Design and Implementation of a software architecture for an extensible network of Satellite Ground Stations

Development of the infrastructure and software components to enable real-time remote control of Satellite Ground Stations and a centralized Operation Center from scratch, including both reusable modular software.

FACULTAT D'INFORMÀTICA DE BARCELONA - Fall 2020

Specialization in SOFTWARE ENGINEERING

Aina Garcia Espriu

Director: Adriano Camps Carmona
Co-directors: Joan Adrià Ruiz-de-Azúa Ortega,
Joan Francesc Muñoz Martin
and Adrián Pérez Portero
Ponent: Juan Carlos Cruellas Ibarz

Abstract

With the increasing number of satellite constellations, the number of Ground Stations needed to control them has risen dramatically. This complicates their management and operation, resulting in an increase of the workload for operators, which in turn become more prone to commit errors. For this reason, automation is becoming more important when controlling the Ground Segment. The Nano-Satellite and Payload Laboratory (NanoSat Lab) has also followed this trend, extending its current Ground Station network. This work presents the software architecture needed to achieve a modular and scalable system that copes with this current demanding increase. The design of this architecture is detailed in the following sections, highlighting the different experiences during the deployment of the software. Additionally, results of the system performance are presented as well. Finally, the work is concluded with future tasks and recommendations.

Resum

Amb l'increment del número de constel·lacions de satèl·lits, cada cop es necessiten més Ground Stations per tal de controlar-les. Això dificulta més i més la seva gestió i operació, implicant també un augment en la càrrega de treball pels operadors, fent així més propensos els errors. És per això que cada vegada és més important l'automatització a l'hora de controlar el Ground Segment. El Nano-Satellite and Payload Laboratory (NanoSat Lab) també ha apostat per aquesta tecnología alhora que estén la seva xarxa de Ground Stations. Aquest treball presenta l'arquitectura de software necessària per aconseguir un sistema modular i escalable que fa front a l'actual increment de demanda. El disseny d'aquesta arquitectura està detallat en les següents seccions, fent incís també en diferents experiències viscudes durant el desplegament del software. Així mateix, es presenten els resultats del funcionament del sistema. Finalment, el treball acaba amb una avaluació de feina futura i recomanacions.

Resumen

Con el incremento del número de constelaciones de satélites, cada vez se necesitan más Ground Stations para controlarlas. Esto dificulta más y más su gestión y operación, implicando también un aumento en la carga de trabajo por los operadores, haciendo así más propensos los errores. Es por eso que cada vez es más importante la automatización en la hora de controlar el Ground Segment. El Nano-Satellite and Payload Laboratory (NanoSat Lab) también ha apostado por esta tecnología a la vez que extiende su red de Ground Stations. Este trabajo presenta la arquitectura de software necesaria para conseguir un sistema modular y escalable que hace frente al actual incremento de demanda. El diseño de esta arquitectura está detallado en las siguientes secciones, haciendo inciso también en diferentes experiencias vividas durante el despliegue del software. También se presentan los resultados del funcionamiento del sistema. Finalmente, el trabajo acaba con una evaluación de trabajo futuro y recomendaciones.

You cannot alter your fate. However, you can rise to meet it.

- Princess Mononoke (1997)

Acknowledgements

Many people has selflessly offered their help during the year that I have been writing this thesis. I am thankful to everyone that has stayed by my side during this journey.

My thanks and appreciation to my family and all my friends, who supported me along the way and helped me disconnect when necessary. To everyone in the NanoSat Lab, for providing an environment where it is so easy to learn and feel comfortable.

I would also like to thank Professor Adriano Camps for giving me the opportunity to conduct my thesis in the NanoSat Lab and for having an open mind with the ideas that we propose. My thanks also to Juan Carlos Cruellas for acting as an intermediary with the university.

Special thanks to Joan Adrià, JuanFran and Adrián, for advising, guiding and helping me throughout the process. Your knowledge and willingness to help have been of great importance in this work.

And last, I would like to thank Adrián again for sharing his optimism towards every difficult moment I faced and helping me self-improve day by day.

List of Figures

2.1 CubeSat missions segments . . . 14
2.2 Basic communication between Earth and Satellites . . . 16
2.3 Communication with Satellites Requirements . . . 17
2.4 Elevation and Azimuth angles for a signal emitter [1] . . . 17
2.5 Commands Conversion . . . 19
2.6 Montsec GS . . . 20
2.7 Stakeholders diagram . . . 21

4.1 Workflow dependencies . . . 34
4.2 GANTT diagram text part . . . 36
4.3 GANTT diagram . . . 37
4.4 Break down of expenses . . . 41
4.5 Final GANTT tasks list . . . 45
4.6 New GANTT diagram . . . 46
4.7 Updated budget . . . 47

5.1 System design overview . . . 50

6.1 Ground Station (GS) software overview . . . 52
6.2 UML Diagram of the GS Software . . . 55
6.3 GS Database Structure . . . 60
6.4 API call transaction . . . 61

7.1 Operation Center (OpCen) overview . . . 64
7.2 Satellites' Card Mockup . . . 65
7.3 Satellites' Tab Main View Mockup . . . 65
7.4 Satellites' Tab Forms Mockup . . . 66
7.5 Ground Stations' Tab Main Window Mockup . . . 67
7.6 Ground Stations' Tab Control Panel Mockup . . . 67
7.7 Passes' Tab Main View Mockup . . . 68
7.8 Passes' Tab Schedule Form Mockup . . . 69
7.9 Health Tests' Tab Main Window Mockup . . . 70
7.10 Health Tests' Tab Form Mockup . . . 70
7.11 Downlink Control Client Mockup . . . 71
7.12 Uplink Control Client Mockup . . . 72

7.13 Viewer Client Mockup . . . 73
7.14 NanoSat Lab Logo . . . 74
7.15 Graphical User Interface (GUI) Color Palette . . . 74
7.16 Backend Structure . . . 75
7.17 OpCen Database . . . 79

8.1 Many-To-One Replication Architecture . . . 81
8.2 VPN example . . . 82

10.1 Hardware access configuration . . . 90

11.1 Final system connection . . . 91

12.1 TDT Results I . . . 93
12.2 TDT Results II . . . 94

F.1 Main view of the satellite's tab . . . 195
F.2 Detailed Satellite view . . . 196
F.3 New satellite form . . . 197
F.4 Edit Satellite I . . . 198
F.5 Edit Satellite II (with delete button) . . . 199
F.6 Ground Stations tab main view . . . 200
F.7 Montsec Ground Station detailed view . . . 201
F.8 Passes main view . . . 202
F.9 Schedule pass view example I . . . 203
F.10 Schedule pass view example II . . . 204
F.11 Scheduling progress . . . 205
F.12 Scheduled passes overview . . . 206
F.13 Viewer client . . . 207

List of Tables

2.1 Available options comparative 24

4.1 Summary of the tasks to be performed 33
4.2 Responsibility Assignment Matrix 35
4.3 Annual Salary of different project roles 40
4.4 Summary of the final tasks to be performed 44
4.5 Objectives completion review 49

6.1 Hardware incidents management 62

9.1 Go vs. Python comparison summary 85

13.1 Sustainability matrix 97

14.1 Techonology licenses' summary 101

Acronyms

AOS Acquisition of Signal. 68

API Application Programming Interface. 9, 43, 51, 52, 58, 61, 62, 74–76, 82–85, 87, 90, 92–94, 103

CC Code Coverage. 85

CSS Cascading Style Sheets. 73

ECI Earth-Centered Inertial. 16, 89

ESA European Space Agency. 20, 23

GS Ground Station. 3, 9, 12, 13, 15–28, 30–33, 35, 38, 43–45, 48–52, 56, 58–64, 66, 68, 70, 71, 75–78, 80–87, 89, 91–95, 98–100, 102, 103

GUI Graphical User Interface. 4, 9, 13, 27, 64, 73, 74, 85, 86, 91, 92, 94–96

HTTP Hypertext Transfer Protocol. 76, 83

IEEC Institut d'Estudis Espaials de Catalunya. 20

LEO Low Earth Orbit. 14

LOS Loss of Signal. 68

NORAD North American Aerospace Defense Command. 15, 68, 77

OpCen Operation Center. 3, 4, 9, 12, 13, 15, 20–28, 30–33, 35, 38, 39, 43–45, 48–52, 56, 58, 61, 63, 64, 68, 69, 71, 74–82, 85–87, 91, 92, 94–96, 98, 99, 102

