## **Software Design**

In the following section, an overview of the software modules that will be used is provided.

### **Purpose**

Software both on the on-board part of the experiment and the ground station is going to be used for different reasons.

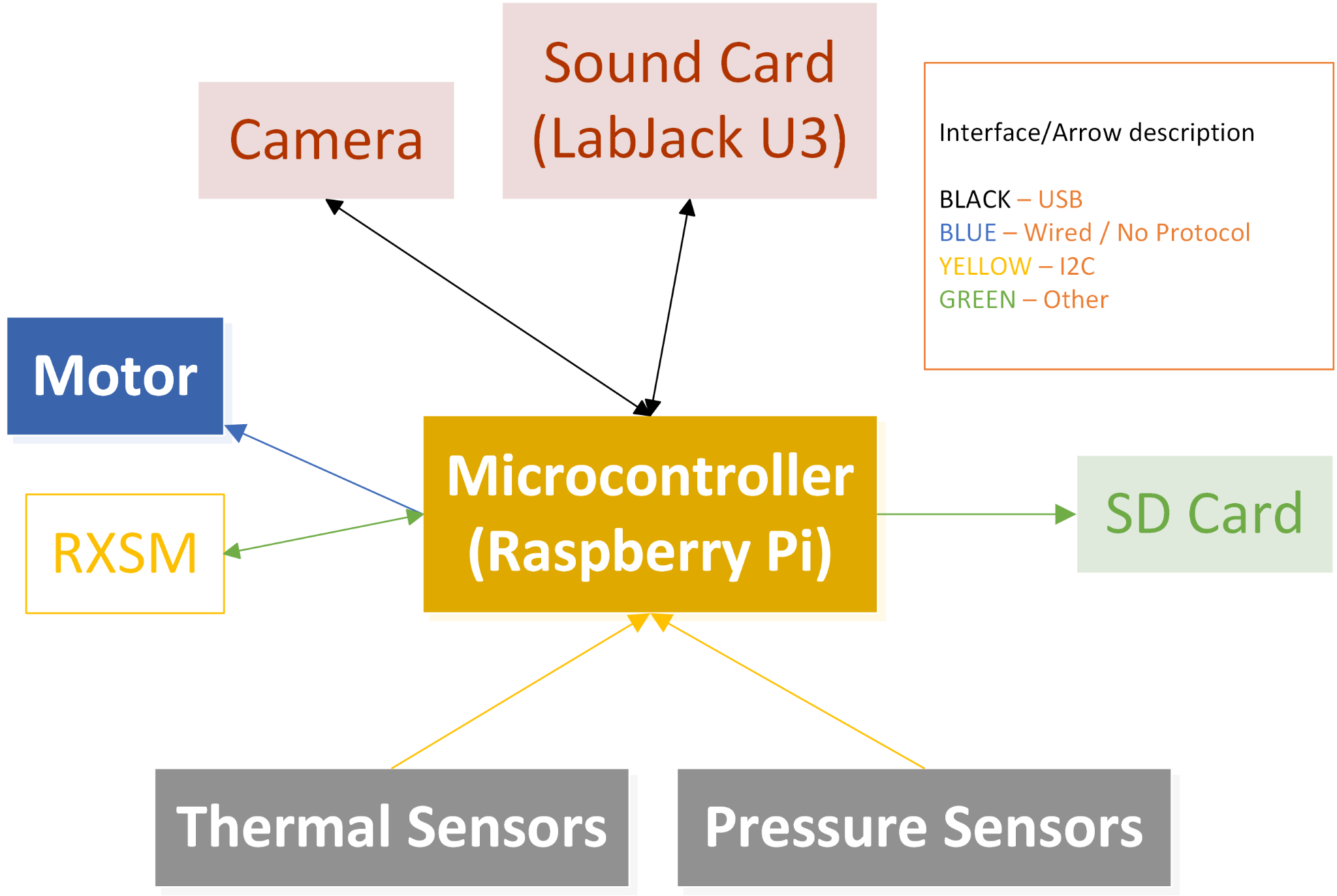
* The **on-board system** is going to :
  + Monitor the heaters’ temperature (when to turn on and off).
  + Be responsible for the movement of the emulsification motor.
  + Turn on and off the I-VED technique.
  + Store the data from the I-VED technique in a logfile in the SD card.
  + Store the data from the pressure and thermal sensors in a logfile in the SD card.
  + Create a collective logfile where all of the data is going to be stored in the SD card.
  + Control the Camera and store the video data in the camera’s SD card.
  + Send a sample of the data of the sensors and the system events to the ground station.
  + Have a testing mode, which will be telecommanded by the Uplink, to be able to turn on or off each of the above separately before lift-off for testing and verification purposes.
* The **ground station** **system** is going to :
  + Receive the sample data from the on-board system using the rocket’s Downlink.
  + Include a GUI for the visualization of the data during the flight and the telecommand of the rocket while Uplink is available.
  + Log/Backup the sample of data that will be sent from the rocket through Downlink.
  + Send commands to the rocket to turn on and off certain components before Lift off through Uplink.

