writes to database file.

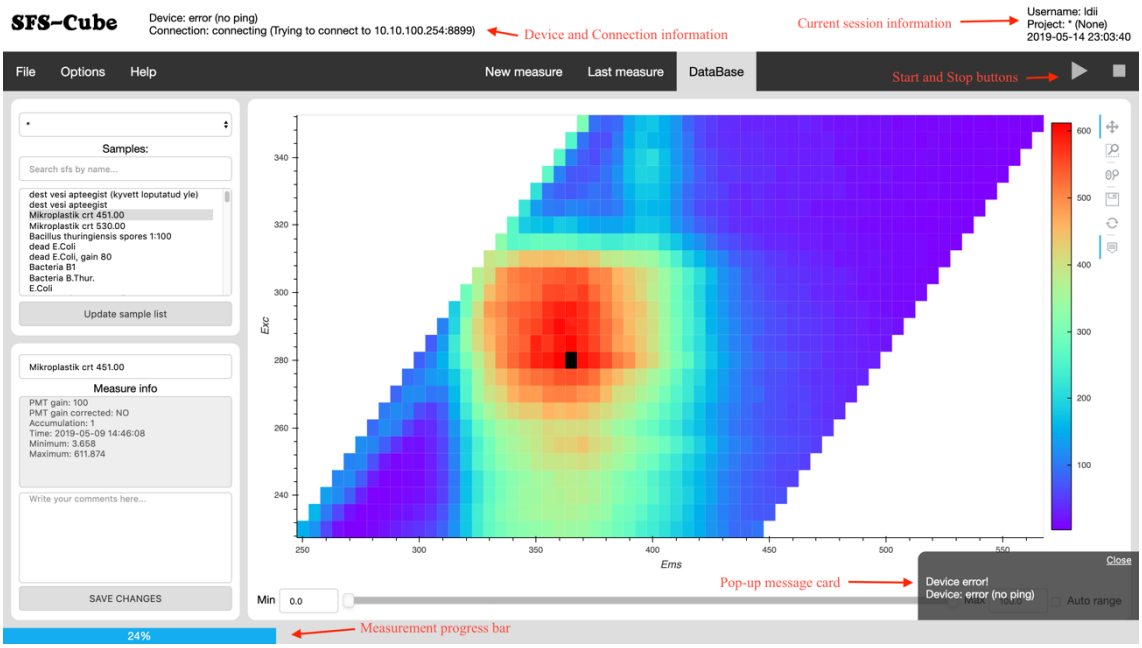

Figure 6. Database page with outlined Controls elements [19]

4.4 API

One of the company requirements was that the backend software was able to be embedded into frontend and could be further styled.

Bokeh provides an Application Programming Interface (API) that allows front-end developer to divide document into *roots*, that could be further arranged on the page. With this API development could be replaced by agreement between front-end developer and back-end developer on the naming scheme of the Bokeh elements.

After the naming scheme is decided, on the back-end part name attributes of Bokeh *Widget* class instances are set. Now front-end application can request the document for current user and break it down using those names.

5 Summary

The aim of this thesis was to develop a backend part of the application that would allow users to control and receive measurement data from SFS Cube™ device developed by LDI Innovation. Secondary aim of this work was to develop methodology of software development that would save time of product release for OEMs such as LDI Innovation.

LDI Innovation is a manufacturer of diagnostic instruments that produce large amount of data that needs to be received, managed, stored and visualised. To satisfy those needs several programming languages, database engines and data visualisation libraries were compared.

Python programming language was chosen, because of its adaptivity, portability and community, that provides many different libraries.

Storage system is implemented with SQLite database engine, that is compact, simple and fast.

For data visualisation Bokeh accompanied by Holoviews were selected. Bokeh proved to be the most customisable data visualisation engine that had a lot of options for export, most important of which is dynamic element rendering and possibility of embedding into a web page. Holoviews was used for simplification of Bokeh plot creation.

In conclusion, the main objective of this thesis was successfully accomplished. The software that was developed is currently in use by LDI Innovation and their clients, such as HTNova (China), Inov (Portugal), Gastops (Canada) and Nazarbayev University (Kazakhstan).

References

[1] A. Witzøe, “Boxes with dead fish are piled up after the algae attack,” 2019. [Online]. Available: https://salmonbusiness.com/boxes-with-dead-fish-are-piled-up-after-the-algae-attack/. [Accessed 17 05 2019].

[2] educba, “Differences Between Python vs C++,” 2019. [Online]. Available: https://www.educba.com/python-vs-c-plus-plus/. [Accessed 17 05 2019].

[3] C. Zapponi, “GitHut - Programming Languages and GitHub,” 2014. [Online]. Available: https://githut.info/. [Accessed 17 05 2019].

[4] NumPy Developers , “NumPy – NumPy,” 2005. [Online]. Available: https://www.numpy.org/. [Accessed 17 05 2019].

[5] AQR Capital Management, LLC, Lambda Foundry, Inc. and PyData Development Team, “Python Data Analysis Library — pandas: Python Data Analysis Library,” 2008. [Online]. Available: https://pandas.pydata.org/. [Accessed 17 05 2019].

[6] SQLite, “Zero-Configuration,” 2000. [Online]. Available: https://www.sqlite.org/zeroconf.html. [Accessed 17 05 2019].

[7] SQLite, “35% Faster Than The Filesystem,” 2000. [Online]. Available: https://www.sqlite.org/fasterthanfs.html. [Accessed 17 05 2019].

[8] SQLite, “SQLite: Single File Database,” 2000. [Online]. Available: https://www.sqlite.org/onefile.html. [Accessed 17 05 2019].

[9] SQLite, “SQLite Library Footprint,” 2000. [Online]. Available: https://www.sqlite.org/footprint.html. [Accessed 17 05 2019].

[10] M. Drake, “SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems,” 19 03 2019. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed 17 05 2019].

[11] A. Bonomi, “GitHub - andreax79/python-cstruct: C-style structs for Python,” 2017. [Online]. Available: https://github.com/andreax79/python-cstruct. [Accessed 17 05 2019].

[12] Python Software Foundation, “Built-in Functions — Python 3.7.3 Documentation,” 2018. [Online]. Available: https://docs.python.org/3/library/functions.html?highlight=property#property. [Accessed 17 05 2019].

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

[14] The Matplotlib development team, “Matplotlib: Python plotting — Matplotlib 3.0.3 documentation,” 2012. [Online]. Available: https://matplotlib.org/index.html. [Accessed 17 05 2019].

[15] Plotly, “Modern Analytic Apps for the Enterprise - Plotly,” 2019. [Online]. Available: https://plot.ly/. [Accessed 17 05 2019].

[16] M. Waskom, “seaborn: statistical data visualization — seaborn 0.9.0 documentation,” 2012. [Online]. Available: https://seaborn.pydata.org/. [Accessed 17 05 2019].

[17] D. Smirnov, “The Development OF Web-Based Graphical User Interface for the Original Equiment Manufacturer (LDI Innovation),” Tallinn, 2019.

[18] PyViz developers, “HoloViews — 1.12.2,” 2019. [Online]. Available: http://holoviews.org/. [Accessed 17 05 2019].

[19] Python Software Foundation, “threading — Thread-based parallelism — Python 3.7.3 documentation,” 2001. [Online]. Available: https://docs.python.org/3.7/library/threading.html. [Accessed 17 05 2019].