OS Operating System. 85

SDR Software Defined Radio. 18, 56, 57, 76

SGP4 Standard General Perturbations Satellite Orbit Model 4. 23, 24, 101

TCP Transmission Control Protocol. 89

TDT Table-Driven Test. 85, 93

TLE Two-Line Element. 15, 16, 69, 75–77

UHF Ultra High Frequency. 20, 22, 24, 49, 89, 103

UPC Universitat Politècnica de Catalunya. 20

USSTRATCOM United States Strategic Command. 15

VHF Very High Frequency. 20, 22, 24, 49, 89, 103

VPN Virtual Private Network. 82, 87, 92, 95

Contents

Acronyms **6**

1 Introduction **12**

2 Context and Motivation **14**
2.1 CubeSat missions . . . 14
2.2 Communication with Satellites . . . 15
2.2.1 Establishing communication . . . 15
2.2.2 The communication process . . . 18
2.2.3 Operations management . . . 19
2.3 Laboratory context . . . 19
2.4 Stakeholders . . . 21
2.5 Existing solutions . . . 22
2.6 Justification . . . 23

3 Project Scope **25**
3.1 Objectives . . . 25
3.2 Requirements . . . 26
3.3 Non-Functional Requirements . . . 27
3.4 Risks and obstacles . . . 27
3.5 Methodology . . . 28

4 Budget and planning **29**
4.1 Initial Project Planning and budget . . . 29
4.1.1 Task definition . . . 29
4.1.2 Main schedule and alternatives . . . 35
4.1.3 Budget . . . 38
4.1.4 Management control . . . 42
4.2 Final Project Planning and Budget . . . 43
4.2.1 New Tasks . . . 43
4.2.2 Final Schedule . . . 45
4.2.3 Effect on project cost . . . 47
4.2.4 Project feasibility and objectives . . . 48

5 Software architecture **50**

6 Ground Station Design **52**
6.1 Ground Station overview . . . 52
6.2 Detailed design . . . 52
6.2.1 Communication with the Hardware . . . 56
6.2.2 Managing Adapters access . . . 56
6.2.3 The Controllers logic . . . 57
6.2.4 The Task Executor . . . 58
6.2.5 Operations through the Application Programming Interface (API) 58
6.2.6 Database . . . 59
6.2.7 Software robustness . . . 60
6.2.8 Software scalability . . . 62

7 Operation Center Design **63**
7.1 Operation Center Overview . . . 63
7.2 OpCen GUI Design . . . 64
7.2.1 Operation Center Management Client . . . 64
7.2.2 Downlink Control Client . . . 71
7.2.3 Uplink Control Client . . . 71
7.2.4 Viewer Client . . . 72
7.2.5 UI/UX and Brand Image . . . 73
7.3 Backend and API . . . 74
7.4 Database . . . 77

8 System Communication Design **80**
8.1 Adding adapters to a GS . . . 80
8.2 Adding GS to the OpCen Network . . . 80
8.3 Data Synchronization . . . 80
8.4 API . . . 82
8.5 Secure Connection . . . 82

9 Selected Technologies **83**
9.1 Ground Station technologies . . . 83
9.2 Operation Center technologies . . . 85
9.3 Database technologies . . . 86
9.4 System communication . . . 87
9.4.1 Networking . . . 87
9.4.2 Database synchronization . . . 87
9.4.3 File synchronization . . . 88

10 Additional software **89**
10.1 Go-Satellite . . . 89
10.2 GS Adapters . . . 89

11 Deployment **91**
11.1 System set up . . . 91
11.2 Deployment of the backend . . . 92
11.3 Deployment of the OpCen GUI . . . 92

12 System Verification **93**
12.1 Software Testing . . . 93
12.2 Requirements Verification . . . 94
12.3 Non-Functional Requirements Justification . . . 95

13 Sustainability report **97**
13.1 Introduction . . . 97
13.2 Project Put in Production . . . 97
13.2.1 Design consumption, Environmental dimension . . . 97
13.2.2 Initial budget, Economic dimension . . . 97
13.2.3 Personal impact, Social dimension . . . 98
13.3 Lifetime . . . 98
13.3.1 Ecological footprint, Environmental dimension . . . 98
13.3.2 Viability plan, Economic dimension . . . 98
13.3.3 Social impact, Social dimension . . . 99
13.4 Risks . . . 100
13.4.1 Environmental risks, Environmental dimension . . . 100
13.4.2 Economic risks, Economic dimension . . . 100
13.4.3 Social risks, Social dimension . . . 100

14 Laws and Regulations **101**

15 Conclusions **102**

Bibliography **104**

A System Requirements **108**
A.1 GS Requirements . . . 108
A.2 OpCen Requirements . . . 114
A.3 System Communication Requirements . . . 118

B GS API Documentation **122**

C OpCen API Documentation **138**

D System Level Tests **179**

E Requirements verification **188**
E.1 GS Requirements Verification . . . 188
E.2 OpCen Requirements Verification . . . 190

E.3 System Communication Verification 192

F GUI Results **194**

1 Introduction

Due to the popularization of constellations based on low-cost satellite platforms (CubeSats), the Ground Segment facilities need to expand for the sake of handling all those satellite missions. The automation in the management of the Ground Segment is becoming more and more important in order to make the operators' job easier. With the recent expansion in the number of CubeSats developed in the NanoSat Lab, the need of an extense Ground Station (GS) network arises. The aim of this work is to study the different existing solutions to implement this GS network, and design a single solution that provides the NanoSat Lab with an infrastructure capable of managing a growing Ground Segment. This solution must then be capable of handling an extensible network of GSs which will be managed by a single Operation Center (OpCen).

First of all, the concept of CubeSats, their motivation and different uses, and the way they communicate with the ground segment is explored thoroughly. Furthermore, the latest project that has prompted for this work is introduced, together with its different requirements. Then, the stakeholders related to this project are analyzed, and at which level each of them would benefit from the project. Afterwards, existing solutions are discussed and analyze if they would fit the project need.

Next, the main objectives and requirements of the project are presented along with an evaluation of risks and obstacles that may need to be faced along the way. This section concludes with the definition of the methodology used for the development of the project. Afterwards, an initial planning and budget is presented, together with a report of the modifications and consequences of the changes in those.

The following sections include a detailed description of all the development process. It starts with the design phase where the architecture of the solution is presented. Starting with an overview of the whole system, each section will further delve into each subsystem. The GS software is then presented, followed by the OpCen software and the communication of those two systems.

After all the design has been elaborated, different technologies that might fit the design requirements are explored and evaluated. For each component of the system,some commonly used technologies are studied, and then a final decision is reached.

Next, the specific software that has been implemented as a byproduct of the main software is explained. There, it is shown how the system was set up, including the deployment in the GSs, and the OpCen backend and the Graphical User Interface (GUI) for the operators.

The development process ends with the verification phase. There, the methodology used to test the correct performance of the software is explained and detailed documentation of which things have been tested is provided. Finally, the initial requirements accomplishment is verified.

Afterwards, a sustainability report is presented in order to reflect on the social, environmental and economic impact of the project and what it contributes to society.

The document ends with the conclusions, where a self-reflection is made on the fulfillment of the objectives initially set, future work, and what the project has contributed on personal growth.

2 | Context and Motivation

2.1 CubeSat missions

A CubeSat is a type of miniaturized satellite that is composed of CubeSat units, which are a cube with 10 cm x 10 cm x 10 cm of envelope. CubeSats have a mass of no more than 1.33 kilograms per unit (1U). They are developed to participate in scientific or technological missions where they are typically launched in a Low Earth Orbit (LEO) having an altitude of 400-500km approximately.

Traditionally, all space missions missions have two main segments, as Figure 2.1 presents. The first one is the Space Segment which is composed of the satellites themselves. The second one is the Ground Segment, which consists of all the elements used to design, manage and conclude space missions, such as testing facilities, and infrastructure to operate and control the spacecraft.