### **Design**

#### **Process Overview**

As seen in the diagram below, the microcontroller will receive data from the three digital sensors (they all work as both pressure and thermal ones) which are going to be directly connected to it ~~and according to them it will adjust the Heaters’ temperature while also~~ storing their data in the SD card. Meanwhile, it will also be able to control the motion of the DC Motor using the Motor Controller and communicate with the Camera and Sound Card (for the I-VED technique) in order to turn them on and store their data in the SD Card. The Downlink will be used in order to send a sample of data to the ground station.

All of the components will be connected to the Raspberry communicating with it using the GPIO pins of it except for the camera that will use a USB cable.

****

**Figure 4.47: Block diagram of the experiment components.**

There will be 2 different modes in which the experiment is going to operate:

* Experiment mode: Used before lift off / all of the components can be switched on and off separately / telecommanded by Uplink / serves testing and verification purposes / versatile to its use.
* Flight Mode: Used only after lift off / the components are switched on and off according to the preprogrammed rocket signal commands / has strict restriction to its use.

#### **General and safety related concepts**

As shown in the 3.5.2 Risk assessment paragraph, the probable errors or safety issues that Software subteam has to be concerned over are :

* Microcontroller failure
* Connection Loss between the rocket and the ground station
* Sensor Software Failure
* Protocol Errors

##### *4.8.2.2.1 Microcontroller Failure or Breakdown*

This error’s probability is extremely low, but if something like that happens the consequences are critical for the experiment. Without the microcontroller the data from the sensors, the control of the motor and the sound card are non-existent, and depending on the timing of the occurrence, it could be completely fatal for the experiment since the camera could never be signaled to start recording.

The ways to mitigate the probability of this happening are :

* Buying the product from a well-known and trusted vendor, in order to avoid badly made soldering or PCB printing, mistreatment of the product, and in order to have possible customer support in case something occurs.
* Testing it under extreme conditions and avoid mistreating or endangering it, so that we can be confident enough that the microcontroller won’t misbehave during the flight.

##### *4.8.2.2.2 Connection Loss*

This includes both total and temporary connection loss between the rocket and the ground station.

In case of total connection loss, no data will be sent to the ground station and thus the team will have no clue on the status of the experiment. There, also, will be no way to backup the data.

On the other hand, a case of a temporary connection loss equals a partial loss of data, which in this case will not be so severe, because the only reason the data is sent to the ground is to monitor the experiment and have a backup option for a sample of the data.

The probability of such a thing happening is quite high but manageable up to a point with Go-Back protocols and algorithms.

The actions that can be taken in order to avoid these problems are:

* Testing the code extensively in order to make sure that the problem is not created by itself.
* Testing different kinds of situations and try to come up with error handling scenarios for these.
* Testing with the rocket system mounted to see if some of the problems are created because of other connectivity issues besides the code itself or whether some problems are created because of different protocols.

##### *4.8.2.2.3 Sensor Software Failure*

Sensor software failure can be considered highly unlikely but its consequences can range between highly dangerous to the experiment to just a partial distortion of the final data. Therefore it creates an issue that should be addressed.

There are very few ways to mitigate the possibilities of such a thing happening, mainly because software vendors are taking care over these issues and publish new updates regularly, but these are some of the ways :

* Testing the code extensively for possible unknown mistakes or exceptions happening,
* Using well known libraries in order to avoid instabilities in the code,
* Update libraries frequently and test the code on the newest updates.

##### *4.8.2.2.4 Protocol Errors*

Protocol Errors are highly unlikely to happen, mainly because the DLPs implement error detection but the consequences of this happening can be catastrophic to the results of our experiment by distorting the final data. Therefore, it creates an issue that should be addressed.

The only way to mitigate the problem is to have error correction in the DLPs that we already have implemented as shown in the paragraph below.

#### **Interfaces**

In terms of the interfaces:

* All of the experiments’ components will be wired-soldered together using the I2C protocol except for the camera and the Sound Card whose connections will be USB.
* The experiment will connect to the Downlink and the Uplink of the rocket using an RS-422 port. The data bandwidth usage is described below in the following tables.

Another way to visualize the interfaces is through Figure 4.24.

The description of the Downlink and Uplink data protocols is following below:

The communication will follow the point-to-point system between the experiment and the ground station.

This serial communication between them, the baudrate of the communication will be configured to 9600.

