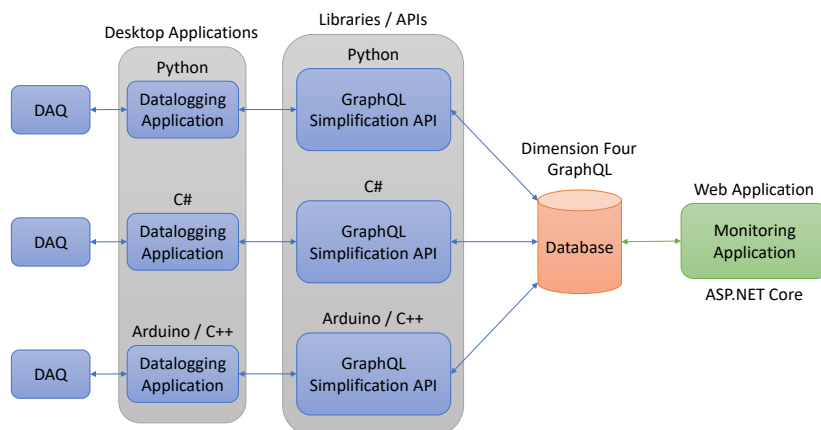


FMH606 Master's Thesis 2022  
Industrial IT and Automation

# Development of Open Source Datalogging and Monitoring Resources for IoT Platform



Håkon Helgesen

Faculty of Technology, Natural Sciences and Maritime Sciences  
Campus Porsgrunn

**Course:** FMH606 Master's Thesis 2022

**Title:** *Development of Open Source Datalogging and Monitoring Resources for IoT Platform*

**Pages:** 68

**Keywords:** <API, GraphQL, IoT, C#, Python, Arduino, Raspberry PI, DAQ>

**Student:** *Håkon Helgesen*

**Supervisor:** *Hans-Petter Halvorsen*

**External partner:** *Dimension Four*

**Summary:**

Planning, development, and partial deployment of APIs and a web-application designed to interact with a GraphQL API. The project is aimed at everyone, to remove the need for GraphQL knowledge, while simultaneously allowing the use of Dimension Four's GraphQL solution. Explanations for key aspects of the concept of "internet of things", the planning and development of the APIs and web-application, and the resulting product. Documentation on how to install and use the APIs and web-application.

# Preface

GraphQL is a query language which is becoming ever more present in the Internet of Things. For an interested developer, beginner or expert, the time to learn and understand GraphQL may not be available. Thus the project which is documented in this paper was created. This paper documents the development of APIs and a web-application. The APIs are in the form of libraries for both Python and Arduino, while the web-application is developed in ASP.net Core for data monitoring. The project came to be because the University of South-Eastern Norway wants to support local business, by using their service. Because GraphQL is not a subject in the curriculum, nor is there capacity to include it, it was suggested to streamline the integration of the service with APIs. Thus, this project was conducted and completed to circumvent the need for GraphQL knowledge. Initially planned for students, this project is for everyone interested in using the service provided by Dimension Four.

During this project, we have worked with both the University and Dimension Four, taking suggestions from both. I would like to thank Dimension Four for allowing me to carry out this project, and especially thank Daniel Warholm, for showing great interest in the project all the way from start to finish. I would also like to thank the project supervisor from USN, Hans-Petter Halvorsen, for giving suggestions, constructive criticism, and guidance throughout the project. Porsgrunn, 18th May 2022

Håkon Helgesen

# Contents

- Preface** **3**
  
- Contents** **5**
  - List of Figures . . . . . 7
  - Nomenclature . . . . . 8
  
- 1 Introduction** **9**
  - 1.1 Background . . . . . 9
  
- 2 Internet of Things** **11**
  - 2.1 MQTT . . . . . 11
  - 2.2 REST . . . . . 12
  - 2.3 GraphQL . . . . . 12
  - 2.4 ASP.NET Core . . . . . 12
  - 2.5 Thingspeak . . . . . 12
  - 2.6 Grafana . . . . . 13
  - 2.7 Github . . . . . 13
  - 2.8 IoT Maker Devices . . . . . 13
    - 2.8.1 Arduino . . . . . 13
    - 2.8.2 Raspberry PI . . . . . 14
  
- 3 Programming Languages** **15**
  - 3.1 Python . . . . . 15
  - 3.2 C# . . . . . 15
  
- 4 Dimension Four** **16**
  - 4.1 GraphQL Solution . . . . . 16
    - 4.1.1 Dimension Four Terminology . . . . . 16
  - 4.2 GraphQL Playground . . . . . 17
  
- 5 Development** **19**
  - 5.1 Planning . . . . . 19
  - 5.2 Python Development . . . . . 19
  - 5.3 Arduino Development . . . . . 25
  - 5.4 C# Development . . . . . 27

<b>6</b>	<b>Python Library</b>	<b>46</b>
6.1	Python Examples . . . . .	47
<b>7</b>	<b>Arduino Library</b>	<b>48</b>
<b>8</b>	<b>ASP.net Web-Application</b>	<b>49</b>
<b>9</b>	<b>Discussion</b>	<b>57</b>
<b>10</b>	<b>Conclusion</b>	<b>58</b>
	<b>Bibliography</b>	<b>59</b>
<b>A</b>	<b>Project Description</b>	<b>61</b>
<b>B</b>	<b>Gantt Diagram</b>	<b>64</b>

# List of Figures

1.1	.....	10
4.1	Tabs for organizing queries. ....	17
4.2	Paginated text field for querying, with query. ....	18
4.3	Headers for proving authority. ....	18
4.4	Response field with response from Create Space query. ....	18
5.1	The Create Space function in Python. ....	21
5.2	How to remove unnecessary syntax from a dictionary. ....	21
5.3	Sending Posts with Python. ....	22
5.4	The Python Create Headers function. ....	23
5.5	A function for saving data. ....	24
5.6	The Arduino Library Header file. ....	25
5.7	Passing Clients and Serials to an Arduino Library. ....	26
5.8	Establishing the query object in the Arduino Library. ....	26
5.9	Serializing and posting the query object to Dimension Four. ....	27
5.10	The Space class, it's properties, and the first half of the instancing function. It prepares sub-spaces, in case the space is a parent. ....	28
5.11	This figure depicts the second half of the Space class' instancing function. It prepares the list of points belonging to the space. ....	29
5.12	This is the Point class. ....	30
5.13	This depicts the Signal class. ....	31
5.14	This is the Space List back-end. ....	33
5.15	This depicts the HTML and javascript behind the Space List Razor-page. .	34
5.16	This is the javascript function for partial update of the points list. ....	35
5.17	This figure depicts the back-end of the Signal Display. ....	37
5.18	This is the javascript code responsible for initially drawing the plot. ....	38
5.19	This is the javascript function responsible for redrawing the plot with new data. ....	39
5.20	This figure represents the back-end of the Admin Page. ....	41
5.21	This is the HTML of the first two administrative actions, as well as the action selector. ....	42
5.22	This is the javascript behind the first two administrative actions, Create Space and Create Sub-Space. ....	43
5.23	The back-end of the Index page. ....	44

5.24	This is the javascript for declaring API credentials. . . . .	45
8.1	This is the Index page of the web-app. This is where the user presents their API credentials. . . . .	50
8.2	This is the Space List page of the web-app. Select a space to display points and signals. . . . .	51
8.3	This is the Signal Display page. Here the user may choose points and signal types to display from. . . . .	52
8.4	This is the initial Administration page. The user may select an action to perform from the list. . . . .	53
8.5	This is the Create Space action page. . . . .	53
8.6	This is the Create Sub-Space action page. The user may select a parent space from the right-hand list. . . . .	54
8.7	This is the Delete Space action page. . . . .	54
8.8	This is the Create Point action page. The user may select a parent space from the right-hand list. . . . .	55
8.9	This is the Add Signal Type action page. . . . .	55
8.10	This is the Remove Signal Type action page. . . . .	56
8.11	This is the Delete Point action page. . . . .	56

# Nomenclature

Symbol	Explanation
API	Application Programming Interface
IoT	Internet of Things
AJAX	Asynchronous Javascript And XML
UI	User Interface
UX	User Experience
GUI	Graphical User Interface
DAQ	Data Acquisition
USN	University of South-Eastern Norway
MQTT	An IoT communication protocol
GraphQL	An API query language
Python	A popular, highly readable programming language
C#	A popular, high level programming language
IT	Information Technology
UML	Unified Modelling Language
ASP.net	A popular C# framework
HTTP	HyperText Transfer Protocol



# 1 Introduction

In today's world of Industry 4.0 and internet of things, new solutions for data-collection and data-accessibility are being developed constantly. For many people, for example hobbyists and students, time may be a hot commodity and thus effective solutions become inaccessible. This project aims to increase accessibility to such a solution by establishing APIs in several popular programming languages.

## 1.1 Background

USN uses different IoT platforms today in within the IT and Automation Bachelor program and the Industrial IT and Automation Master program, both for educational and research purposes.

Dimension Four (<https://dimensionfour.io>), a local company in Grenland Norway has developed a new IoT platform, which may be relevant to use by USN in the future. The IoT platform uses MQTT and GraphQL.

To use this platform at USN, both in education and research, APIs and practical examples need to be developed for different devices (PCs with DAQ devices from National Instruments, Arduino, and Raspberry Pi) and programming platforms. The main programming environments and programming languages at USN within the Bachelor/Master programs mentioned above are Visual Studio/C#, Python, LabVIEW, and MatLAB.

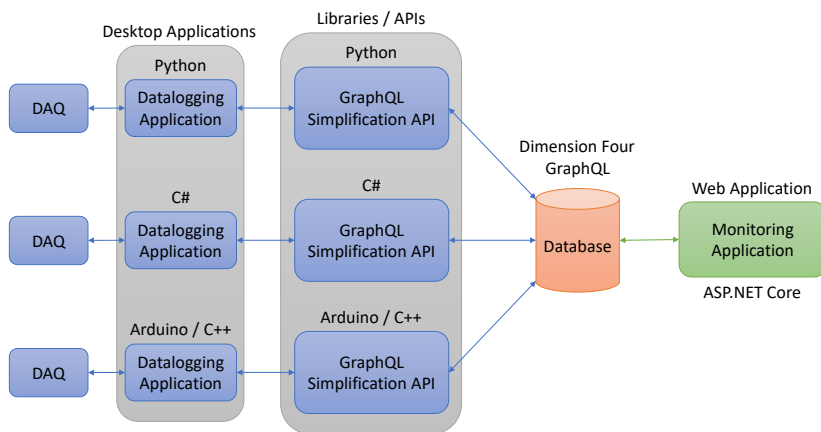


Figure 1.1:

## 2 Internet of Things

Internet of Things is a concept which is at the forefront of the Industry 4.0 revolution. At the core it is all about information, where "things" either gather or acts upon obtained information. "Things" may do both the gathering and acting. A "thing" refers to any object which can gather or act, and is able to communicate with other "things" [1]. Communication is often done over the internet, but may in some cases communicate through other devices via Bluetooth or the like.