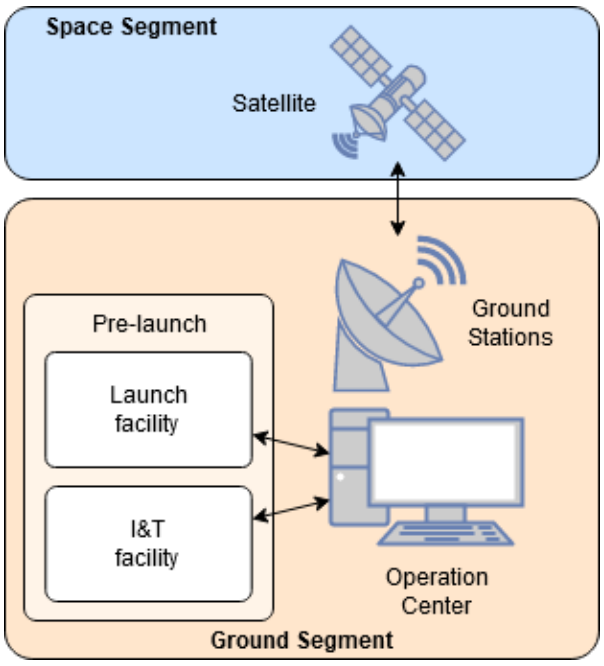


Figure 2.1: CubeSat missions segments

The Ground Segment includes different support equipment essential to achieve the mission. The common and more important facilities of this segment are the GS and the OpCen. The GSs are the installations used to communicate with the satellite, uploading telecommands and downloading data. Therefore, it includes all the antennas, and radio interfaces to achieve this goal. They contain the hardware and software that makes possible the communication with the satellite. The OpCen is where the missions are managed. All the personnel manning the diferent stations in an operation center are called the operators, and their competences range from monitoring the output of the satellite or preparing the commands to upload, to maintaining the GSs infrastructure to communicate with the satellite. The OpCen is connected to GSs using a specific network known as the GS Network, enabling operators to manage the GS and the spacecraft remotely.

2.2 Communication with Satellites

In order to have an effective communication, the Ground Segment and the satellite must speak the same language, or in other words, they have to interpret the messages of each other. Usually, there is a fixed number of messages that can be understood, which are the known *commands*. This computer-generated commands have to be sent through radio waves, which poses two challenges: (1) being capable of transmit and receive radio waves, and (2) retrieve the exchanged data from the radio interface. In the following sections we will discuss how those two challenges are normally faced.

2.2.1 Establishing communication

Satellites are constantly moving, following an orbit trajectory. This trajectory can be modeled using orbital parameters. These parameters are formatted in the well-known Two-Line Element (TLE) File. It is provided by either the satellite's launcher (i.e. Arianespace, SpaceX) or the United States Strategic Command (USSTRATCOM)[1] afterwards. Websites such as Celestrak provide the most recent Two-Line Element (TLE) files identified by the North American Aerospace Defense Command (NORAD) Satellite Catalog Number, which is the unique ID each satellite has.

In order to communicate with the Ground Segment, radio frequency transceivers are used. Satellites have a configured *downlink frequency* for sending data to the Ground Segment and an *uplink frequency* for receiving commands, as it can be seen in Figure 2.2. Although the figure presents this uplink and downlink separately, this information exchange can be done in the same GS.

These communications can only be achieved when the satellite is flying over the GS. Due to the satellite motion, the spacecraft remains visible from a GS during a lapse of

[1]The USSTRATCOM is in charge of detecting, tracking, identifying, and maintaining a catalog of all man-made objects in Earth orbit.

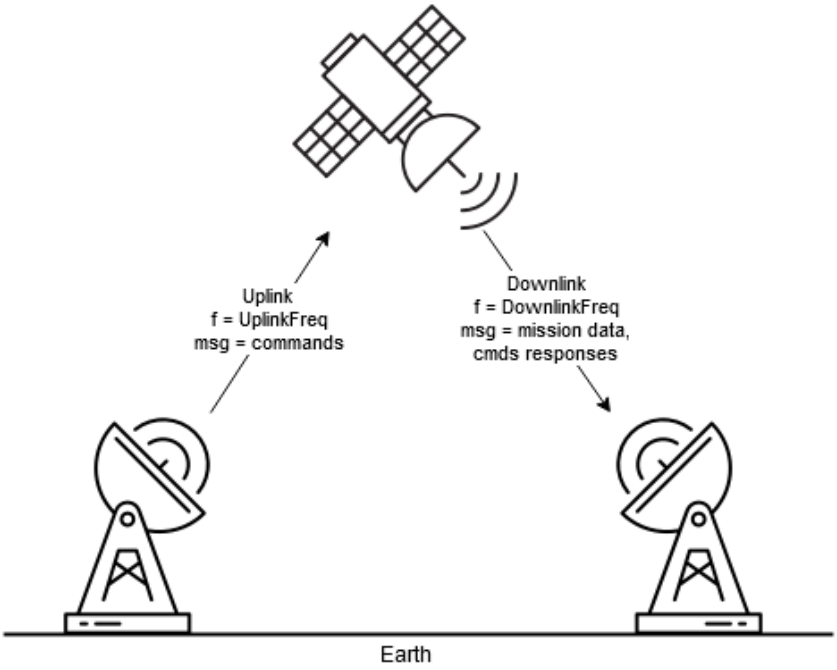


Figure 2.2: Basic communication between Earth and Satellites

time, called GS pass. During this contact, the satellite changes its position with respect to the GS one. Therefore, the GS has to be capable to point its antennas towards the satellite. The interval of the orbit in which the satellite will be visible is known as the *coverage interval*. When a satellite travels above the coverage interval, the *GS pass* is feasible.

So, two things are needed to effectively communicate with the satellite during this pass: (1) defining where the antennas have to point and (2) being able to receive and transmit at the required frequencies. Figure 2.3 shows a schema of the process, used to achieve those two objectives.

As mentioned previously, the TLE file contains all the parameters to calculate the position and velocity of a satellite at any given time. With this TLE and a list of timestamps, it is possible to generate a sequence of position and velocity using an orbit propagator. It is basically a software implementation of one of the Simplified Perturbations Models [2], which are a set of Keplerian calculations that predict the position and velocity of the satellite relative to Earth-Centered Inertial (ECI) coordinate frame[2] [3].

These coordinates indicate where the satellite is, but do not take into consideration the relative position with respect to the GS. Therefore, a transformation of coordinates

[2]ECI Reference Frame have their origins at the center of Earth and do not rotate with respect to the stars.

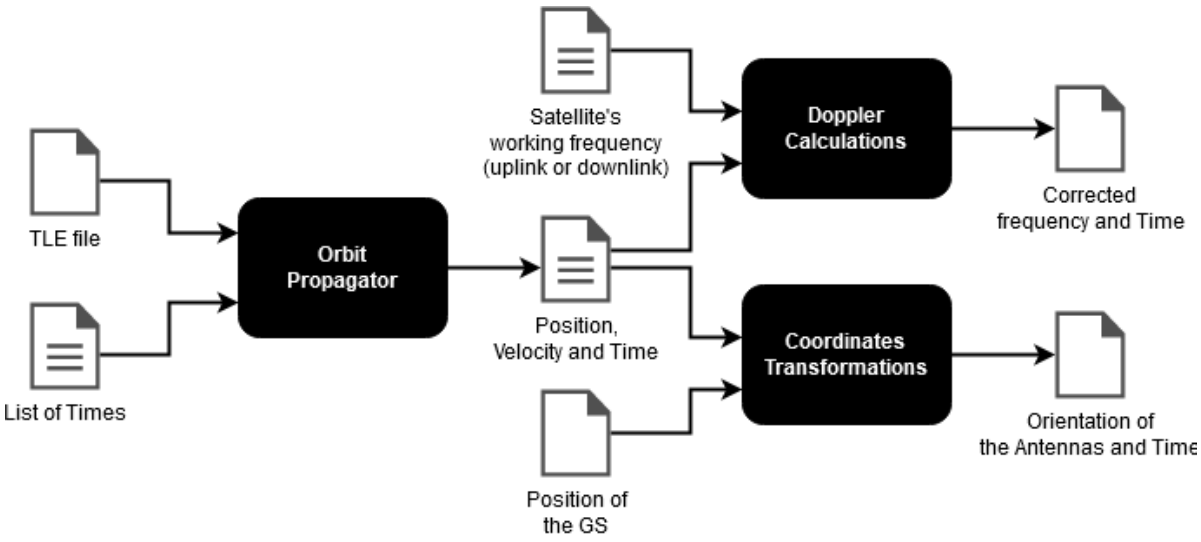


Figure 2.3: Communication with Satellites Requirements

is performed to retrieve the Azimuth and Elevation angles of the GS antenna that points to the satellite. This frame is illustrated in Figure 2.4. Azimuth tells what direction to face and Elevation tells how high up in the sky to look. Azimuth ranges from 0 to 360 degrees, starting North (X axis) and increasing clockwise and Elevation from 0 to 180 degrees. If we remove from the list the points where the Azimuth and Elevation are not achievable by the antennas (i.e. negative elevations), we get all the positions and instants of time to effectuate a pass.