As for the Downlink, the maximum bandwidth will only be ***648 bits/s***. As calculated below:

The communication protocol developed will produce frames (as seen below) composed from **27 bytes**. More precisely,a start word of **2 bytes**, a segment ID of **2 bytes**, a payload of **19 bytes**, a CRC Checksum of **2 bytes** and an end word of **2 bytes.**

****

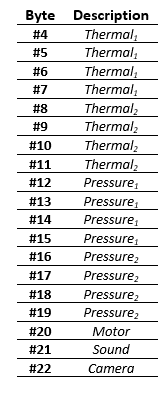
**Table ???: Communication Frame**

The Start and End bytes will be set to static values for detecting each package received.

The Segment ID field of each frame will have a unique value so it can be identified.

The CRC bytes will be calculated using the *CRC-16/MODBUS* algorithm.

As mentioned, the payload will be **19 bytes** long and will look like this:

**~~~~**

**Table ???: Payload structure**

The values of the Thermal and Pressure fields are each 4 bytes long, because a floating point value will be returned that follows the *IEEE 754* format.

The Motor, Sound and Camera fields will contain some standard values to make sure that each component is working.

**~~Table 4.13: Data to be sent through the Downlink.~~**

| **~~Data to be sent through the Downlink~~** | | |
| --- | --- | --- |
| ~~Data~~ | ~~bits~~ | ~~Frequency (Hz)~~ |
| ~~Thermal Sensor 1~~ | ~~16~~ | ~~3~~ |
| ~~Thermal Sensor 2~~ | ~~16~~ | ~~3~~ |
| ~~Thermal Sensor 3~~ | ~~16~~ | ~~3~~ |
| ~~Pressure Sensor 1~~ | ~~16~~ | ~~3~~ |
| ~~Pressure Sensor 2~~ | ~~16~~ | ~~3~~ |
| ~~Pressure Sensor 3~~ | ~~16~~ | ~~3~~ |
| ~~Motor Status~~ | ~~8~~ | ~~1~~ |
| ~~Heater Status~~ | ~~1~~ | ~~1~~ |
| ~~Sound Card Status~~ | ~~1~~ | ~~1~~ |
| ~~Camera Status~~ | ~~8~~ | ~~1~~ |
| ~~Error~~ | ~~8~~ | ~~1~~ |
| ~~Sum :~~ | ~~122~~ | ~~Max : 3~~ |

**~~Table 4.14: Example of data link protocol structure.~~**

| ~~Sync1~~ | ~~Sync2~~ | ~~MsgID~~ | ~~MsgCNT~~ | ~~Data~~ | ~~CSM~~ | ~~CSM~~ | ~~CRC~~ | ~~CRC~~ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~8~~ | ~~8~~ | ~~8~~ | ~~8~~ | ~~122~~ | ~~8~~ | ~~8~~ | ~~8~~ | ~~8~~ |

**~~Table 4.15: Calculation result.~~**

| ~~Baud Rate (bits/s):~~ | ~~558~~ |
| --- | --- |

~~Which derived using the the following formula:~~

~~In kB/s that is roughly 0.07kB/s as a maximum Bandwidth of the Downlink.~~

As for the Uplink, a different DLP has to be used:

**Table 4.16: Data to be sent through the Uplink.**

| **Data to be sent through the Uplink** | | |
| --- | --- | --- |
| Data | bits | Frequency (Hz) |
| Turn on/off Different Components | 8 | 3 |
| Sum : | 8 | Max : 3 |

**Table 4.17: Example of the messages sent through the Uplink and their meaning.**

| **Meaning of Each Byte Sent** | |
| --- | --- |
| Turn On Thermal Sensors | x01 |
| Turn Off Thermal Sensors | x02 |
| Turn On Pressure Sensors | x03 |
| Turn Off Pressure Sensors | x04 |
| Turn On Sound Card | x05 |
| Turn Off Sound Card | x06 |
| Turn On Camera | x07 |
| Turn Off Camera | x08 |
| Motor Full Speed | x09 |
| Motor Stop | x0A |
| Turn On Heaters | x0B |
| Turn Off Heaters | x0C |

**Table 4.18: Example of the data link protocol structure.**

| Sync1 | Sync2 | MsgID | MsgCNT | Data | CSM | CSM | CRC | CRC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

**Table 4.19: Calculation result.**

| Baud Rate (bits/s): | 216 |
| --- | --- |

Which is derived using the following formula:

br =( + Protocolbits)\*(Frequency)

= (8 +64 )\*3 = 216 bits/s

In kB/s that is roughly 0.026kB/s as a maximum Bandwidth of the Uplink.

Adding the two of them together results in an average of 888 bits/s, during the testing phase when the Uplink will be used.

#### **Data acquisition and storage**

The data collected are going to be split into 3 different categories:

* Video data, which are going to be buffered in the Camera at first and then stored in a file inside the SD card,
  + The video file size is approximately going to reach 4-5 GBs, but further tests will be done in order to determine the final size because it depends on many factors, mainly on the technical details of the camera.
* Sound Card data, of the I-VED technique, which are going to be stored inside the the SD card of the Raspberry Pi,
  + The file is going to be in a `.wav` format and the size of it ranges between 1-12 GBs because the frequency in which the Sound Card will be functioning hasn't been finetuned yet. It definitely can be stored inside the SD card of the Raspberry Pi, though.
* Sensor and  Event data, which are going to be stored in the same folder as the Sound Card data, will be stored in multiple different logfiles with a `.csv` format and a collective CSV file which will make sure the data are backed up.
  + In the image below we can deduct that the total storage size used up by the logfiles is 117kB.