An example of Internet of Things may be a personal weather station with a monitoring unit. The sensors may be connected together via WIFI, allowing you to observe the local weather from the kitchen while brewing coffee. In addition to updating your local monitoring unit, the sensors could communicate weather to a wider network. This would all users of the same network to observe weather from your network connected sensors, and visa versa.

Some pros of Internet of Things is that it encourages development of efficient architecture and software, due to the miniature nature of devices. The efficient nature of said software may in turn lead to a lower bar for entry into tinkering and tech development, increasing tech-literacy. In addition to efficient software, the collection of data may in turn increase the accuracy and efficiency of industrial processes, increasing productivity and lowering costs. However, mass collection of data, especially concerning end-users, is a huge privacy concern. Data is collected everywhere by everyone, and this data is often abused. While Internet of Things may bring forth easy to access tech development tools and concepts, the vast amount of IoT-devices may not encourage people without tech-interests to increase their tech-literacy. This may decrease overall tech-literacy altogether.

### 2.1 MQTT

MQTT is an OASIS standard and has quickly become the de-facto IoT communications protocol. It works on a simple publish/subscribe model, where publishers push messages to a broker and subscribers retrieves gets said messages from the broker. Due to the miniature nature of IoT, both space, processing power, and battery consumption was taken into account when MQTT was designed [2].

## 2.2 REST

REST is often misinterpreted as a protocol, but is in fact an architectural style for developing web services and APIs [3]. The REST architecture is composed of guidelines for system developments. These guidelines are designed to maximise scalability and modifiability. Software and APIs developed using REST architecture are often referred to as "RESTful" [4].

## 2.3 GraphQL

GraphQL is a query language which makes the creation of version-less APIs possible. It was developed by the company formerly known as Facebook, now Meta, in 2012. In 2015, GraphQL went open-source and is maintained by The GraphQL Foundation. GraphQL is designed to make API creation easy and lightweight, where devices can request the specific data they need from the back-end. Due to GraphQL being lightweight, it is commonly used in IoT solutions [5].

## 2.4 ASP.NET Core

ASP.NET Core is an open-source extension of the .NET framework, for developing web applications. Core aspects of ASP.NET are performance and cross-platform compatibility. As with C# and the .NET framework, ASP.NET Core is developed and maintained by Microsoft [6].

## 2.5 Thingspeak

Thingspeak is a platform designed as an IoT data-analytics solution. The platform allows for easy visualization of live and/or cloud stored data. To make data analysis easy, Thingspeak comes with MatLAB integration, allowing the user to pull data from the Thingspeak cloud, to MatLAB. Thingspeak is made and maintained by MathWorks Inc, the company behind MatLAB, as an easy solution in an ever growing IoT world [7].

## 2.6 Grafana

Grafana is an open-source application, which aims to unify data from independent sources in an orderly fashion. Grafana can access data from the user's choice of storage, be it a database or a single board computer, like the "Raspberry PI". The application was launched in 2014, and has been in continuous development since [8].

## 2.7 Github

Github is a cloud-based repository for tracking and maintaining software code. It allows developers to develop code alongside other developers, on the same project [9]. Projects may also be public, which means code is visible for everyone, or private, which means the code is only visible for select individuals. Github gives developers oversight over project code and version. A developer may choose to fork or branch code from a repository. This means creating a duplicate project, where changes may be done without affecting main project code. When changes done to the duplicate is done, it may be merged together with the original repository. This process then includes changes done to the duplicate in the original repository.

## 2.8 IoT Maker Devices

Internet of Things is not only an industrial revolution, but has also prompted the creation of several hobbyist tools. These devices may not be designed for industrial use and durability, but they make excellent tools for research and prototyping. Used in this project for testing purposes are the micro-controller board, "Arduino", and the single board computer, "Raspberry PI". These boards allow simple programming to interact with physical hardware, such as temperature sensors, and to communicate with other software, across WiFi and internet.

### 2.8.1 Arduino

Arduino is a brand of small single board micro-controllers, designed by a company in Italy with the same name. The goal of the Arduino company is to empower individuals to create electronically controlled creations and IoT devices. Arduinos are designed to be easy to use, inexpensive and open-source, both in hardware and software. This allows users to prototype, then remove non-essential components from production boards [10].

## 2.8.2 Raspberry PI

The Raspberry PI is a single board computer, running a custom made, lightweight, distribution of Linux. The computer comes with a GPIO, which can be used to interface with user designed electrical circuits and/or sensors [11]. It comes in different sizes, where the smallest lack a GUI and must be accessed through another computer. The Raspberry PI comes with several programming languages installed, the most popular and versatile of which are Python. The relatively small size and computing power makes it a popular choice as a base for IoT devices.

## 3 Programming Languages

Several programming languages has been a part of the development process for this project. Python and C# has been actively used during development, while GraphQL is the core of Dimension Four's product. This section contains the reason for creation of these languages, as well as their areas of use. While technically not a language, but a framework, ASP.NET Core is described in this section as well.

### 3.1 Python

Python is an easy to learn, high-level programming language. It has simple syntax and emphasizes code-readability. It is an interpreted language, which means it is translated to machine code during execution. Because of this, there is no compilation stage, and bug testing and fixing is therefore easier [12]. Python is open-source and the source code is maintained by the Python Foundation. Due to the OSI-approved open-source license which Python is licensed with, it is freely usable and distributable. The free distribution extends to commercial use [13].

### 3.2 C#

C# is an object-oriented, open-source and cross-platform programming language. Due to it being part of the C family of programming languages, it is similar to C, C++, and Java, thus easy to pick up if you already know one of these. It is engineered to create robust and durable applications, for the .NET framework. It was developed and released by Microsoft in 2000, along with the .NET framework [14].

## 4 Dimension Four

Dimension Four AS is a tech-company from Grenland, Norway, and was established in 2018. Dimension Four saw a need for a developer friendly API, which removed the complexity of working with an IoT back-end. Their goal is to develop a platform agnostic back-end, where companies and their developers can choose the hardware they want or need. In early 2021, Dimension Four launched their IoT platform [15].

### 4.1 GraphQL Solution

Dimension Four is a tech-company from Grenland, Norway, and is developing a GraphQL API. The aim of the API is to be both fast and scalable, being easy for the front-end to interact with. The API consists of ready-made levels of organization and functions for inserting, organizing, and retrieving desired data. GraphQL can be interacted with through HTTP and thus the Dimension Four API is indifferent to what device is sending or retrieving information, given it presents user generated credentials. In addition to the GraphQL API, Dimension Four runs an MQTT broker. The credentials for said broker can be requested from the API.

#### 4.1.1 Dimension Four Terminology

In their GraphQL Solution, Dimension Four has established several points of organization. Spaces, Points, and Signals form the structure of which data is sorted. Spaces acts as folders, while points and signals represents devices and measurements. Spaces can be used to represent a general collection of other spaces or points, locations, or other features points may have in common. As an example, a hierarchy of spaces may be "Factory 01 - Production Hall 04 - Process 02". Since points represent devices, they are not allowed to have sub-points. A point may be named "TMP36 01", if a system has one or more TMP36 temperature sensors. If a point is able to collect more than one type of measurement, the type and unit is stored with the measured value in a signal. As with points, signals can not have sub-signals. If a point / device is measuring several unit types at once, they are to be stored as different signals, under the same point. Each space, point, and signal have their own ID, which can be used to access information about it.



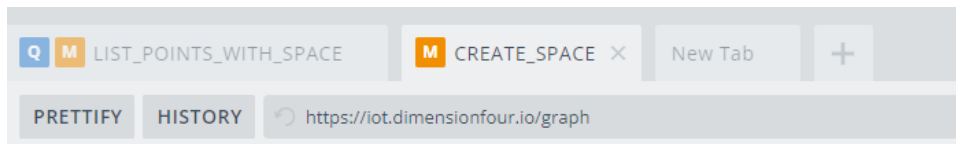


Figure 4.1: Tabs for organizing queries.

## 4.2 GraphQL Playground

Dimension Four’s GraphQL service can be accessed directly through a web-browser. Accessing the URL: `iot.dimensionfour.io/graph`, loads a web-application the company have named “Playground”. Giving an overview of the playground, and starting at the top, there are individual tabs for organizing sessions, in case the user needs to access the API with different tenants and/or headers, or to organize their queries. Beneath this, a button for prettifying the queries, showing history query, and a URL-bar, displaying the client URL. Figure 4.1 depicts these UI elements.

Beneath this, the application is split into a left- and right-hand side. The left-hand side is for querying the API, while the right-hand side displays the API response. The left-hand, query side, contains a large field with paginated lines, used for the actual queries. This field can contain multiple queries, if needed. This field is depicted by figure 4.2. In the bottom of the left-hand side, is two tabs, query variables and HTTP headers. The query variables tab is used to store variables for querying, like space IDs or point IDs. Variables which are often used, but hard to remember. The second tab, HTTP headers, is used to send credentials, like tenant ID and tenant key. This area is depicted by figure 4.1.

In the middle of the web-application, between the query-field and the response-display, sits a “play button”. This button is used to send the queries in the query-field to the API. If there are multiple queries in the query-field, a list of the queries are shown, and the user is asked to select one. The selected query is then sent to the API. The field on the right-hand shows responses from the GraphQL API when queried. This field along with a response is depicted by figure 4.4. Finally, on the far-right edge of the web-application, there are two tabs containing guides and templates for queries, if this is needed.

```
1 mutation CREATE_SPACE {
2   space {
3     create(input: {name: "Production Hall 2" }) {
4       id
5       name
6     }
7   }
8 }
```

Figure 4.2: Paginated text field for querying, with query.

```
QUERY VARIABLES HTTP HEADERS (2)
1 { "x-tenant-id": " ", "x-tenant-key": " " }
```

Figure 4.3: Headers for proving authority.

```
{
  "data": {
    "space": {
      "create": {
        "id": "6282675b7a4250b91ea4f0e0",
        "name": "Production Hall 2"
      }
    }
  }
}
```

Figure 4.4: Response field with response from Create Space query.