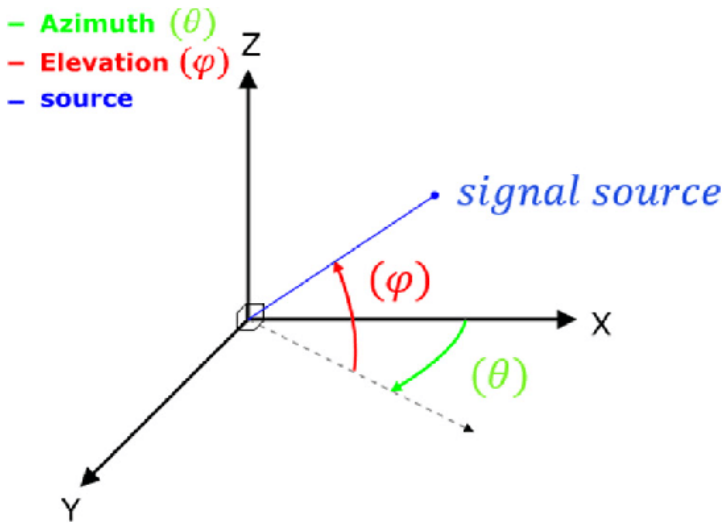


Figure 2.4: Elevation and Azimuth angles for a signal emitter [1]

At this point of the process, the first problem is already solved. The only thing

missing is the radio frequency for transmission and reception. As there exists a relative movement between the Satellite and the GS, the radio waves suffer from the so-called Doppler effect. The Doppler effect is the change in the instantaneous frequency due to the relative velocity between two nodes.

Due to the Doppler effect, the frequencies at which we have to receive or transmit to the satellite vary depending on the relative velocity it has at each point of time. As the velocity is known thanks to the Orbit Propagator, and the frequency of the satellite is also known, the Doppler shift can be calculated and thus the frequency can be compensated accordingly.

Considering $\overrightarrow{P}_{SAT}$ and $\overrightarrow{V}_{SAT}$ the satellite position and velocity respectively, and $\overrightarrow{P}_{GS}$ and $\overrightarrow{V}_{GS}$ the GS position and velocity respectively, the relative position $\overrightarrow{P}$ and velocity $\overrightarrow{V}$ are defined in Eq. (2.1) and (2.2).

$$\overrightarrow{P} = \overrightarrow{P}_{SAT} - \overrightarrow{P}_{GS} \tag{2.1}$$

$$\overrightarrow{V} = \overrightarrow{V}_{SAT} - \overrightarrow{V}_{GS} \tag{2.2}$$

The range rate (RR) defines the rate of change between the satellite and the GS. This means that it is calculating the component of the satellite's velocity that would increase or reduce the Doppler effect.

$$RR = \frac{\overrightarrow{P} \cdot \overrightarrow{V}}{|\overrightarrow{P}|} \tag{2.3}$$

Applying the frequency Doppler definition with the received frequency (f) to get the Corrected Frequency (CF).

$$CF = f \cdot (1 - \frac{RR}{c}) \tag{2.4}$$

2.2.2 The communication process

In order to convert the commands from bits to radio waves and viceversa, it is important to follow a few steps. The process from bits to radio waves is explained below, but as it can be seen in Figure 2.5, the process can be applied in reverse.

First, the bits need to be transformed into a signal modulated by amplitude, phase and/or frequency . This modulation is done using GNURadio[3], a software which performs the required signal processing. That output is then transmited to a Software Defined Radio (SDR) that transforms the processed bits into a voltage signal, that is

[3]**GNURadio:** https://www.gnuradio.org/

then upconverted to the transmitting frequency and finally amplified. Finally, the antenna transmits signals in the direction where the antenna is pointing.

The satellite is then able to do the reverse conversion by collecting, downconverting, and sampling the radio waves and transforming them into phase and quadrature, which can then be demodulated and decoded into the original signal.

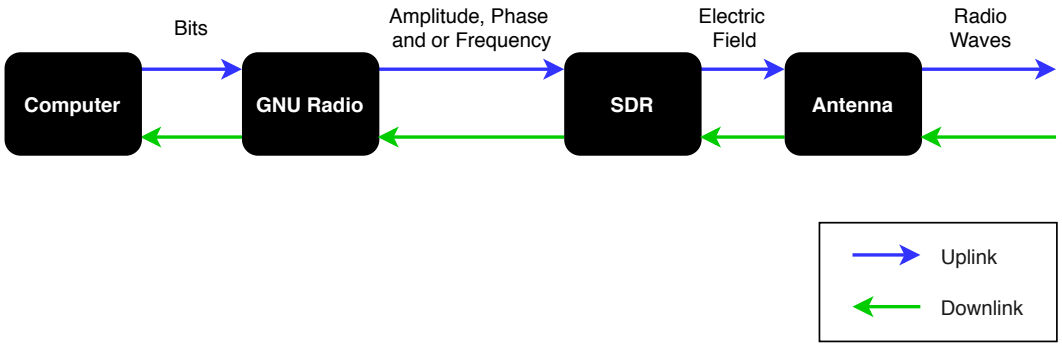


Figure 2.5: Commands Conversion

2.2.3 Operations management

When operating the satellite, there are three main roles that need to be covered by the operators. The GS manager, the responsible of the Uplink and the responsible of the Downlink.

- **GS Manager:** The GS Manager is responsible for scheduling passes, executing health tests to guarantee the correct functioning and performance of the GS before, during and after performing operations like a satellite pass.
- **Uplink Manager:** The Uplink Manager is the operator in charge of planning the commands flow for one pass and sending them to the satellite at each scheduled time.
- **Downlink Manager:** Is the operator in charge of receiving and analyzing the telemmetry of the satellite. Additionally, she/he must inform to the Uplink Manager on what to do next.

Those three roles work together in a coordinated manner, but each of them focused on its own specific tasks.

2.3 Laboratory context

The work presented in this report has been conducted in the Nano-Satellite and Payload Laboratory (NanoSat Lab) [4] of UPC. It is an initiative led by Professor Adriano

Figure 2.6: Montsec GS

Camps belonging to the Department of Signal Theory and Communications, located in the Universitat Politècnica de Catalunya (UPC) – UPC BarcelonaTech (Campus Nord). Its main objective is the development of CubeSats' [5] subsystems and payloads used for Earth Observation and Communications. The goal of the laboratory is mainly educational, meaning that it is made by students who want to learn about these topics. It offers different fields of study, from hardware to software, and try to have a global vision of satellite missions, from the Space Segment (CubeSats and their respective payloads) to the Ground Segment (GSs and OpCen) in order to have autonomy managing the mission's operation and data acquisition. The thesis is part of one of the missions of the lab, particularly, the FSSCat, which is presented below.

At this moment, the laboratory has integration and testing facilities, and a GS installed at the facilities of the Montsec Observatory [6] managed by the Institut d'Estudis Espaials de Catalunya (IEEC), which can be seen in Figure 2.6. This GS is equipped with S-Band, Ultra High Frequency (UHF) and Very High Frequency (VHF) chains. Currently the laboratory lacks an OpCen where the missions can be managed efficiently instead of accessing the GS directly.

The development of the 3Cat-4 (read "cube-cat-four") is being conducted currently in the laboratory. As mentioned in the web page [7], it is the fourth member of the CubeSat series of UPC's NanoSat Lab. This mission aims at demonstrating the capabilities of nano-satellites, and in particular those based in the 1-Unit CubeSat standard, for challenging Earth Observation using Global Navigation Satellite System - Reflectometry and L-band microwave radiometry, as well as for Automatic Identification Services. There is also a new satellite mission in development with the European Space Agency (ESA), the FSSCat [8], that should had been launched on March 23rd. The mission is an innovative concept consisting of two federated 6-Unit Cubesats, called 3Cat-5/A

and 3Cat-5/B, in support of the Copernicus Land and Marine Environment services. The FSSCat mission is the winner of the *2017 Copernicus Master ESA Small Satellite Challenge S3* and *Overall Winner*. Finally, other space-related companies like Open Cosmos [9] are also interested in using testing facilities and the Montsec GS so they can send and receive data from their own satellites.