Εικόνα που περιέχει πίνακας

Περιγραφή που δημιουργήθηκε αυτόματα

**Figure 4.48: A simulation of the final data logged in the Raspberry Pi storage.**

Apart from this data, a backup logfile will be kept on the ground station from the sample of the data that will be sent from the experiment. Since the data sent is the CSV with the collective sensor and event data, the size of it is going to be about 57kB.

~~Additionally, there will be some data that will be created during the tests. This data doesn’t need to be stored because its size is going to be very small.~~

#### **Process Flow**

**Timeline

Description automatically generated with low confidence**

**Figure 4.49: Experiment timeline.**

Diagram

Description automatically generated with medium confidence Diagram

Description automatically generated

**Figure 4.50: ~~From left to right -~~ Motor Control Process Flow ~~- Heater Controller Process Flow.~~**

Diagram

Description automatically generatedDiagram

Description automatically generated

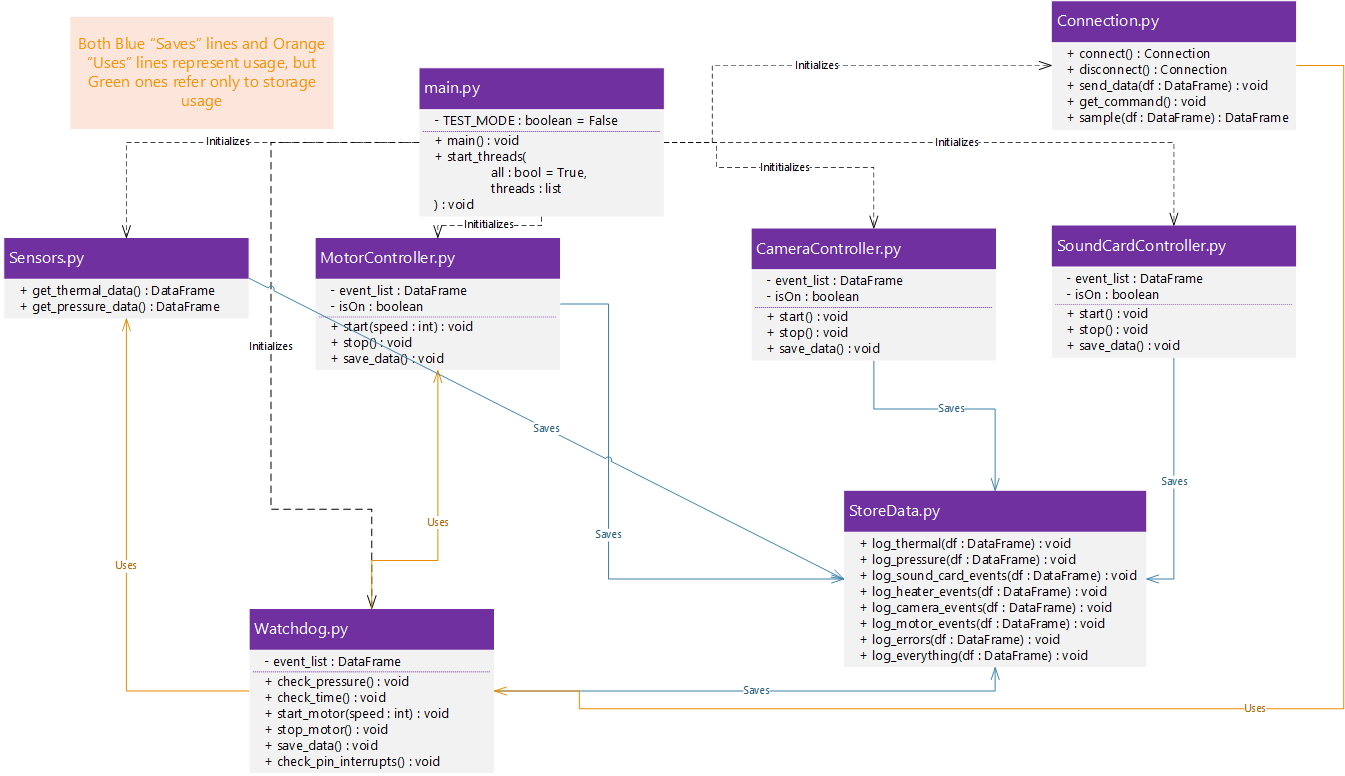
**Figure 4.51: From left to right - Camera Controller Process Flow - Sound Card Controller Process Flow.**