## 5 Development

This chapter concerns the planning and development of Python and Arduino Libraries/APIs, as well as the monitoring web-application. It is divided into sub-chapters, starting with 5.1, then continuing with the development in the order the different project parts were developed. First 5.2, then 5.3, ending on 5.4.

### 5.1 Planning

The development of this project has gone through the phases of Unified process, where each individual part is planned and developed in its own iteration. The inception of the project came from the University of Southeastern Norway's desire to use locally developed IoT tools. From this, several uses were elaborated upon, such as which programming languages needed an easy way to access this tool. As one of the first steps during planning, a system sketch was drawn. Figure 1.1 depicts the system sketch. To keep track of planned features of the different libraries, UML class diagrams were to be created for each library. UML diagrams were only created for the Python Library, while the rest of the project were planned using a Kanban Board. The following programming language sections appear in the order they were developed. To organize code, a GitHub repository was established and is available at: "<https://github.com/USN-Helgesen/df-api-and-monitor>".

### 5.2 Python Development

While planning the Python library, seven functions were drawn up in a UML class diagram. These were functions for creating spaces, points, and signals, as well as listing these, and a final one for retrieving the latest signal from a specific point. These seven functions were then added to the main script of the library, ready to be developed. Each of these functions contain a string called "query", which contains the GraphQL query for the Dimension Four back-end, and a Python dictionary called "json". This dictionary consists of the "query" string and the variables sent to the function.

The first developed function was "create\_space", which is used to create spaces in the back-end. To send the create space query to Dimension Four, an eighth function was

created. "send\_post" is taken from the Dimension Four's example. This function receives the target domain, json, and headers as inputs and sends a request to the target. It was shaven down to prevent it from printing the received data to the terminal, and returns the data to the call instead. During the initial testing of the "create\_space" function, permission denial became a problem. The tenant token which was established for development purposes had every administrative permission, yet the back-end denied the creation of a space. This was an internal error at Dimension Four, which was fixed in a matter of days. This was the only major hurdle during the Python library development. Figure 5.1 depicts the Create Space function, which adheres to the general structure of all query functions. The query string is defined, then combined with variables to create a query object in the "json" variable. The query object is then sent along with the target and headers, to the Send Post function, figure 5.3. The target variable passed through the function is a string variable equal to the Dimension Four playground URL, "https://iot.dimensionfour.io/graph". The Send Post function, inspired by Dimension Four's example [16], sends the query object as an HTTP request to the target, with the headers credentials, and check for any bad requests or other problems. If the query was successful, it returns the query object.

```

def create_space(space_name, target, headers):
    """Creates a Space in the Dimension Four back-end.

    :param space_name: str
    :param target: str
    :param headers: dict
    :return: None
    """

    query = """mutation CREATE_SPACE(
    |   $spaceName: String!
    | ) {
    |   space {
    |     create(input: { name: $spaceName}) {
    |       id
    |       name
    |     }
    |   }
    | }"""

    json = {
    |   "query": query,
    |   "variables": {
    |     "spaceName": str(space_name),
    |   },
    | }
    ans = send_post(target, json, headers)

```

Figure 5.1: The Create Space function in Python.

```

raw_signal = [ans['signalsConnection']['edges'][0]['node']['data']['rawValue'],
|   ans['signalsConnection']['edges'][0]['node']['timestamp']]
return raw_signal

```

Figure 5.2: How to remove unnecessary syntax from a dictionary.

```

def send_post(target, json, headers):
    """Sends request to Dimension Four back-end.

    :param target: str
    :param json: dict
    :param headers: dict
    :return: dict
    """
    try:
        res = requests.post(target, json=json, headers=headers)
    except Exception as exception:
        print("Error!: " + exception)

    if res.status_code != 200:
        print("Send error!")
        print(res.text)

    else:
        res_json = res.json()
        if "errors" in res_json.keys():
            print("Query error!")
            print(res_json["errors"])
        else:
            print("Success!")
            return res_json["data"]

```

Figure 5.3: Sending Posts with Python.

During the development and testing of the initial seven functions, a ninth function for creating a header dictionary was created to cut down on the use of dictionaries. Python dictionaries share a common structure with json. This "create\_header", figure 5.4, function takes the tenant ID string and the tenant token string as inputs, and returns a header dictionary. The "retrieve\_latest\_signal" function contains an additional Boolean, which is false by default. This feature is developed to alter the returned data. A signal return query from Dimension Four is very unwieldy, as shown by figure 4.4 and is thus de-nestled to a list containing the data value and timestamp. If the full query is desired, simply turn this Boolean true. Figure 5.2 depicts the process of de-nestling the query. The final function planned and created was the "save\_data" function. This simply takes the data and saves it as either a .csv or .json file. The final step of the development process is uploading the project to PYPI. Doing this allows for the installation of the library through the Python package manager, "pip". Most guides were somewhat outdated and the setuptools web-page was hard to read. The overall process is not too advanced. The first step is to pip install "setuptools", which is used while creating a "setup.py" file for the project. The setup.py script is used to structure metadata about the package, like the name, author and version. Two additional files are required to build the package: "\_\_init\_\_.py", which can be empty, and "pyproject.toml", which is needed to select the desired build tool.

The second package needed to be pip installed is "build". When this is installed, change directory to where the setup.py file is located. Then type "py -m build" to build the package. When the building is finished, two additional files are created: a ".tar.gz" and a ".whl". These files are the ones being uploaded to PYPI. This is the final steps of getting the package available on pip. First, register a user on PYPI.org. This user is needed when uploading the package. Next, pip install "twine". Twine is used to upload to PYPI. When in the same directory as setup.py, execute "twine upload dist/\*" in the terminal. When requested, enter your PYPI username and password. If successful, the package should now be installable with pip install. The Dimension Four API are now installable

```
def create_header(tenant_id, tenant_key):
    """Creates a Header for accessing the Dimension Four back-end.

    :param tenant_id: str
    :param tenant_key: str
    """
    headers = {
        "x-tenant-id": tenant_id,
        "x-tenant-key": tenant_key,
    }
    return headers
```

Figure 5.4: The Python Create Headers function.

```
def save_data(data, is_json=False):
    """Saves data in either CSV or Json format.

    :param data: list or dict
    :param is_json: bool
    :return: None
    """
    if is_json == True:
        with open('data.json', 'a', encoding='utf-8') as file:
            json.dump(data, file, ensure_ascii=False, indent=4)
    else:
        with open('data.csv', 'a', encoding = 'utf-8', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(data)
```

Figure 5.5: A function for saving data.

with "pip install dfapi".



## 5.3 Arduino Development

As the Arduino is a programmable single-board micro-controller with very limited memory, the Arduino-Library should be short and contain only needed functions. Organizing queries, such as creating points or spaces, and listing these, are less useful for a device which is mostly used for data gathering or system controlling. From the planning phase, three core functions were deemed necessary. A function for retrieving signals, a function for posting signals, and functionality to query the Dimension Four GraphQL back-end.

The first development iteration concerned the retrieving signals function. The function started as a test of the Dimension Four Opla example. It is a simple example, using the `WifiClient` and `ArduinoHttpClient` libraries to send HTTP requests. The first hurdle of turning said example into a library function, was passing the HTTP-client and `Serial` from the main sketch to the library. As seen in figure 5.6, references for a serial stream and HTTP client is created under the private section. Then, these are passed to the library in the `InitStream` function, depicted by figure 5.7. The second iteration of the development phase concerned the posting function. As most of the difficulties had been overcome in the first iteration, the posting function was made by modifying the example given by Dimension four [16], changing the GraphQL query and adding necessary variables to the request. Figure 5.8 depicts the first half of the read function, where the query string and variables are combined into a query object. Figure 5.9 illustrates further, where the query object is serialized and sent to Dimension Four. The Arduino Library was tested on both an Arduino MKR 1010 and an Arduino Uno WiFi. However, when tested with several regular Arduino Unos using an Ethernet shield, the sketch did not run properly. The sketch would compile and upload to the device, but the Arduino would freeze at random steps in the startup function.

```
6 #ifndef DimensionFourApi_h
7 #define DimensionFourApi_h
8
9 #include "Arduino.h"
10 #include "ArduinoJson.h"
11 #include "ArduinoHttpClient.h"
12
13 class DimensionFourApi{
14 private:
15     Stream * printer;
16     HttpClient * httpClient;
17 public:
18     DimensionFourApi();
19     void InitStream(Stream *print, HttpClient *client);
20     float ReadLatestSignal(const char* PointId, const char* TenantId, const char* TenantToken, const char* Server);
21     void PostSignal(float Signal, char* timestamp, const char* PointId, const char* TenantId, const char* TenantToken, const char* Server);
22 };
23
24 #endif
```

Figure 5.6: The Arduino Library Header file.

```

void DimensionFourApi::InitStream(Stream *print, HttpClient *client){
    printer = print;
    httpclient = client;
}

```

Figure 5.7: Passing Clients and Serials to an Arduino Library.

```

float DimensionFourApi::ReadLatestSignal(const char* pointId, const char* tenantId, const char* tenantToken, const char* server){
    String query = R"("""(
        query LATEST_SIGNALS(
            $pointId: String!
        )
        ){
            signalsConnection(
                where: {pointId: {_EQ: $pointId}}
                paginate: {last:1}
            ){
                edges {
                    node {
                        id
                        timestamp
                        createdAt
                        type
                        unit
                        pointId
                        data {
                            numericValue
                            rawValue
                        }
                    }
                }
            }
        }
    )""";

    String postData;
    DynamicJsonDocument doc(2048);

    doc["query"] = query;

    JsonObject variables = doc.createNestedObject("variables");
    variables["pointId"] = pointId;
}

```

Figure 5.8: Establishing the query object in the Arduino Library.

```

serializeJson(doc, postData);
debugJson(doc, Serial);
debugln("Contacting Server");
httpClient->beginRequest();
httpClient->post("/graph");
httpClient->sendHeader(HTTP_HEADER_CONTENT_TYPE, "application/json");
httpClient->sendHeader(HTTP_HEADER_CONTENT_LENGTH, postData.length());
httpClient->sendHeader("x-tenant-id", tenantId);
httpClient->sendHeader("x-tenant-key", tenantToken);
httpClient->endRequest();
httpClient->print(postData);

int httpCode = httpClient->responseStatusCode();
debugln(httpCode);
if (httpCode > 0) {
    debug("[HTTPS] POST... code: ");
    debugln(httpCode);

    if (httpCode == 200) {
        String payload = httpClient->responseBody();
        debugln(payload);
        DynamicJsonDocument response(1024);
        deserializeJson(response, payload);
        auto signal = response["data"]["signalsConnection"]["edges"][0]["node"]["data"]["rawValue"].as<float>();
        return signal;
    } else {
        debugln("[HTTPS] POST... failed");
        String payload = httpClient->responseBody();
        debugln(payload);
    }
}
}
}

```

Figure 5.9: Serializing and posting the query object to Dimension Four.