Thus, the laboratory needs an OpCen that can communicate with a remote GS that is independent of the hardware available in each of the transmission / reception chains in the GSs. The downloaded data needs to be managed for further processing or forwarded to external clients if it is the case. The idea is to be able to have more than one GS in the future and that all of them can be controlled from the same OpCen. This is the reason of proposing a modular and reusable software, which only changes in the way it accesses to the specific hardware of each transmission chain. Having this, the laboratory can easily and quickly expand the number of GSs with new installations.

2.4 Stakeholders

The project has many involved parties, which can be ordered by how close they are to the final product and how much they interact with it, as shown in Figure 2.7. The first circle represents the stakeholders who have direct interaction with the product, the second circle is composed of the ones who receive the direct benefits from it, and in the outer circle the ones who indirectly benefit from it.

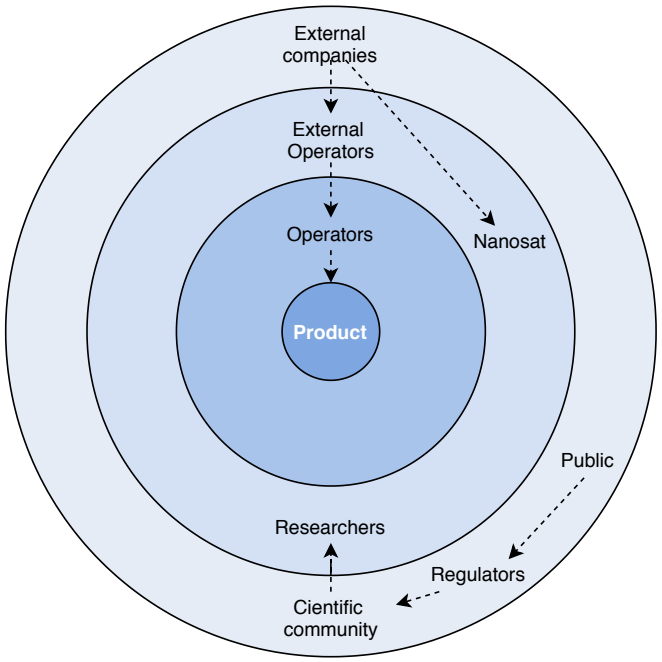


Figure 2.7: Stakeholders diagram

The stakeholders that have *direct interaction* with the product, and thus inside the first circle, are the operators. The **operators** are the ones that work the OpCen and use the product to manage the missions.

On the next level of the diagram, there are the stakeholders that receive the **benefits** from the product, the **NanoSat Lab**, which gets economic income from companies that want to use the product and the facilities, the external operators from these companies, and the **researchers** that receive the downloaded data from the missions. The **external operators** are the external company members that are in charge of providing the needed information to the internal operators when their company use the NanoSat Lab facilities. The researchers can be NanoSat Lab students who need the data for their projects or any other kind of researchers that are involved with the NanoSat Lab missions.

Finally, the stakeholders that **indirectly benefit** from the product, and thus inside the first circle, results are the **scientific community**, who get access to the studies made with the mission data, and the **regulators** and **public** who can act accordingly to what has been discovered by the studies mentioned above. **External companies** also benefit from the data received when contracting the NanoSat Lab facilities.

2.5 Existing solutions

A market research has been conducted in order to analyze the features that similar softwares offer in order to decide if any of them can be incorporated or adapted for the final solution. The most significant ones are listed as follows:

- **SatNOGS [10]:** It is a global and collaborative network of GSs designed as an open source project. The main idea is that you can add your own GS to the network so that you can share the data received with other people, and they can use it if allowed to. It comes with loads of information of how to build your own GS to collaborate to the project. Moreover, it normally operates with UHF and VHF bands.
- **Amazon Ground Stations [11]:** Amazon Web Services owns a global network of GSs which can be used to communicate with your satellite. It includes uplink and downlink using X-Band and S-Band frequencies. It requires payment in order to use its facilities.
- **Kongsberg Ground Stations (KSAT) [12]:** Similar to Amazon's Ground Station Service, Kongsberg offers a global network of GSs with uplink and downlink options. Moreover, in respect to Amazon's offerings, Kongsberg has a wider reach in regards to the bands they offer and includes UHF and VHF frequencies alongside the S-Band mentioned above.

- **GENSO [13]:** Although is not an operative solution, it is worth mentioning that the ESA had the first initiative in which they tried to develop a confederated network of GSs. This project wanted to provide a common software for different GSs hardware but in the end, it was never finished.

Some utility tools and libraries have also been studied to determine if they could be integrated to the project as partial solutions. This resources include orbit propagation libraries and data visualization software.

For the orbit propagation, the Standard General Perturbations Satellite Orbit Model 4 (SGP4) [14] is the main contender, as it is the one used in the NanoSatLab and largely accepted by the scientific community due to its accuracy. Celestrak has the code of that propagator in Fortran [15], but it can be translated to any other language. Some of the already existing implementations are:

- **sgp4 (Python):** Official Python library [16]
- **satellite-js (Javascript):** Javascript transcript [17]
- **go-satellite (Golang):** This implementation is transcribed from Celestrak. It has tests to ensure accuracy [18]
- **sgp4 (C++):** It is also a transcription from the code provided by Celestrak [19]

For the data visualisation software, two open source alternatives are available:

- **Chartist-js:** It is a data visualisation tool written in Javascript [20]. It has many different chart types and maps and is easy to load your own data.
- **OpenMCT:** It is an open-source data visualisation tool created by the NASA to aid in space missions [21]. It offers some chart types and other mission control options.

2.6 Justification

As shown in Table 2.1, none of the options fulfill completely the problem mentioned above. Neither SatNOGS or Amazon offer the desired frequency bands. Any of the options allow us to use our own OpCen, they all give an interface to work with, but cannot be adapted to our specific needs. Finally, all of them (except SatNOGS) are closed, so the code cannot be reused to make our own system. The only one that allows us to use our already existing GS is SatNOGS, but it would imply to open the implementation opening it to the community and might not be accessible 100% of the time nor managed by the NanoSat Lab. As none of the options fulfill the project needs, the backend of the GS and OpCen will be designed from scratch. Furthermore, it will also be used in educational projects [22].

Table 2.1: Available options comparative

	SatNOGS	**Amazon Ground Stations**	**Kongsberg Ground Stations**
Includes S-Band	No	Yes	Yes
Includes UHF/VHF	Yes	No	Yes
Can use own OpCen	No	No	No
Can use own GS	Yes	No	No

Regarding the partial solutions, the SGP4 can be reused as it is available in many languages, so there is no reason to write it again. The data visualisation tools can be reused too, so for the OpCen there will be needed to write an adapter to those interfaces and modify them to match the required structure and interactions.

3 | Project Scope

3.1 Objectives

The main objective of the thesis is **to design an Operation Center that accepts a network of multiple Ground Stations following a modular architecture**. By creating this structure, we could resolve the problem described previously. To achieve the aforementioned objective, the following secondary objectives have been defined.

OB 1: **Design a software that guarantees that all the roles of the operators can be covered**. It determines at a design level which are the general tasks that the software has to enable.

- OB 1.1: The GS manager can check the well-functioning of the GSs.
- OB 1.2: The GS manager can schedule passes.
- OB 1.3: The downlink manager will be able of receiving data in real time.
- OB 1.4: The uplink manager will be able to plan the uplink commands
- OB 1.5: The uplink manager will be able to send commands to the satellite.
- OB 1.6: All the personnel is kept informed during the operations.

OB 2: **Control GSs from one single place in order to operate satellites.** It defines the main components of the product that need to be implemented, which can be summarised as follows:

- OB 2.1: An OpCen which is connected to a number of GSs.
- OB 2.2: GSs that can communicate with satellites.
- OB 2.3: Build an infrastructure and its required links to become capable of sharing data between the GSs and the OpCen, with the latter acting as a central hub.

OB 3: **Schedule satellite passes automatically.** This objective requires the participation of both the Operation Center and the GS.

- OB 3.1: Being able to schedule passes from the Operation Center.

OB 3.2: Having a Ground Station software which is capable of executing the pass automatically. *This means that it has to be able to move the antennas automatically at the precalculated point, turn on the correct RF chain and download received data to the correct location.*

OB 4: **All the data should be centralised in the Operation Center**. Data from all Ground Stations should be moved to the Operation Center.