#### **Modularization and pseudo code**

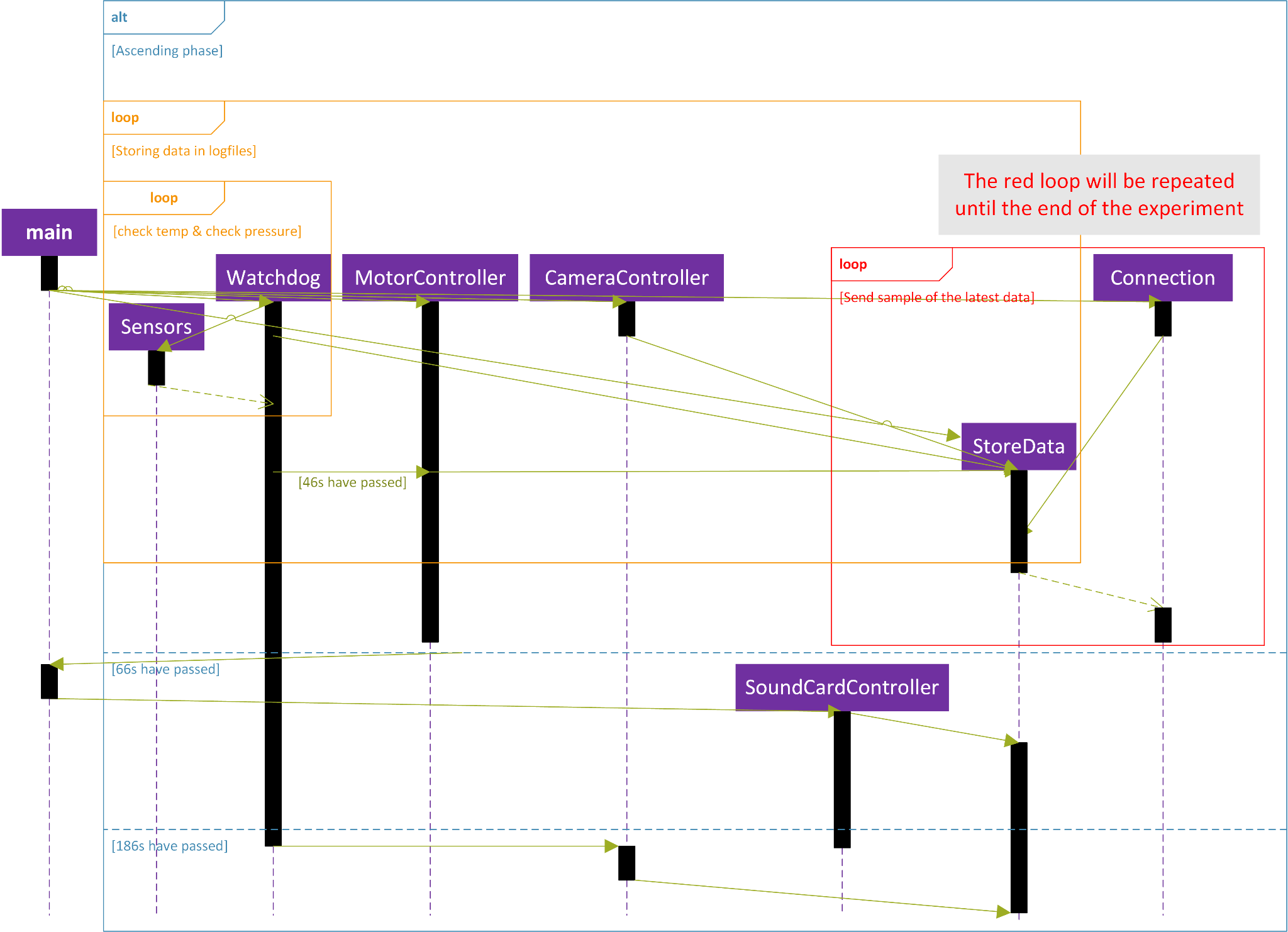
The code of the experiment used on the rocket is going to be separated into several different modules which can be seen in the diagram below.

It will have 2 separate modes. A test and an experiment mode. These 2 modes will be represented in the TEST\_MODE variable in the `main.py` module. The `main.py` will initialize all the component Controllers, the Connection and the `Watchdog.py` in different threads. These threads will be able to draw and message data, when needed, to each other and they will all use the `StoreData.py` module in order to save their data in the SD card. `Connection.py` module will get a sample of this data from `StoreData.py` in order to send it to the ground station. ‘Watchdog.py` module is going to be really important in this setup as it is the one that will judge whether the temperature or the pressure in the rocket’s module is too high and stop the motor accordingly, it will also synchronize a timer with the rocket’s signals using the Raspberry’s IO pins and define whether it needs to switch to Test mode.