## 5.4 C# Development

The ASP.NET web application is the project's monitoring software. The main aspect of the web application is to display information from Dimension Four in an orderly fashion, thus retrieving queries are deemed most important during planning. Planned features for the application are: retrieve spaces and list them, order the spaces and sub-spaces accordingly in said list, display information about single spaces, such as name, ID, sub-spaces and points, display retrieved signals from specific point in a graph, display multiple graphs, create space, create, point, create signal.

The first iteration of the ASP.NET development phase, concerned the retrieval of spaces, as well as listing these with their points. There is not a lot of easily accessible information on querying a GraphQL back-end from ASP.NET, and a lot of the information concerns the creation of a GraphQL back-end. Most of these use GraphQL libraries from the NuGet package manager. After some research, a solution, given by Mauro Petrini [17], on YouTube. A simple query using HTTP requests and Newtonsoft serializer. The tenant token and key were easily added to the HTTP client in the startup class.

To make sure the code was structured in an orderly fashion, all code concerning the querying of the GraphQL back-end were placed in the same class. As queries became useful, functions for sending these to Dimension Four, were developed on the go. To organize and store the responses from Dimension Four, individual classes for Spaces,

```

public class Space
{
    private readonly DFourConsumer _consumer;
    4 references
    public string id { get; set; }
    4 references
    public string name { get; set; }
    3 references
    public List<Point> points { get; set; }
    6 references
    public List<Space> subspaces { get; set; }
    3 references
    public Space(JToken json, DFourConsumer consumer)
    {
        _consumer = consumer;
        id = (string)json["id"];
        name = (string)json["name"];
        subspaces = new List<Space>();
        if (json["children"].HasValues) {
            foreach (var jSpace in json["children"]["edges"])
            {
                if (jSpace != null)
                {
                    var firstNode = jSpace.First;
                    if (firstNode != null)
                    {
                        var secondNode = firstNode.First;
                        if (secondNode != null)
                        {
                            Space space = _consumer.GetSpace((string)secondNode["id"]).Result;
                            subspaces.Add(space);
                        }
                    }
                }
            }
        }
    }
}

```

Figure 5.10: The Space class, its properties, and the first half of the instancing function. It prepares sub-spaces, in case the space is a parent.

Points, and Signals were established. These classes are represented by figures 5.10, 5.11, 5.12, and 5.13. These objects had variables and functions added as the functionality of the software increased. As an example, spaces were originally developed with two strings and a list of points, but as functionality grew, a list for sub-spaces were added too.

```
points = new List<Point>();
foreach (var jPoint in json["points"]["edges"])
{
    if (jPoint != null)
    {
        var firstNode = jPoint.First;
        if (firstNode != null)
        {
            var secondNode = firstNode.First;
            if (secondNode != null)
            {
                Point point = new Point(secondNode);
                points.Add(point);
            }
        }
    }
}
}
```

Figure 5.11: This figure depicts the second half of the Space class' instancing function. It prepares the list of points belonging to the space.

```

public class Point
{
    2 references
    public string id { get; set; }
    2 references
    public string name { get; set; }
    4 references
    public List<string> types { get; set; }
    2 references
    public Point(JToken json)
    {
        id = (string)json["id"];
        name = (string)json["name"];
        types = new List<string>();
        var metadata = json["metadata"];
        if (metadata != null && metadata["types"] != null)
        {
            foreach(string signalType in metadata["types"])
            {
                types.Add(signalType);
            }
        }
    }
}

```

Figure 5.12: This is the Point class..

```

public class Signal
{
    1 reference
    public string id { get; set; }
    1 reference
    public string type { get; set; }
    1 reference
    public string unit { get; set; }
    1 reference
    public float value { get; set; }
    1 reference
    public DateTime timestamp { get; set; }
    1 reference
    public Signal(JToken json)
    {
        id = (string)json["id"];
        unit = (string)json["unit"];
        type = (string)json["type"];
        timestamp = (DateTime)json["timestamp"];
        value = (float)json["data"]["rawValue"];
    }
}

```

Figure 5.13: This depicts the Signal class.

The first Razor-page to be developed, were the display for spaces and their points. The list was animated with AJAX and javascript, so when a space were clicked, the list would display the space's points. Clicking a point would redirect the page to a display for a point's signals. The first iteration of the spaces-list were functional, but deemed unorganized, as it did not order sub-spaces under their respective parents.

Thus, a recursive list was proposed and implemented. A list of spaces were added to the Space class, while the list were created with Razor syntax on the page. Points were moved out of the list, into its separate display on the right-hand side of the page. Double-clicking a space would now bring to to the Signal display, to make the display of signals from related points easier.

The Points-list is updated using requests to the C# back-end via AJAX. The back-end of the page is represented by figure 5.14, while figure 5.15 and 5.16 represents the HTML and javascript of the front-end. The function in figure 5.16 uses an AJAX request to request the points belonging to particular space. On a successful request, the points are sorted alphabetically and stored in a HTML list.



```

public class SpacesListModel : PageModel
{
    private readonly DFourConsumer _consumer;
    public Space space;
    public List<Space> spaces = new List<Space>();
    0 references
    public SpacesListModel(DFourConsumer consumer)
    {
        _consumer = consumer;
    }
    0 references
    public void OnGet()
    {
        PrepSpaces();
    }
    1 reference
    public List<Space> PrepSpaces()
    {
        _consumer.EstablishCredentials();
        spaces = _consumer.GetTopLevelSpaces().Result;
        return spaces;
    }
    0 references
    public JsonResult OnGetSpace(string id)
    {
        _consumer.EstablishCredentials();
        space = _consumer.GetSpace(id).Result;
        return new JsonResult(space);
    }
}

```

Figure 5.14: This is the Space List back-end.

```

<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
</head>
<body>
  <h1>Spaces and Points</h1>
  <div class="row">
    <div class="column">
      <h3>Spaces</h3>
      @foreach (var item in Model.spaces)
      {
        SubList(item);
      }
    </div>

    <div class="column">
      <h3>Points</h3>
      <ul class="points_list" id="points_list">
      </ul>
    </div>
  </div>
</body>
</html>
<script>
$(document).ready(function () {
  $('<div class="space_row">').nextUntil('<tr class="space_row">').slideToggle(10);
  $('<div class="space_row">').click(function () {
    var spaceId = $(event.target).attr('id');
    fillPointsLists(spaceId);
    $(this).nextUntil('<tr class="space_row">').slideToggle(400);
  });
  $('<div class="space_row">').dblclick(function (event) {
    var id = $(event.target).attr('id');
    redirectPage(id);
  });
});
function redirectPage(id) {
  var base_url = window.location.origin;
  var path = "/SignalDisplay?id=" + id;
  window.location.href = base_url + path;
}
}

```

Figure 5.15: This depicts the HTML and javascript behind the Space List Razor-page.

```

function fillPointsLists(id) {
  if (id != null) {
    $.ajax({
      type: 'GET',
      dataType: 'json',
      contentType: 'application/json',
      data: {
        id: id
      },
      url: '/SpacesDisplay?handler=Space',
      success: function (result) {
        var space = result;
        var points = space.points;
        var pointsList = document.getElementById("points_list");
        var first = pointsList.firstChild;
        while (first) {
          first.remove();
          first = pointsList.firstChild;
        }
        for (var i = 0; i < points.length; i++) {
          var point = points[i];
          var li = document.createElement("li");
          li.value = point.id;
          li.innerHTML = point.name;
          pointsList.appendChild(li);
        }
        var options = $("#nameList option");
        options.detach().sort(function (a, b) {
          var at = $(a).text();
          var bt = $(b).text();
          return (at > bt) ? 1 : ((at < bt) ? -1 : 0);
        });
        options.appendTo("#nameList");
      },
      error: function (xhr, status, error) {
        var err = eval("(" + xhr.responseText + ")");
        alert(err.Message);
      }
    });
  }
}
</script>

```

Figure 5.16: This is the javascript function for partial update of the points list.