OB 4.1: Find a method of synchronising the data received in the Ground Stations with the Operation Center.

OB 5: **Provide the capability for the operator to enable external companies to have access to scheduling functionalities.**

OB 6: **Communicate with the existing Ground Station Hardware**. This means creating adapters to interface with prior hardware to substitute the existing software. This results in two minor objectives:

OB 6.1: Create an interface to the existing UHF / VHF chain with the new software structure.

OB 6.2: Create a brand new interface to the new S-Band chain.

3.2 Requirements

In order to plan the development of the aforementioned objectives, a list of requirements was agreed in coordination with the future users of the software (i.e. the operators). A global view of the requirements is presented in this Section. If the reader is more interested on the requirements, address to Appendix A. They are divided according to the subsystem they belong to, including:

- **Ground Station Requirements** refer to all the functionalities (software logic) that the GS has to perform, how the managed data should be stored and how the interface to the GS should be managed. It also includes some design requirements in order to make the software reusable.
- **Operation Center Requirements** denote the requirements related to how the final user communicates with the software user interface, how the OpCen software should work and how it should communicate with the GSs.
- **System Communication Requirements** define the data that needs to be shared between the two subsystems and how it has to be shared, including security and communication protocols.

The requirements are ordered by priority where *high priority* requirements are the ones needed to accomplish the implementation objectives, while the other ones are required at a design level and make reference to the first objective.

Having a list of all the necessary requirements will help to verify the software once the project is finished and analyze which objectives and requirements have been met.

3.3 Non-Functional Requirements

Non-Functional Requirements are requirement that specify criteria that can be used to judge the operation of a system, rather than specific behaviors. These are the non-functional requirements needed to ensure the quality of the final product.

NF-REQ 1: Data exchanges should be done through a secure connection.

NF-REQ 2: The GS software should be scalable and modular.

NF-REQ 3: The GS software has to be reliable and should be able to recover from errors, such as internet or electricity outage.

NF-REQ 4: There has to be data integrity between the GSs and OpCen.

NF-REQ 5: The OpCen's software has to be usable, as it is the one managed by real users.

NF-REQ 6: The OpCen software GUIs have to keep the same look and feel and brand image.

3.4 Risks and obstacles

During the execution of a project, there are always risks and obstacles that need to be detected as soon as possible. In this project, these are the main risks that have been detected.

- **Working remotely:** As the software needs to be distributed, we have the risks associated with working remotely, as there will be no direct easy access with the Ground Stations (i.e. in case of Internet or electrical failure). In order to deal with that obstacle, simulators of the real hardware (same interface) will be developed if necessary to cope with the situations where the hardware is unavailable (i.e while the antenna is not yet installed).
- **Hardware incidents:** Hardware is also an important source of obstacles. It can fail, stop working or even if some new components are needed it might take some time to fix or purchase it. This means that the planning may be delayed until the hardware works properly. In order to minimise the delays, the project is planned with concurrence so other tasks can be done while the equipment is not ready.

- **Testing the hardware:** In order to test the interface between the GSs software and the hardware, it will be necessary to go to Montsec, which will result in some delays too. Working remotely and preparing the tests before going to Montsec can reduce the number of trips needed to test everything.
- **Difficulty with external interfaces:** The hardware might not have clear interfaces, thus requiring extra work of documenting and adapting the software structure to special requisites.

3.5 Methodology

As the final product will be a complex system, we chose to work with Iterative and Incremental development methodology [23]. In this methodology the whole system is built through different iterations, having new functionalities at the end of each one. Each iteration follows a pattern: (1) choose some functionalities, (2) implement them, (3) test and check that the associated requirements are met, (4) product deployment. The main idea is to have a working version with more functionalities for each iteration, although it may not be achievable on the first ones.

As the software is also distributed, the development process will be divided in three main blocks. Each block will follow the methodology mentioned above. We will first focus on the GS software, followed by the OpCen one, and finally, connecting both of them.

At the beginning of the process, the requirements needed will be documented, and the definition of *verified* established. By doing so, it will be easier to verify that the requirements are met once implemented. Once the requirements are fixed, we will focus on a general design of the system, followed by a specific design of each one of the main blocks. Each block will then be developed and tested before moving on to the next one.

In order to determine which tasks are fixed on each iteration weekly meetings will be planned. There, the previous iteration's work will be analysed and verified, and based on the results, new tasks will be scheduled for the upcoming iteration. In order to verify the implemented software, unit tests, integration tests and requirements' verification will be performed. The results of these tests will become the final checklist signalling the completion of the project.

4 | Budget and planning

4.1 Initial Project Planning and budget

The estimated project duration was approximately one year. It started on March 2019 and it was planned to end at the 25th of January 2020.

4.1.1 Task definition

The tasks can be divided in two parts: the project management and the project development ones. There is a summary of all the tasks in Table 4.1.

The **project management** tasks include project organisation documents and weekly meetings. They require a laptop as main resource.

- **PM1 - Context and Scope:** Defining the context and the scope of the project takes about 15 hours and requires *Overleaf*[1] and *draw.io*[2].
- **PM2 - Time Planning:** Making a time plan for the project requires working with *Overleaf*, *GANTTproject* and *LibreOffice Calc*. It takes 15 hours to be completed.
- **PM3 - Budgets and Sustainability:** Making a budget plan and sustainability analysis for the project requires 15 hours and working with *Overleaf* and *LibreOffice Calc*.
- **PM4 - Project Definition:** Grouping all the documents mentioned above to generate a Project Definition file. Requires *Overleaf* and needs PM1, PM2 and PM3 to be completed.
- **PM5 - Weekly meetings:** This are weekly meetings with the directors to analyse the evolution of the project. It requires one hour per week and the reports from that week's work.

The development of the project (itself), is composed of different software development phases: inception, design, implementation, testing, and deployment.

[1]Overleaf: https://www.overleaf.com
[2]Draw.io: https://www.draw.io

The **inception** phase aims at clarify the concept of the project by defining the objectives, requirements, and the mechanisms used to verify them. All the inception tasks require a laptop as the only resource.

- **IN1 - Define objectives:** This task consists of defining and documenting the objectives of the project. It is estimated to take about 5 hours to complete and has no predecessors.
- **IN2 - Define requirements:** In this task the requirements of the project are written and documented, ordered by the type of requirement it represents (functional, non-functional, from the OpCen, from the GS, etc.). Its estimated time is about 15 hours and requires IN1
- **IN3 - Requirement verification methods:** Consists of defining which verification method is used in order to ensure that the final project meets all the requirements. This task takes about 5 hours to be completed and requires task IN2 to be complete.

The **design phase** deals with transforming the gathered requirements into a feature that can be implemented later. It defines the system architecture, the interface in which the software can be accessed and the detailed behaviours of some logic units. The *design phase* includes the general architectural System Design (SD1), the GS with its database design, and the OpCen with its database design. All the design tasks require the same resources which are a laptop and *draw.io* software to make the diagrams.

- **SD1 - System Architecture Design:** Consists of defining the general architecture of the software, treating the OpCen and the GSs as black boxes and defining the interfaces between them. It takes about 5 hours to complete and needs all the inception tasks to be finished in order to design accordingly to the specified needs.
- **SD2 - GS Design:** Designing the software architecture of the GS. Specifically, the interfaces with the hardware, the interface with the OpCen, the logic layer and the access to the database. It takes about 20 hours to complete as it is a more detailed design for a medium size software. It requires the SD1 to be finished.
- **SD3 - GS Database design:** Designing the database structure and choosing one database type. Requires SD2 to be finished in order to know which data is needed and how it is planned to be accessed. It only takes 2 hours to be finished as the SD2 defines in detail the database requirements.
- **SD4 - OpCen Design:** Defining the design of the OpCen software architecture, including the user interface basics, backend functions, database accesses and connection with the GS interface. It requires 10 hours of time and the SD1 design to be complete. SD2 is not required because the interface with the GS is already defined in SD1.

- **SD5 - OpCen Database:** Designing the OpCen Database with compatibility with the one in the GSs. It takes 2 hours and the SD4 task complete in order to know which data needs to be accessed and how.

In the **implementation phase**, the product is built according to the specified design. It includes the GS and the OpCen implementation. Each of them requires that its design is completed, meaning that for the GS implementation, SD2-3 need to be completed and for the OpCen implementation, SD4-5 must be finished. In all the implementation tasks, the resources used are a laptop, git repositories, the design diagrams and visual studio code.