As a backup solution, on the ground the sample of the data, sent from the rocket, will be saved by writing all the data on a logfile.

****

**Figure 4.52: UML of the relations between different modules.**



**Figure 4.53: Sequence UML that explains the usage timeline of each module during Experiment Mode.**

The UML Sequence diagram above describes the usage of each class mentioned before, according to the Timeline represented in the 4.8.2.5paragraph

### **Implementation**

Regarding the implementation, the programming language Python will be used, since it’s easy to write, very rich and it is compatible with all the tasks the Software subteam is called to work on. It is one of the main languages that are used for Data Analysis, the main language that Raspberry Pi uses and it can create very rich GUIs when used correctly. ~~Other lower-level languages like C or Java might also be used in order to implement the Data Link protocols as seen in paragraph 4.8.2.3~~~~but that remains to be defined after the STW.~~

To aid the development, Mu, Spyder, Visual Studio Code (setted up for Python) and PyCharm IDEs will be used each for different stages of the development as they offer different tools for debugging or projecting information.

The Libraries and Frameworks that are definitely going to be used are Anaconda, Django, pandas, matplotlib, scipy, pyserial and numpy. Some libraries, like the adafruit\_blinka, gpiozero, RPi.GPIO, spidev and smbus2, will have to be studied before use in order to choose which one is better or safer to use.

In terms of Operating Systems, Raspbian is going to be used for the Raspberry, so the knowledge of Linux kernels and Debian Linux is vital, and everything is going to be developed in Microsoft Windows.

## **Ground Support Equipment**

The ground station equipment, is split in two categories:

a. Hardware (computers, cables, etc.)

b. Software (IDE, browser, etc.)

After a brief discussion with the experts of the programme, we settled on using the RS232 Port since it will be the only way to test our software,  before the campaign, using the RXSM Simulator. Additionally, a computer (DROPSTAR Team) will be used for receiving, decoding and evaluating the data. The computer will be brought by the team members. As mentioned, the reception will happen through RS232, so a cable needs to be provided as well. Last but not least, a power strip is needed (preferably with 4-5 sockets) and to be supplied. (ESRANGE Space Centre)

The Ground Station software will help us visualize the data received, during both the testing and the experiment phase, this will be implemented with a browser-based GUI (developed with the Django Python framework). The data will be shown through graphs and charts to make monitoring easy for the team members. The received data will be decoded using an interface developed in Python. One page will be used for the testing mode and another one for the experiment mode. (as seen in Figure 4.37 below).

The programming language that will be mainly used is Python and the IDEs of choice are PyCharm and Visual Studio Code (setted up for Python). If there is a need for a lower-level language, C/C++ will be used accordingly.

~~Chart

Description automatically generated~~

**~~Figure 4.54: A first mockup draft of the Ground Station GUI.~~**

**Ground Station GUI**

~~The GUI is currently in a designation phase, but you can observe the first mockups in the following Figures.~~

~~Chart, line chart

Description automatically generated~~

**~~Figure 4.55: The homescreen of the GUI is by default on Experiment Mode.~~**

~~The first landing page of our GUI is the Experiment mode where the status of each experiment component can be seen on the top right, plots demonstrating the current pressure and temperature in the rocket are on the top left and, at the bottom, there’s a brief description about experiment mode. There is also a button which leads to the Test mode (described below).~~