The second Razor-page to be developed, were the Signal display. This page uses the Google Charts web-service to draw and display data. The first iteration sent a static paginate, type and point-id to the C# back-end, which in turn queried the GraphQL API. The response were then used to redraw the signals on the Google chart. This was done as a proof of concept, and the next step was to repeat the process, to get a continually updating display of data. When the graph was fully functional, user specified parameters needed to be implemented, such as signal-type, number of signals to display and which point the signals are related to. The paginate was designed as a drop-down menu with a value of 5 to 100, in increments of 5. Signal-type and point-id was text-boxes. The point-id box would auto-fill if redirected from a point in the space list. To increase the user-experience of the software, these were changed to lists, as memorizing point-id and relating types were deemed infeasible. The graph is now accessible through each space, and the space's points are listed in a drop-down menu. To handle signal-types, types are added to point metadata. The metadata is fetched, the types are extracted and displayed in a drop-down menu. Finally a checkbox is added to make the graph auto update. Figure 5.17 is the Signal Display back-end. Figure 5.18, is the javascript responsible for drawing the initial shape of the graph, lines, size, labels, etc. Figure 5.19, is responsible for redrawing the data in the graph, and does so by requesting data from the C# back-end via AJAX requests.

```

public class SignalDisplayModel : PageModel
{
    private readonly DFourConsumer _consumer;
    public Space space;
    public Point point;
    public List<Point> points = new List<Point>();
    public List<Signal> signals = new List<Signal>();
    0 references
    public SignalDisplayModel(DFourConsumer consumer)
    {
        _consumer = consumer;
    }
    0 references
    public void OnGet()
    {
    }
    0 references
    public JsonResult OnGetSpace(string id)
    {
        _consumer.EstablishCredentials();
        space = _consumer.GetSpace(id).Result;
        return new JsonResult(space);
    }
    0 references
    public JsonResult OnGetPoint(string id)
    {
        _consumer.EstablishCredentials();
        point = _consumer.GetPoint(id).Result;
        return new JsonResult(point);
    }
    0 references
    public JsonResult OnGetUpdateSignals(string id, string type, int paginate)
    {
        _consumer.EstablishCredentials();
        signals = _consumer.GetSignals(id, type, paginate).Result;
        return new JsonResult(signals);
    }
}

```

Figure 5.17: This figure depicts the back-end of the Signal Display.

```

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
  google.charts.load('current', { 'packages': ['corechart'] });
  google.charts.setOnLoadCallback(drawChart);

  function drawChart() {
    var data = new google.visualization.DataTable();
    data.addColumn(['datetime', 'Time']);
    data.addColumn('number', 'Value');

    var options = {
      title: 'Signals',
      curveType: 'function',
      pointsVisible: true,
      lineWidth: 3,
      legend: 'none',
      hAxis: { title: 'Time' },
      vAxis: { title: 'Data' },
      width: '100%',
      height: '100%',
      chartArea: { width: '85%', height: '75%' }
    };

    var chart = new google.visualization.LineChart(document.getElementById('curve_chart'));

    chart.draw(data, options);
  }

```

Figure 5.18: This is the javascript code responsible for initially drawing the plot.

```

function updateChart(id, type, paginate) {
  if (id != null) {
    $.ajax({
      type: 'GET',
      dataType: 'json',
      contentType: 'application/json',
      data: {
        id: id,
        type: type,
        paginate: paginate
      },
      url: '/SignalDisplay?handler=UpdateSignals',
      success: function (result) {
        var data = new google.visualization.DataTable();
        var items = result;
        data.addColumn(['datetime', 'Time']);
        data.addColumn('number', 'Value');

        for (var i = 0; i < items.length; i++) {
          var item = items[i];
          data.addRow([item.timestamp, Number(item.value)]);
        }

        var options = {
          title: 'Signals',
          curveType: 'function',
          pointsVisible: true,
          lineWidth: 3,
          legend: 'none',
          hAxis: { title: 'Time' },
          vAxis: { title: 'Data' },
          width: '100%',
          height: '100%',
          chartArea: { width: '85%', height: '75%' }
        };

        var chart = new google.visualization.LineChart(document.getElementById('curve_chart'));
        chart.draw(data, options);
      },
      error: function (xhr, status, error) {
        var err = eval("(" + xhr.responseText + ")");
        alert(err.Message);
      }
    });
  }
}

```

Figure 5.19: This is the javascript function responsible for redrawing the plot with new data.

The final Razor-page of the web-application is the administration page. This page is used to create or delete spaces, sub-spaces and points, or add and remove point types from the metadata. The page contains a drop-down menu, which is used to select the desired action. Selecting an action then reveals hidden HTML with action-related inputs. Selecting a different action hides the HTML and reveals the HTML relating to the newly selected action. All actions work on the same principle: on a button click, the page takes data from the relating text-boxes, drop-down menus, and labels, and sends a request to the C# back-end. In turn, the C# back-end queries the GraphQL API. Figure 5.20 depicts the structure of the administration page back-end, including several functions. These functions are triggered via AJAX requests from the front-end. Figure 5.21 depicts the HTML relating to the first two administrative actions, Create Space and Create Sub-Space. Figure 5.22 is the javascript and AJAX belonging to these actions. Every administrative action has it's own javascript function and AJAX request.

Up until this point in the development, the tenant ID and tenant key used for authentication at Dimension Four has been initiated in the web application `setup.cs` file. This means the end-user have to modify the source code of the web app to access their stored spaces, points, and signals. To increase UX, session variables was added to store the ID and key for use with the HTTP client. The index page of the web application was modified with inputs for ID and key, allowing the end-user to declare their own credentials. Figure 5.23 depicts the back-end of the Index page, where the user submitted API credentials are declared as session variables. Figure 5.24 is the javascript and AJAX request responsible of transporting the API credentials from the front-end to the back-end.



```

public class AdminPanelModel : PageModel
{
    private readonly DFourConsumer _consumer;
    public List<Space> spaces = new List<Space>();
    public List<string> typeList = new List<string>();
    public Space space;
    public Point point;
    0 references
    public AdminPanelModel(DFourConsumer consumer)
    {
        _consumer = consumer;
    }
    0 references
    public void OnGet()
    {
        PrepSpaces();
    }
    1 reference
    public List<Space> PrepSpaces()
    {
        _consumer.EstablishCredentials();
        spaces = _consumer.GetTopLevelSpaces().Result;
        return spaces;
    }
    0 references
    public JsonResult OnGetSpace(string id)
    {
        _consumer.EstablishCredentials();
        space = _consumer.GetSpace(id).Result;
        return new JsonResult(space);
    }
    0 references
    public JsonResult OnGetCreateSpace(string spaceName, string parentId)
    {
        _consumer.EstablishCredentials();
        if(parentId == null)
        {
            parentId = "";
        }
        bool successState = _consumer.CreateSpace(spaceName, parentId).Result;
        return new JsonResult(successState);
    }
    0 references
    public JsonResult OnGetCreatePoint(string pointName, string parentId)
    {
        _consumer.EstablishCredentials();
        bool successState = _consumer.CreatePoint(pointName, parentId).Result;
        return new JsonResult(successState);
    }
}

```

Figure 5.20: This figure represents the back-end of the Admin Page.

```

<h1>Admin Panel</h1>
<p>Please select an action from the menu:</p>
<select id="actionSelect" name="actionSelect">
  <option style="display:none" disabled selected>-- Select Action --</option>
  <option value="createSpace">Create Space</option>
  <option value="createSubSpace">Create Subspace</option>
  <option value="deleteSpace">Delete Space</option>
  <option value="createPoint">Create Point</option>
  <option value="updateAddTypePoint">Add Signal Type to Point</option>
  <option value="updateRemoveTypePoint">Remove Signal Type from Point</option>
  <option value="deletePoint">Delete Point</option>
</select>
<div id="createSpaceDiv" style="display: none;">
  <h4>Create Space</h4>
  <label>Space Name:</label>
  <input type="text" id="createSpaceInput" name="createSpaceInput" /><br />
  <button id="createSpaceButton" onclick="createSpace()">Create</button>
</div>
<div id="createSubSpaceDiv" style="display: none;">
  <h4>Create Subspace</h4>
  <div class="row">
    <div class="column" style="padding-left:50px">
      <label>Current Selected Parent Space:</label>
      <label id="subSpaceParentNameLabel"></label>
      <label id="subSpaceParentIdLabel" style="display: none;"></label><br />
      <label>Sub Space Name:</label>
      <input type="text" id="createSubSpaceInput" name="createSubSpaceInput" /><br />
      <button id="createSubSpaceButton" onclick="createSubSpace()">Create</button>
    </div>
    <div class="column" style="padding-left:50px">
      <p>Please select a parent space from the list:</p>
      <foreach (var item in Model.spaces)>
        <SubList(item);>
      </foreach>
    </div>
  </div>
</div>
</div>

```

Figure 5.21: This is the HTML of the first two administrative actions, as well as the action selector.

```

function createSpace() {
    var spaceName = document.getElementById("createSpaceInput").value;
    if (spaceName != null) {
        var parentId = "";
        $.ajax({
            type: 'GET',
            dataType: 'json',
            contentType: 'application/json',
            data: {
                spaceName: spaceName,
                parentId: parentId
            },
            url: '/AdminPanel?handler=CreateSpace',
            success: function (result) {

            },
            error: function (xhr, status, error) {
                var err = eval("(" + xhr.responseText + ")");
                alert(err.Message);
            }
        });
    }
}

function createSubSpace() {
    var spaceName = document.getElementById("createSubSpaceInput").value;
    var parentId = document.getElementById("subSpaceParentIdLabel").innerHTML;
    if (spaceName != null && parentId != null) {
        $.ajax({
            type: 'GET',
            dataType: 'json',
            contentType: 'application/json',
            data: {
                spaceName: spaceName,
                parentId: parentId
            },
            url: '/AdminPanel?handler=CreateSpace',
            success: function (result) {

            },
            error: function (xhr, status, error) {
                var err = eval("(" + xhr.responseText + ")");
                alert(err.Message);
            }
        });
    }
}

```

Figure 5.22: This is the javascript behind the first two administrative actions, Create Space and Create Sub-Space.

```
public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;

    0 references
    public IndexModel(ILogger<IndexModel> logger)
    {
        _logger = logger;
    }

    0 references
    public void OnGet()
    {
    }

    0 references
    public void OnGetCreateCredentials(string tenantId, string tenantKey)
    {
        Console.WriteLine(tenantId);
        Console.WriteLine(tenantKey);
        HttpContext.Session.SetString("Tenant Id", tenantId);
        HttpContext.Session.SetString("Tenant Key", tenantKey);
    }
}
```

Figure 5.23: The back-end of the Index page.

```

<div class="text-center">
  <h1 class="display-4">Welcome</h1>
  <p>Get an overview of your spaces and points, or plot signals for monitoring.</p><br /><br />
  <p>Please enter valid credentials before continuing:</p><br />
  <label>Tenant ID:</label>
  <input type="text" id="tenantIdBox" /><br />
  <label>Tenant Key:</label>
  <input type="text" id="tenantKeyBox" /><br />
  <button onclick="createCredentials()">Set Variables</button>
</div>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
function createCredentials() {
  var tenantId = document.getElementById("tenantIdBox").value;
  var tenantKey = document.getElementById("tenantKeyBox").value;
  console.log(tenantId);
  console.log(tenantKey);
  if (tenantId != null || tenantKey != null) {
    $.ajax({
      type: 'GET',
      dataType: 'json',
      contentType: 'application/json',
      data: {
        tenantId: tenantId,
        tenantKey: tenantKey
      },
      url: '/Index?handler=CreateCredentials',
      success: function (result) {
      },
      error: function (xhr, status, error) {
        var err = eval("(" + xhr.responseText + ")");
        alert(err.Message);
      }
    });
  }
}
</script>

```

Figure 5.24: This is the javascript for declaring API credentials.