- **GS1 - GS Interface:** Implementation of the interface from where the OpCen connects with the GSs. It takes about 10 hours as it does not require any complex code.
- **GS2 - GS Logic:** Implementation of all the logic the GS needs to conduct its functionalities, including access to the database. It takes 50 hours.
- **GS3 - GS Hardware communication:** Implementation of the communication between the GS software (mostly the logic part) and the transmision chains' hardware. It takes 50 hours.
- **GS4 - GS Database:** Implementation of the database creation code and setting it to a local environment for further testing. Requires 2 hours as the design is already defined in detail.
- **OC1 - OpCen GUI:** Implementation of the OpCen user interface. It s about 50 hours.
- **OC2 - OpCen Backend:** Implementation of the OpCen backend software which communicates with the database and the GSs. It does not require the GS to be finished as the interface is already defined in the SD1. It takes 50 hours to complete.
- **OC3 - OpCen Database:** Implementation of the OpCen's database creation code. Requires 2 hours as the design is already defined in detail.

In the **testing phase**, a project plan is developed in order to find whether the product meets the requirements and if it behaves as it should in all scenarios. During the *testing phase* the GS and the OpCen are tested according to the project plans. The resources used are the same for both tasks, including visual studio code, a laptop and git.

- **TT1 - GS Testing:** To conduct the unit tests and integration tests. For each unit test, the specific implementation task needs to be complete in order to get a valid report. For the integration test, both of the tasks involved need to be completed. This task takes about 50 hours. When testing the Hardware communication, the actual hardware or at least, a simulator with the same interface and behaviour needs to be accessed.

- **TT2 - OpCen Testing:** To conduct the unit tests and integration tests. It takes about 50 hours and the implementation task related with the functionality to be tested, has to be finished.

- **TT3 - System Level Testing:** Testing the system as a whole. It requires DP3 to be completed and it is expected to take 20 hours.

The **deployment phase** puts the product into production. It can be divided in two tasks, the GS software deployment and the OpCen deployment.

- **DP1 - GS Deployment:** To deploy the GS software requires 20 hours. In order to deploy the code to the GSs, the git repository has to be accessed and a laptop with remote access to the GS's computer is required.

- **DP2 - OpCen Deployment:** To deploy the OpCen Software requires 20 hours. In order to deploy the software, there needs to be access to the OpCen's server and a laptop with the git repository.

- **DP3 - System Connection:** To connect the two deployed softwares (DP1, DP2) in order to have a full system deployment. It requires access to both softwares and takes about 10 hours.

Finally there is a **verification phase**. Its main purpose is to verify that all the tests that passed in the testing phase, still pass in its production build. It also checks which requirements are met at that iteration. There, the deployed project is verified and the documentation finished.

- **PV1 - Project Verification:** To check that the final project meets the requirements and verification criteria established during the inception phase. It requires DP1 and DP2 to be completed and to have access to the OpCen software. It takes about 20 hours to be completed.

- **DC1 - Writing documentation:** It includes writing all the documents needed before the project ends. It takes about 60 hours or more and requires a laptop with *Overleaf*, *draw.io* and all the produced material.

To summarize, in Figure 4.1, there is a schema of the workflow for each phase. The roles assigned to each phase are found in Table 4.2.

Table 4.1: Summary of the tasks to be performed

ID	Name	Time(h)	Predecessors	Resources
	Project Management			
PM1	Context and Scope	15		PC, *draw.io*, *Overleaf*
PM2	Time Planning	15		PC, GANTTproject, *Overleaf*, calc
PM3	Budgets and Sustainability	15		PC, *Overleaf*, calc
PM4	Project Definition	10	PM1, PM2, PM3	PC
PM5	Weekly Meetings	20		Work Reports
	Inception			
IN1	Define objectives	5		PC
IN2	Define requirements	15	IN1	PC
IN3	Establish requirements verification methods	5	IN1	PC
	Software Design			
SD1	System Architecture Design	5	IN1, IN2, IN3	PC, *draw.io*
SD2	GS Design	20	SD1	PC, *draw.io*
SD3	GS Database Design	2	SD2	PC, *draw.io*
SD4	OpCen Design	10	SD1	PC, *draw.io*
SD5	OpCen Database Design	2	SD4	PC, *draw.io*
	Software Implementation			
	GS Software			
GS1	GS Interface	10	SD3	PC, visual studio code, design diagrams, git
GS2	GS Logic	50	SD3	PC, visual studio code, design diagrams, git
GS3	GS Hardware communication	50	SD2	PC, visual studio code, design diagrams, git
GS4	GS Database	2	SD3	PC, visual studio code, design diagrams, git
	OpCen Software			
OC1	OpCen GUI	50	SD4	PC, visual studio code, design diagrams, git
OC2	OpCen Backend	50	SD5	PC, visual studio code, design diagrams, git
OC3	OpCen Database	2	SD5	PC, visual studio code, design diagrams, git
	Testing			
TT1	GS Testing	50	GS1, GS2, GS3, GS4 *	PC, visual studio code, git, hardware access
TT2	OpCen Testing	50	OC1, OC2, OC3	PC, visual studio code, git
TT3	System Level Testing	20	DP3	Access to both softwares
	Deployment & Conclusion			
DP1	GS Deployment	20	TT1	PC, git, GS remote access
DP2	OpCen Deployment	20	TT2	PC, git, OpCen server access
DP3	System Connection	10	DP1, DP2	Access to both softwares
PV1	Project Verification	20	DP1, DP2	OpCen access
DC1	Writing documentation	60	ALL **	PC, *Overleaf*, *draw.io*, project resources
	Total:	**603**		

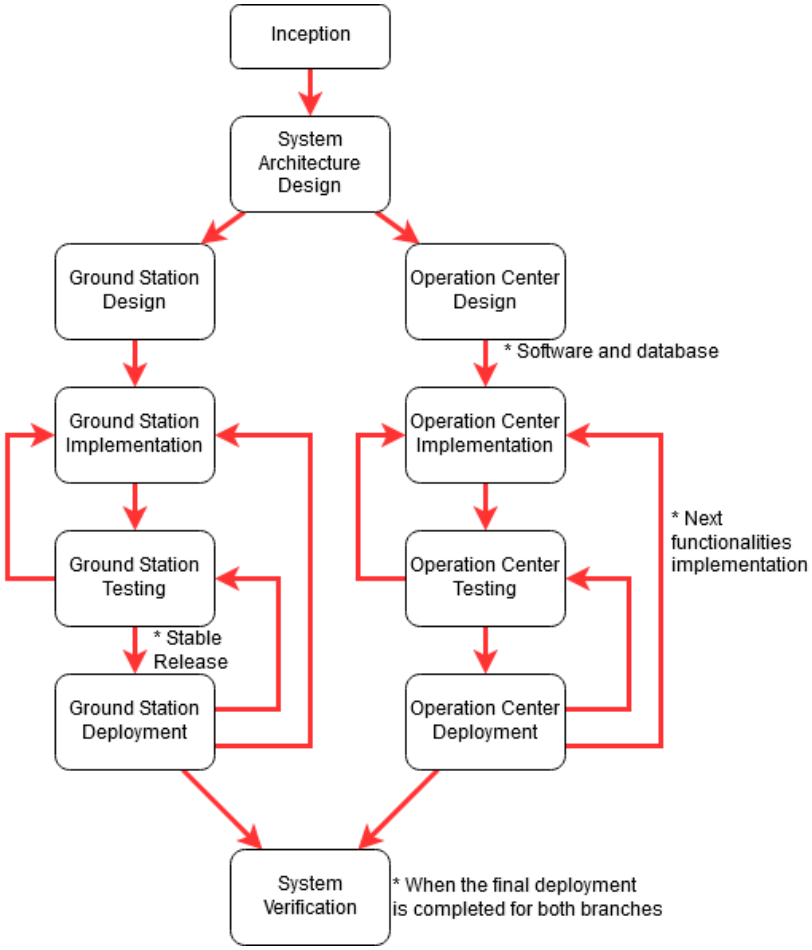


Figure 4.1: Workflow dependencies

Table 4.2: Responsibility Assignment Matrix

R - Responsible C - Consulted I - Informed **Work Package**	**TW**	**DO**	**P**	**SA**	**T**	**SWA**	**PO**
Project Management	R						C
Inception				R			
Design				C		R	I
Implementation			R			C	
Testing			C		R		
Deployment		R					I
Verification		R					I
Documentation	R						C

TW: Technical Writer
DO: DevOps
P: Programmer
SA: System Analyst
T: Tester
SWA: Software Architect
PO: Product Owner

4.1.2 Main schedule and alternatives

4.1.2.1 GANTT diagram

In this section, an overview of the complete schedule is provided in Figures 4.2 and 4.3. The project starts with the inception phase. After all the tasks in that phase are completed, the OpCen or the GS design phase can begin.