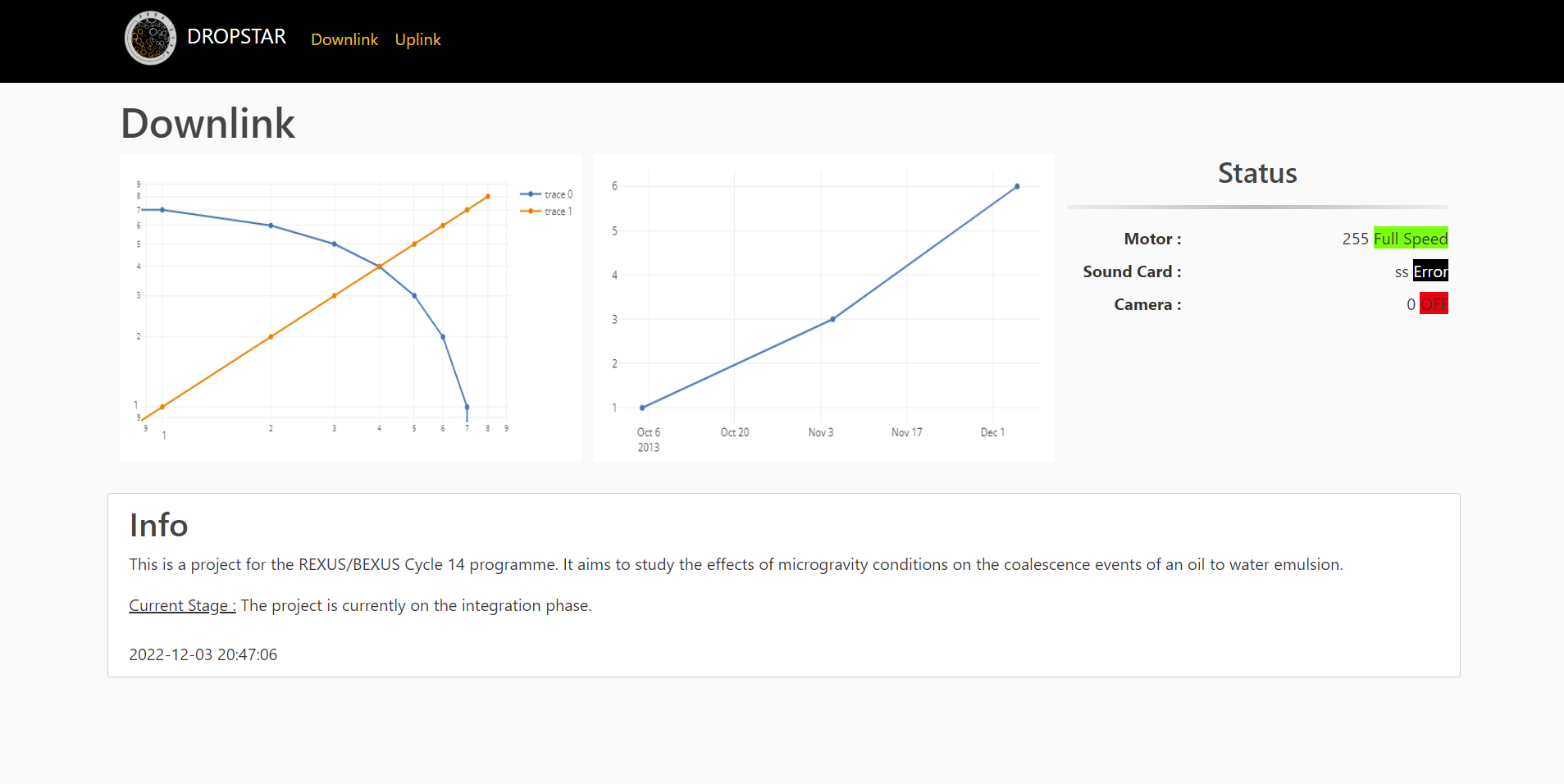
~~Diagram, schematic

Description automatically generated~~

**~~Figure 4.54: The Test mode page for the preliminary tests during the launch phase.~~**

~~The Test Mode consists of switches mainly, that turn on and off the different experiment components separately. Whenever a component is switched on or off, there’s a brief wait for confirmation from the downlinked data. As for the Motor switch, there are 2 modes, STOP and FULL SPEED, where, according to each mode and the downlinked data, the values beneath the switch change (and vary between 0-255, where 0 is fully stopped and 255 is on full speed). Beneath the switch there is also a text field and a button where the user can also insert a certain speed value manually and not just full speed. At the top right there is a button leading to the Experiment Mode page which also causes every component to turn off when clicked.~~