## 6 Python Library

The Python Library works on a simple premise of sending json strings to the GraphQL playground, via its URL. The json string contains several pieces of data, such as the query, data values such as strings and integers, and the header. The query is specific for each type of request. The query is itself in a json format, and is the command or function which the GraphQL playground is to execute. The data values in the json string are automatically placed in the query. The header contains the tenant id and tenant token, which determines your privileges for reading and writing data. The Python Library contains 10 functions, 9 of which are designed to be used by the user, while the last is used by all the other functions to send HTTP requests to Dimension Four.

The first function of the library is the "create\_header" function. This function takes the tenant ID and tenant key as string parameters, and returns it as a dictionary structure. This structure is used by all functions which requests data from Dimension Four. The second and third functions are "create\_space" and "create\_point" respectively. Create Space takes a user defined space name in the form of a string, along with a header dictionary and the Dimension Four playground URL, and establishes a new space. Create Point takes a point name and the space ID of the parent space, as strings, the target URL and the headers, and establishes a point.

Function four, "create\_signal", creates a new signal under a point. It takes seven variables, value, unit, signal\_type, timestamp, point\_id, and target, which are all strings, and headers, which is a dictionary. The function combines the five first variables with a GraphQL query, and sends an HTTP request to the target, along with the header.

Function five, six, and seven are all used for listing spaces, points, and signals. These are "list\_spaces", "list\_point", and "list\_signals" respectively. The List Spaces function lists all spaces available to the user, while the List Points function lists all points and their parent spaces. Both of these function takes two variables, the target string and the header dictionary. List Signals lists the latest 100 signals in a point. This function takes a point ID string, a target string and a header dictionary as variables.

Function eight, "retrieve\_latest\_signal", is used to retrieve the values from the latest signal in a point. This function takes four variables. The first is a string of the point id, the next the target URL string. Thirdly, the header dictionary. The fourth variable is a boolean for returning a dictionary structure of the signal.

Function nine is a simple save function. "save\_data" takes two variables, the data to-be-saved as a string, and a boolean to decide save format. The user may choose to save as either a .csv or .json, depending on if they want to save dictionary structures.

Function ten, "send\_post" is a posting function, used by other functions to send HTTP requests to Dimension Four. it takes three variables: target URL string, the query object, and the header dictionary. It sends the request and waits for the response. It will catch exceptions arising from bad requests.

## 6.1 Python Examples

To aid with visualizing the use cases for the Python library, several examples is provided. The first example is a continuous posting Python script. This example takes a random number between 273 and 300, and creates a signal in a specified point. The second script is a simple read function, which reads a value every 10 seconds. The third example creates a space, while the fourth illustrates a saving function. The fifth and final example in the example file is logger, which draws a pyplot of the last 100 signals i a point. All these examples may be found in the GitHub repository mentioned in 5.1.

## 7 Arduino Library

The Arduino Library is lighter in functions than the Python Library, due to the size limitation of Arduino memory. Due to unidentified problems, the library will only function on the Arduino Uno WiFi rev.2 or the MKR series of Arduinos. When using the library, "SPI.h", "WiFiNINA.h", "ArduinoJson.h", and "ArduinoHttpClient.h" must be included, in order for it to function properly. An instance of the library is then simply created. To illustrate, it can be instantiated like this: "DimensionFourApi dfour;". This creates an instance named "dfour".

The library contains 3 functions, one for setup, one for sending, and one for receiving signals. The setup function is called "initStream" and is used to pass the Serial and HTTP client to the library. This function must always be performed, in order for the library to query the Dimension Four GraphQL API. To illustrate the use of this function, in a system where the library has been instantiated as "dfour", the serial as "Serial", and the HTTP client as "client", call the function like this: "dfour.initStream(&Serial, &client);". The second function would then be called as: "dfour.ReadLatestSignal(pointId, tenantId, tenantKey, server);", where the non-obvious server is the Dimension Four playground URL: "iot.dimensionfour.io". All inputs are strings. This function queries the GraphQL API for the latest value in a point and returns a float. Following the examples of the two previous functions, the third function is called by: "dfour.PostSignal(signal, timestamp, pointId, tenantId, tenantKey, server)". Inputs shared with the read functions are the same, while signal is a float representing a measured value, and timestamp is a string of the time of creation for said signal. Examples using the library, along with the library itself can be found in the github.com repository linked to in chapter 5.1.



## 8 ASP.net Web-Application

Upon accessing the ASP.net web-application, the index page asks for Dimension Four credentials. The index is displayed in figure 8.1. How to get these credentials are explained in chapter 4.2. The tenant must have sufficient permissions to perform all actions through the web-application. The main functions of the web-app are accessed through "Admin Panel" and "Spaces" on the header-bar. Upon accessing the Admin Panel, the user is shown a drop-down menu. This menu contains the seven administrative actions of the web-application. These are: Create Space, Create Sub-Space, Delete Space, Create Point, Add Point Signal Type, Remove Point Signal Type, Delete Point.

When selecting Create Space, the user will be asked to enter a name for the space-to-be-made. When the name text-box is filled, the user clicks the "Create" button to create the space. This creates a new top-level space. Figure 8.5 shows the display for this action. To create a sub-space, select the Create Sub-Space action. To perform this action, a user is asked to name the space, and select a parent space from the recursive list on the right-hand side. The web-application notifies the user of the currently selected parent space. Once the name-box and a parent space has been selected, the user can create a sub-space with the "Create" button. Figure 8.6 depicts this display. Should the user wish to delete a space for any reason, they can use the Delete Space action. The Delete Space action requires the user to select a space from a list, like in the previous action. Select a space from the recursive list, displaying both top-level spaces and sub-spaces. The currently selected space is displayed, to avoid undesired deletion of spaces. Once a space is selected, the user may delete it with the "Delete" button. Figure 8.7 depicts the display for this action.

Should the user wish to create a point, they can use the Create Point action. Once selected, they are asked to fill in a name for the point, and select a parent space from the right-hand side list. To create the point, press the "Create" button. Figure 8.8 depicts this display. To delete a point, select the Delete Point action. Select the parent space of the point in the right-hand list, then select the point-to-be-deleted from the drop-down menu on the left. Click the "Delete" button to delete the point. Figure 8.11 depicts this action. Signal types used within a point are stored in the point metadata. To add a signal type to a point, select the Add Point Signal Type. The user then selects the point's parent space on the right-hand list, then the point in the drop-down menu, then enter the signal type in the text-box. Then use the "Add" button to add the signal type to the point metadata. Figure 8.9 depicts this display. To remove a signal type from the

# Welcome

Get an overview of your spaces and points, or plot signals for monitoring.

Please enter valid credentials before continuing:

Tenant ID:   
Tenant Key:

Figure 8.1: This is the Index page of the web-app. This is where the user presents their API credentials.

metadata, select the Remove Point Signal Type action. Select the parent space from the spaces list, then the point from the points drop-down menu. Finally, select the signal type to remove, then use the "Remove" button. Figure 8.10 depicts the user interface for this action.

To use the monitoring functions of the web-application, click on "Spaces" on the header-bar. This redirects the user to a page displaying spaces and points. When a space is clicked, the list reveals the space's sub-spaces, and the space's points are listed in the points list. This user interface is depicted by figure 8.2. To display the signals belonging to a space and its respective points, double-click the space in the list. This redirects the user to the graphical display for signals. To plot the signals, select a point, signal type, and the number of signals to display, then tick the update plot checkbox. Leaving the checkbox checked will redraw the plot every second. Any new signal uploaded to the point will then be displayed. Point and signal type can be changed during use and will be used on the next redrawing of the graph. Figure 8.3 represents this display.

# Spaces and Points

## Spaces

- Production Hall 1
  - Heater 1
  - Heater 2
- Production Hall 2
- usn-expo

## Points

- BMP 280 - 01
- BMP 280 - 02
- BMP 280 - 03

Figure 8.2: This is the Space List page of the web-app. Select a space to display points and signals.

# Heater 1

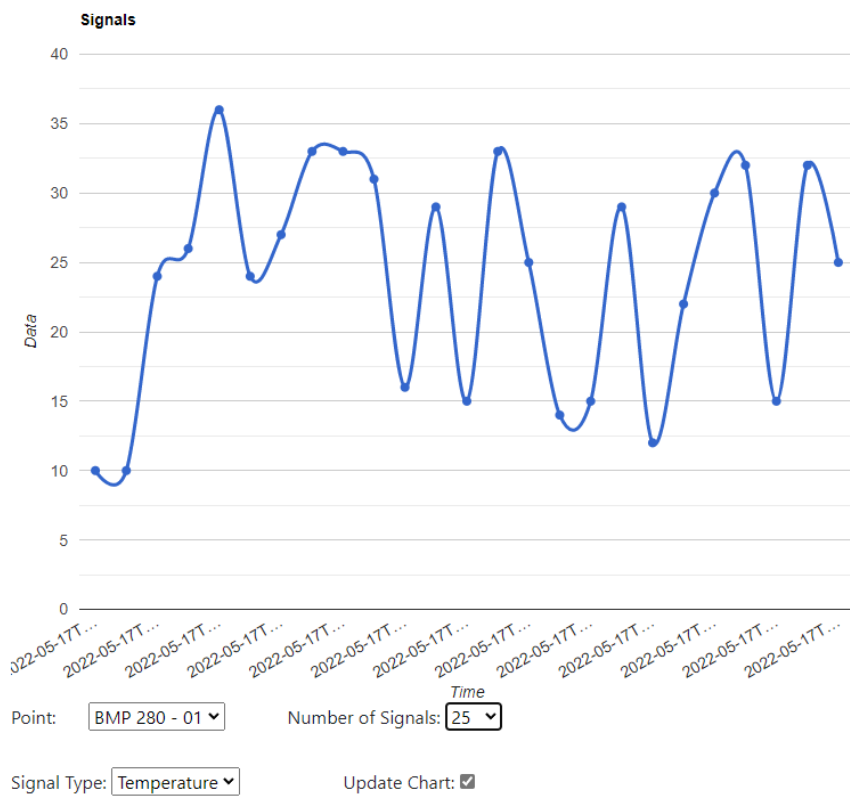


Figure 8.3: This is the Signal Display page. Here the user may choose points and signal types to display from.