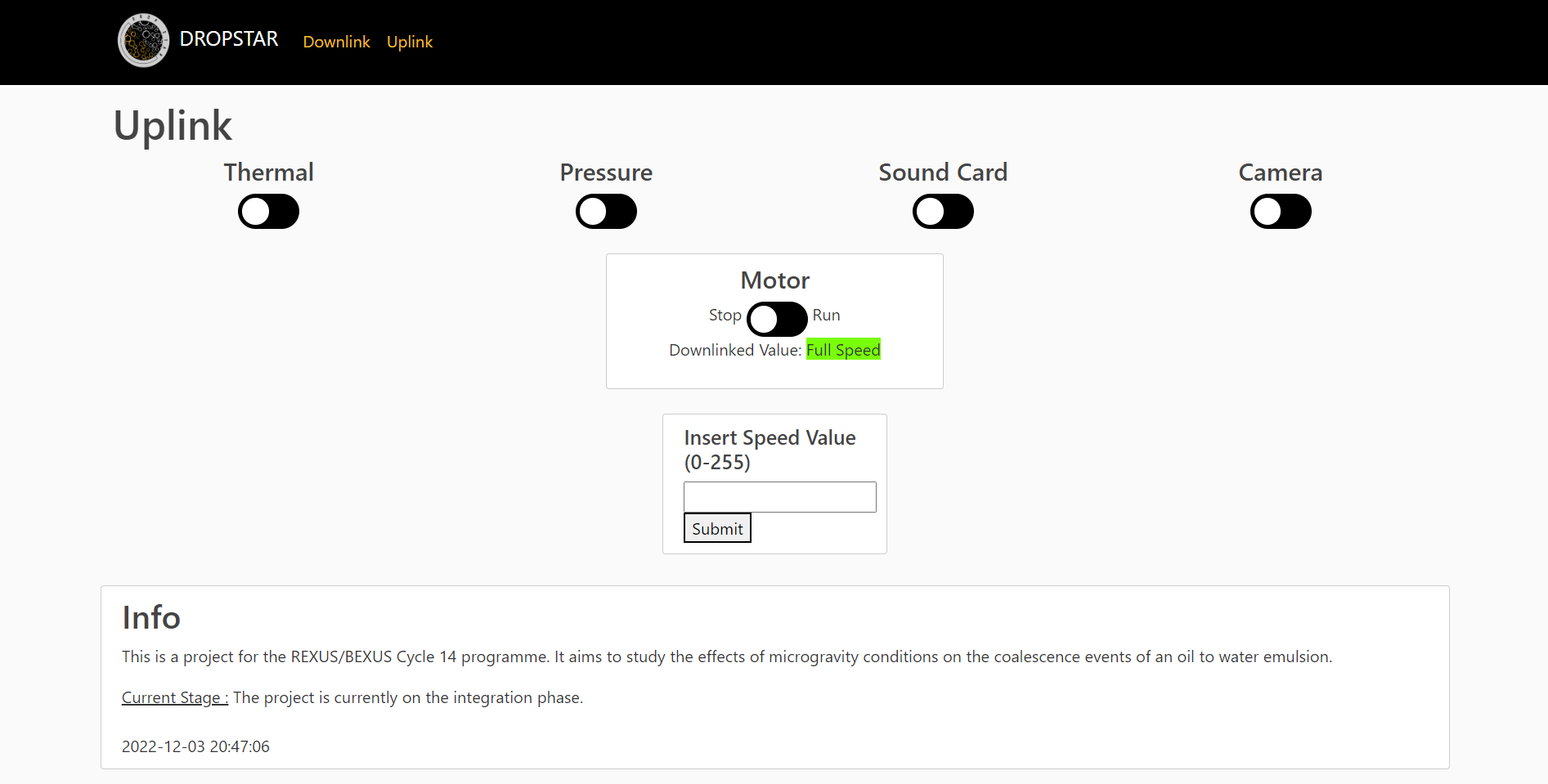
Currently more than 60% of the GUI is designed and functional. You can see in the following Figures its appearance.

****

**Figure 4.55: The homescreen of the GUI is by default on Downlink.**

The first landing page of our GUI is the downlink. There the status of each experiment component can be seen on the top right, the downlinked value is displayed and a user-friendly coloured tag demonstrates the info accordingly. Plots demonstrating the current pressure and temperature in the rocket are on the top left and, at the bottom, there’s a brief description of the experiment’s current stage. There is also a button on the navigation headbar which leads to the uplink page (described below).

The plots and the component’s status are not currently continuously updated according to the data, but we are in the process of coding the corresponding part in JavaScript using the plotly library and AJAX requests.

****

**Figure 4.54: The Uplink page.**

The uplink consists of switches mainly, that turn on and off the different experiment components separately. For the Motor switch, there are 2 modes, STOP and RUN. According to each mode and the downlinked data, the motor status below of the switch changes. Since it varies between 0-255, for 0 it shows “Stopped”, for 255 “Full Speed”, and for the intermediary values “Speed Up”. Beneath the switch, there is also a text field and a button where the user can also insert a certain speed value manually and not just full speed. At the page’s navigation bar there is a button leading to the downlink page which will also cause every component to turn off when clicked.

The switches change colour when they’re switched on or off and an extra label will be added to them which will be used to verify whether the Uplinked value of them has been received by the experiment on the rocket and thus has been downlinked back down to the Ground Station.

Same as the Downlink page, the label below the motor status is not refreshing in real-time but this functionality is currently being developed using Javascript and AJAX requests.

The GUI was coded using Django which is a high-level python web framework. Django was chosen since it provides a powerful templating library in order to handle rendering forms as dynamic HTML. Another advantage of Django is that, since it uses Python, it can also implement the reception of data from the rocket much more easily without an intermediary or a second language.

## **Timeline for Countdown and Flight**

The timeline of events for DROPSTAR, during the countdown and the flight, is as:

**Table 6.5: DROPSTAR's timeline for countdown and flight.**

| **Time [s]** | **Signal** | **Function** |
| --- | --- | --- |
| T- 600s | POWER ON | Experiment is switched on and waits for lift-off ~~for final connectivity testing~~ |
| ~~T-300s~~ | ~~SODS~~ | ~~Testing Mode of the experiment is turned on to test each component pre-flight~~ |
| T=0s | LO | After lift-off is used for time sync and the Camera turns on |
| T+46s |  | Motor gets power and starts moving full speed |
| T+66s | SOE | Motor stops and I-VED Technique starts taking measurements |
| T+186s | SOE OFF | I-VED Technique turns off |
| T+600s | POWER OFF | Camera turns off storing its data, Experiment turns off |