# Admin Panel

Please select an action from the menu:

Figure 8.4: This is the initial Administration page. The user may select an action to perform from the list.

# Admin Panel

Please select an action from the menu:

## Create Space

Space Name:

Figure 8.5: This is the Create Space action page.

## Admin Panel

Please select an action from the menu:

Create Subspace

### Create Subspace

Current Selected Parent Space: Production Hall 2

Sub Space Name: Electrolysis Machine 1

Create

Please select a parent space from the list:

- Production Hall 1
  - Heater 1
  - Heater 2
- Production Hall 2
- usn-expo

Figure 8.6: This is the Create Sub-Space action page. The user may select a parent space from the right-hand list.

## Admin Panel

Please select an action from the menu:

Delete Space

### Delete Space

Current Selected Space For Deletion: Electrolysis Machine 1

DELETE

Please select a space from the list:

- Production Hall 1
- Production Hall 2
  - Electrolysis Machine 1
- usn-expo

Figure 8.7: This is the Delete Space action page.

## Admin Panel

Please select an action from the menu:

Create Point ▾

### Create Point

Current Selected Parent Space: Electrolysis Machine 1

Point Name:

Please select a parent space from the list:

- Production Hall 1
- Production Hall 2
  - Electrolysis Machine 1
- usn-expo

Figure 8.8: This is the Create Point action page. The user may select a parent space from the right-hand list.

## Admin Panel

Please select an action from the menu:

Add Signal Type to Point ▾

### Add Signal Type to Point

Current Selected Parent Space: Heater 1

Point:

Type:

Please select a parent space from the list:

- Production Hall 1
  - Heater 1
  - Heater 2
- Production Hall 2
- usn-expo

Figure 8.9: This is the Add Signal Type action page.

# Admin Panel

Please select an action from the menu:

Remove Signal Type from Point ▾

## Remove Signal Type from Point

Current Selected Parent Space: Heater 1

Please select a parent space from the list:

Point: BMP 280 - 03 ▾

Type: Temperature ▾

Remove

- Production Hall 1
  - Heater 1
  - Heater 2

- Production Hall 2

- usn-expo

Figure 8.10: This is the Remove Signal Type action page.

# Admin Panel

Please select an action from the menu:

Delete Point ▾

## Delete Point

Current Selected Parent Space: Electrolysis Machine 1

Please select a parent space from the list:

Point: Mass Flow - 01 ▾

DELETE

- Production Hall 1
- Production Hall 2
  - Electrolysis Machine 1
- usn-expo

Figure 8.11: This is the Delete Point action page.



## 9 Discussion

The project objective of eliminating the need for GraphQL query knowledge, has been sufficiently completed. The Python Library contains functions for the most important use cases, allowing it to be used as an administrative tool, for creating spaces and points, as well as being deployed in Python based data acquisition units or data processors. The Python Library is open-source and available at the github repository listed in chapter 5.1. In addition to this, the library has been deployed to PYPI, and thus is available through Python's package manager, PIP, as dfapi.

The Arduino Library has all planned functionality, however, it will only work with newer models which has more memory than the Arduino Uno. This result is sub-par with the planned feature of being universal, but it is functional nonetheless. The Arduino Library's two use case functions are both operational, allowing for the creation of Arduino-based IoT DAQ units. The creation of read/write interaction with the Dimension Four API has been sufficiently streamlined.

Being the biggest part of the project, the ASP.net Core web-application contains all planned features. All administrative functions, as well as displays are fully operational, and ready for limited operations. The web-application lacks action response feedback to the user, which means it does not notify the user of the outcomes of different administrative actions. This may be rolled out in a later iteration of the application.

The report documents the planning and development of APIs/Libraries for Python and Arduino, and a web-application used for monitoring and administration. The report contains, descriptions of essential themes, sources, detailed walk-throughs of development, and illustrations to aid with explanations. The report follows a natural breakdown of the development cycle. The report sufficiently documents the project and its development.

## 10 Conclusion

Data collection, processing and storage are ever-increasing with industry 4.0. Good solutions for data handling are being developed, but may not be accessible for all potential users. This project intended to make a GraphQL API and storage solution more accessible, and has done so. The project has made the solution accessible to Python developers and "Makers", who are inexperienced with GraphQL, but are in need of a data processing solution for their IoT projects.

The Python Library is designed in such away, that even inexperienced Python-programmers without extensive knowledge of dictionaries, json, or GraphQL, may take full advantage of Dimension Four's service. Be it for structuring spaces or establishing points, or log or request signals. Should the need arise, it can even save signals as a .csv or .json. While the Arduino Library lacks any administrative functions, it is functional for creating DAQ- or controller-units, without any prior knowledge of json or GraphQL. Though it is not functioning with the popular Arduino Uno, it works well with WiFi enabled devices with more memory.

The ASP.net web-application supports an array of different user actions and displays. Any user may now run this application and access Dimension Four's service, given they have valid credentials. The application gives a good overview of spaces and points, and allows users to monitor logged data. It is now ready for limited public use. Further development may include stress-testing, bug-hunting, security probing, development of a unique style sheet, and making existing code more efficient. Anyone wanting to use Dimension Four's GraphQL API, including the University of South-Eastern Norway, is now able to use Dimension Four's product.

# Bibliography

- [1] A. S. Gillis. ‘What is iot (internet of things) and how does it work?’ (2021), [Online]. Available: <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>.
- [2] ‘Mqtt version 5.0.’ (2019), [Online]. Available: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.
- [3] I. Hübschmann. ‘A complete guide to rest apis in iot.’ (2021), [Online]. Available: <https://www.nabto.com/rest-api-iot-guide/>.
- [4] A. Keranen, M. Kovatsch and K. Hartke. ‘Restful design for internet of things systems.’ (2017), [Online]. Available: <https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-05.html>.
- [5] ‘GraphQL | a query language for your api.’ (2022), [Online]. Available: <https://graphql.org/>.
- [6] ‘What is asp.net core? | .net.’ (2022), [Online]. Available: <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core>.
- [7] ‘Learn more - thingspeak iot.’ (2022), [Online]. Available: [https://thingspeak.com/pages/learn\\_more](https://thingspeak.com/pages/learn_more).
- [8] ‘Grafana features | grafana labs.’ (2022), [Online]. Available: <https://grafana.com/grafana/?plcmt=footer>.
- [9] ‘What is github? a beginner’s introduction to github.’ (2021), [Online]. Available: <https://kinsta.com/knowledgebase/what-is-github/>.
- [10] ‘What is arduino? | arduino.’ (2018), [Online]. Available: <https://www.arduino.cc/en/Guide/Introduction>.
- [11] ‘Raspberry pi foundation - about us.’ (2022), [Online]. Available: <https://www.raspberrypi.org/about/>.
- [12] ‘What is python? executive summary | python.org.’ (2022), [Online]. Available: <https://www.python.org/doc/essays/blurbs/>.
- [13] ‘About python tm | python.org.’ (2022), [Online]. Available: <https://www.python.org/about/>.
- [14] ‘A tour of c# - c# guide | microsoft docs.’ (2021), [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>.

- [15] 'Home | dimension four.' (2021), [Online]. Available: <https://dimensionfour.io/>.
- [16] 'Http (json) | d4 docs.' (2022), [Online]. Available: <https://docs.dimensionfour.io/getting-started/http>.
- [17] 'Call github graphql api using c#.' (2019), [Online]. Available: [https://www.youtube.com/watch?v=xSU1Sr\\_eY0Q](https://www.youtube.com/watch?v=xSU1Sr_eY0Q).

# **Appendix A**

## **Project Description**

Appendix A is the Project Description for this project. It includes the background for the project, as well as a set of key tasks. The Project Description lays down certain practical arrangements, entitled supervision time and the like.

# FMH606 Master's Thesis

**Title:** Development of Open Source Datalogging and Monitoring Resources for IoT Platform

**USN supervisor:** Hans-Petter Halvorsen

**External partner:** Dimension Four

## **Task background:**

USN uses different IoT platforms today in within the IT and Automation Bachelor program and the Industrial IT and Automation Master program, both for educational and research purposes.

Dimension Four (<https://dimensionfour.io>), a local company in Grenland Norway has developed a new IoT platform, which may be relevant to use by USN in the future. The IoT platform uses MQTT and GraphQL.

To use this platform at USN, both in education and research, APIs and practical examples need to be developed for different devices (PCs with DAQ devices from National Instruments, Arduino, and Raspberry Pi) and programming platforms. The main programming environments and programming languages at USN within the Bachelor/Master programs mentioned above are Visual Studio/C#, Python, LabVIEW, and MATLAB.

## **Task description:**

In this project the following activities should be performed:

- Give an overview of existing IoT solutions for Datalogging and Monitoring. Azure and ThingSpeak are platforms used by USN today, but there exist many others that may be relevant to use in the future.
- Give an overview of different standards and protocols within IoT and compare and discuss advantages and disadvantages, some examples are REST, MQTT, and GraphQL. Industrial protocols like OPC UA are also relevant to use.
- Discuss especially Data Security issues within the different platforms and protocols
- Give an overview of the Dimension Four IoT platform and their existing API. Make a detailed list of pros and cons and suggest a list of improvements and give examples how these can be implemented.
- Development of APIs and practical examples that communicates with the Dimension Four IoT platform, both datalogging (publishing data) and monitoring/visualization (retrieving data) of data for the programming platforms Visual Studio/C#, Python, LabVIEW, and MATLAB. Julia may also be an alternative, as well as ASP.NET Web Applications for presentation of data.
- The APIs and the practical examples should be tested out on different devices such as standard Windows PC with different DAQ devices from National Instruments, Arduino, and Raspberry Pi.

- Arduino: An open-source (GitHub) Arduino Library should be part of the solution
- Raspberry Pi: An open-source (GitHub) Python Library should be part of the solution
- Visual Studio/C#: An open-source (GitHub) Nuget package should be part of the solution
- LabVIEW: Open-source (GitHub) distribution via VI package Manager (VIPM)
- The APIs and the practical examples need to be investigated, tested, and explored thorough on practical applications within Home Automation and Industrial Applications. Here should also data analysis and presentation be in focus, e.g., Machine Learning.
- Compare and discuss the different IoT platforms and their features, advantages and disadvantages and suitable applications for the different IoT platforms.
- The APIs and the practical examples should be open source and should be available at GitHub.
- The different APIs and the practical examples need to be documented properly both as written documents (e.g., in GitHub), but also in form of videos available on YouTube.

**Student category:** IIA, both campus and online, but also for industry master students that want to take a project outside their own company.

**The task is suitable for online students (not present at the campus):** Yes. All work can be done online.

**Practical arrangements:**

Necessary resources and help will be provided by Dimension Four.

External partner (Dimension Four) will be responsible for providing a sensor for the project that will grade the work in collaboration with the supervisor from USN.

The resulting report should be public available.

**Supervision:**

As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

**Signatures:**

Supervisor (date and signature):

Student (write clearly in all capitalized letters):

Student (date and signature):

# **Appendix B**

## **Gantt Diagram**

The Gantt Diagram has been the main system for estimating work load and deadlines. The following appendix contains four pages of important tasks and their time-frames.



ID		Task Mode	Task Name	Duration	Start	Finish	Predecessors
1			<b>Write Report</b>	<b>88 days?</b>	<b>Mon 17.01.22</b>	<b>Wed 18.05.22</b>	
2			Format Report	7 days	Mon 17.01.22	Tue 25.01.22	
3			<b>Write IoT Chapter</b>	<b>55 days?</b>	<b>Mon 17.01.22</b>	<b>Fri 01.04.22</b>	
4			Write IoT Section	1 day	Mon 17.01.22	Mon 17.01.22	
5			Write MQTT Section	14 days	Mon 17.01.22	Thu 03.02.22	
6			Write REST Section	5 days	Mon 24.01.22	Fri 28.01.22	
7			Write GraphQL Section	14 days	Mon 17.01.22	Thu 03.02.22	
8			Write ASP.NET Core Section	6 days	Thu 03.02.22	Thu 10.02.22	
9			Write Thingspeak Section	10 days	Mon 21.03.22	Fri 01.04.22	
10			Write Grafana Section	10 days	Mon 21.03.22	Fri 01.04.22	
11			<b>Write IoT Maker Devices Section</b>	<b>11 days?</b>	<b>Fri 04.02.22</b>	<b>Fri 18.02.22</b>	
12			Write IoT Maker Devices Section	11 days	Fri 04.02.22	Fri 18.02.22	
13			Write Arduino Subsection	6 days	Fri 11.02.22	Fri 18.02.22	
14			Write Raspberry PI Subsection	6 days	Fri 11.02.22	Fri 18.02.22	
15			<b>Write Programming Language Chapter</b>	<b>14 days</b>	<b>Mon 24.01.22</b>	<b>Thu 10.02.22</b>	
16			Write Python Section	14 days	Mon 24.01.22	Thu 10.02.22	
17			Write C# Section	14 days	Mon 24.01.22	Thu 10.02.22	
18			<b>Write Dimension Four Chapter</b>	<b>11 days?</b>	<b>Fri 04.02.22</b>	<b>Fri 18.02.22</b>	
19			Write Dimension Four Chapter	11 days	Fri 04.02.22	Fri 18.02.22	
20			<b>Write GraphQL Solution Section</b>	<b>11 days</b>	<b>Fri 04.02.22</b>	<b>Fri 18.02.22</b>	
21			Write GraphQL Solution Subsection	5 days	Fri 04.02.22	Thu 10.02.22	
22			Write Dimension Four Terminology Subsection	11 days	Fri 04.02.22	Fri 18.02.22	
23			Write GraphQL Playground Section	11 days	Fri 04.02.22	Fri 18.02.22	

Project: D4-API  
Date: Tue 17.05.22

Task		Manual Summary Rollup	
Split		Manual Summary	
Milestone		Start-only	
Summary		Finish-only	
Project Summary		External Tasks	
Inactive Task		External Milestone	
Inactive Milestone		Deadline	
Inactive Summary		Progress	
Manual Task		Manual Progress	
Duration-only			

ID		Task Mode	Task Name	Duration	Start	Finish	Predecessors
24			<b>Write Development Chapter</b>	<b>82 days</b>	<b>Mon 24.01.22</b>	<b>Tue 17.05.22</b>	
25			Write Planning Section	25 days	Mon 24.01.22	Fri 25.02.22	
26			Write Python Development Section	45 days	Mon 24.01.22	Fri 25.03.22	
27			Write Arduino Development Section	15 days	Mon 21.03.22	Fri 08.04.22	
28			Write C# Development Section	27 days	Mon 11.04.22	Tue 17.05.22	
29			<b>Write Chapters about Finished Product</b>	<b>42 days</b>	<b>Mon 21.03.22</b>	<b>Tue 17.05.22</b>	
30			Write Python Library Manual	10 days	Mon 21.03.22	Fri 01.04.22	
31			Write Arduino Library Manual	5 days	Mon 11.04.22	Fri 15.04.22	
32			Write Web-Application Manual	12 days	Mon 02.05.22	Tue 17.05.22	
33			Write Discussion Chapter	12 days	Mon 02.05.22	Tue 17.05.22	
34			Write Conclusion Chapter	12 days	Mon 02.05.22	Tue 17.05.22	
35			Write Preface	12 days	Mon 02.05.22	Tue 17.05.22	
36			Write Summary	12 days	Mon 02.05.22	Tue 17.05.22	
37			General Report Structuring	17 days	Mon 25.04.22	Tue 17.05.22	
38			<b>Programming</b>	<b>82 days</b>	<b>Mon 17.01.22</b>	<b>Tue 10.05.22</b>	
39			Create Github Repository	0 days	Mon 17.01.22	Mon 17.01.22	
40			Register D4 User	0 days	Mon 17.01.22	Mon 17.01.22	
41			Experiment With Python and D4	10 days	Mon 17.01.22	Fri 28.01.22	
42			Draw System Sketch	1 day	Mon 24.01.22	Mon 24.01.22	
43			<b>System Planning Python</b>	<b>3 days</b>	<b>Wed 19.01.22</b>	<b>Fri 21.01.22</b>	
44			Define Requirements	3 days	Wed 19.01.22	Fri 21.01.22	
45			Draw UML Diagrams	3 days	Wed 19.01.22	Fri 21.01.22	
46			<b>Program Python</b>	<b>45 days</b>	<b>Mon 17.01.22</b>	<b>Fri 18.03.22</b>	
47			Develop Create Space Function	5 days	Mon 17.01.22	Fri 21.01.22	














Project: D4-API  
Date: Tue 17.05.22

Task		Manual Summary Rollup	
Split		Manual Summary	
Milestone		Start-only	
Summary		Finish-only	
Project Summary		External Tasks	
Inactive Task		External Milestone	
Inactive Milestone		Deadline	
Inactive Summary		Progress	
Manual Task		Manual Progress	
Duration-only			

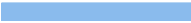

















ID		Task Mode	Task Name	Duration	Start	Finish	Predecessors
48			Develop Post Function	7 days	Thu 20.01.22	Fri 28.01.22	
49			Develop Create Point Function	6 days	Fri 21.01.22	Fri 28.01.22	
50			Develop Create Signal Function	8 days	Wed 26.01.22	Fri 04.02.22	
51			Develop List Space Function	6 days	Fri 04.02.22	Fri 11.02.22	
52			Develop List Point Function	6 days	Fri 04.02.22	Fri 11.02.22	
53			Develop List Signals Function	6 days	Fri 04.02.22	Fri 11.02.22	
54			Develop Retrieve Latest Signal Function	9 days	Fri 11.02.22	Wed 23.02.22	
55			Develop Save Data Function	6 days	Fri 25.02.22	Fri 04.03.22	
56			Develop Create Headers Function	2 days	Fri 04.03.22	Mon 07.03.22	
57			Develop Python Examples	11 days	Fri 04.03.22	Fri 18.03.22	
58			<b>Program Arduino</b>	<b>15 days</b>	<b>Mon 21.03.22</b>	<b>Fri 08.04.22</b>	
59			Pass Serial and Client to Library	5 days	Mon 21.03.22	Fri 25.03.22	
60			Develop Post Function	6 days	Fri 25.03.22	Fri 01.04.22	
61			Develop Retrieve Function	6 days	Fri 25.03.22	Fri 01.04.22	
62			Develop Examples	6 days	Fri 01.04.22	Fri 08.04.22	
63			<b>Program ASP.net</b>	<b>22 days</b>	<b>Mon 11.04.22</b>	<b>Tue 10.05.22</b>	
64			Develop HTTP Client	5 days	Mon 11.04.22	Fri 15.04.22	
65			Develop Space List Page	10 days	Mon 11.04.22	Fri 22.04.22	
66			Develop Signal Display	11 days	Fri 15.04.22	Fri 29.04.22	
67			<b>Develop Administration Page</b>	<b>6 days</b>	<b>Fri 29.04.22</b>	<b>Fri 06.05.22</b>	
68			Develop Create Space Function	3 days	Fri 29.04.22	Tue 03.05.22	
69			Develop Create Sub-Space Function	3 days	Fri 29.04.22	Tue 03.05.22	

Project: D4-API  
Date: Tue 17.05.22

Task		Manual Summary Rollup	
Split		Manual Summary	
Milestone		Start-only	
Summary		Finish-only	
Project Summary		External Tasks	
Inactive Task		External Milestone	
Inactive Milestone		Deadline	
Inactive Summary		Progress	
Manual Task		Manual Progress	
Duration-only			

ID		Task Mode	Task Name	Duration	Start	Finish	Predecessors
70			Develop Delete Space Function	2 days	Mon 02.05.22	Tue 03.05.22	
71			Develop Create Point Function	3 days	Wed 04.05.22	Fri 06.05.22	
72			Develop Add Signal Type Function	3 days	Wed 04.05.22	Fri 06.05.22	
73			Develop Remove Signal Type Function	3 days	Wed 04.05.22	Fri 06.05.22	
74			Develop Delete Point Function	3 days	Wed 04.05.22	Fri 06.05.22	
75			Add Session Stored Credentials	3 days	Fri 06.05.22	Tue 10.05.22	



Project: D4-API Date: Tue 17.05.22	Task		Manual Summary Rollup	
	Split		Manual Summary	
	Milestone		Start-only	
	Summary		Finish-only	
	Project Summary		External Tasks	
	Inactive Task		External Milestone	
	Inactive Milestone		Deadline	
	Inactive Summary		Progress	
	Manual Task		Manual Progress	
	Duration-only	