The GS software is the starting point, as controlling the hardware comprises the minimum viable product to operate the satellite without needing an OpCen. Thus GS software design, implementation and testing tasks were scheduled. In order to ensure that each layer was working properly, the functionalities were implemented from the closest to the hardware to the closest to the final user. Thus, functionalities were added in an incremental order so at the end of each iteration, a stable release of the software is produced and the integration with the previous layers can be tested. Tests should be done within that iteration before undertaking development for new functionalities. Once the GS is done, the same process is used with the OpCen. During all the process, documentation is written as soon as a task is completed.

Begin date	End date	ID	Durat...	Name	Coordinator	Risk
3/12/19	3/27/19	INX	12	Inception		Low
3/12/19	3/15/19	IN1	4	Establish requirements verification methods	System Analyst	Low
3/18/19	3/21/19	IN2	4	Collect requirements	System Analyst	Low
3/22/19	3/27/19	IN3	4	Collect objectives	System Analyst	Low
9/16/19	10/18/19	PMX	25	Project Management		Low
9/16/19	9/23/19	PM1	6	Context and Scope	Technical Writer	Low
9/24/19	9/27/19	PM2	4	Time Planning	Technical Writer	Low
9/30/19	10/4/19	PM3	5	Budgets and Sustainability	Technical Writer	Low
10/7/19	10/18/19	PM4	10	Project Definition	Technical Writer	Low
3/28/19	10/23/19	SDX	146	Software Design		Low
3/28/19	4/3/19	SD1	5	System Architecture Design	Software Architect	Low
4/4/19	4/24/19	SD2	13	Ground Station Design	Software Architect	Low
4/25/19	4/29/19	SD3	3	Ground Station Database Design	Software Architect	Low
10/10/19	10/17/19	SD4	6	Operation Center Design	Software Architect	Low
10/18/19	10/23/19	SD5	4	Operation Center Database Design	Software Architect	Low
4/30/19	1/2/20	IMP	172	Software Implementation		Low
4/30/19	10/2/19	GSX	110	Ground Station Software		Low
4/30/19	6/26/19	GS3	41	Ground Station Hardware communication	Programmer	High
4/30/19	5/22/19	GS4	16	Ground Station Database	Programmer	Low
5/23/19	10/2/19	GS2	94	Ground Station Logic	Programmer	Low
6/24/19	9/27/19	GS1	69	Ground Station Interface	Programmer	Low
10/24/19	1/2/20	OCX	47	Operation Center Software		Low
10/24/19	1/2/20	OC1	47	Operation Center GUI	Programmer	Low
10/24/19	11/6/19	OC3	9	Operation Center Database	Programmer	Low
11/7/19	1/2/20	OC2	38	Operation Center Backend	Programmer	Low
4/30/19	1/20/20	TTX	183	Testing		Low
4/30/19	10/9/19	TT1	115	Ground Station Testing	Tester	Low
10/24/19	1/2/20	TT2	47	Operation Center Testing	Tester	Low
1/17/20	1/20/20	TT3	2	System Level Testing	Tester	Low
10/10/19	1/16/20	DPX	66	Deployment		Low
10/10/19	10/16/19	DP1	5	Ground Station Deployment	DevOps	High
1/3/20	1/9/20	DP2	4	Operation Center Deployment	DevOps	Medium
1/10/20	1/10/20	DP3	1	System Connection	DevOps	Medium
1/13/20	1/16/20	PV1	4	Project Verification	DevOps	Low
3/12/19	1/10/20	DC1	210	Writing documentation	Technical Writer	Low

Figure 4.2: GANTT diagram text part

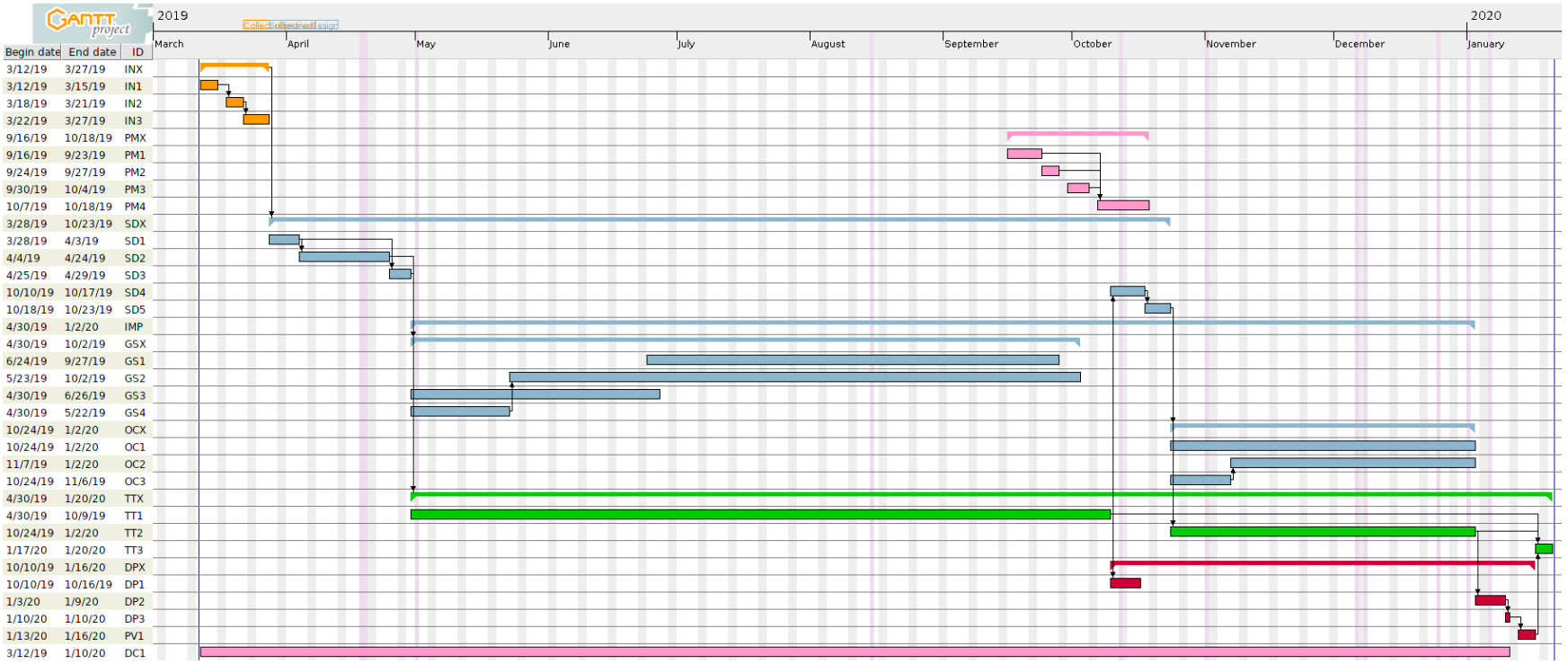


Figure 4.3: GANTT diagram

4.1.2.2 Problems and alternatives

The most critical part of the project is the GS software. It may be delayed because the hardware does not work or can not be tested when planned. If that happens, we can begin another iteration on another independent part of the software or start designing the OpCen. Additional tasks may take place, such as going to Montsec to repair the hardware or order new components and install them. This could delay the planning by one or two weeks.

Another risk is to underestimate the time needed to code some parts of the project. In that case, the project would be delayed by one or two weeks. Another option would be getting help from other programmers, meaning an increase of human resources.

4.1.2.3 Project feasibility

The estimated time for the project completion is 603 hours. Working 20h/week would mean needing 30 weeks to complete the project. As this project has a duration of one year and the GS software is almost complete, there should be enough time to finish it.

4.1.3 Budget

4.1.3.1 Identification of costs

For the correct development of the project, human and material resources are needed. The material resources are the hardware in which the software needs to run (a server for the OpCen and the already installed GS hardware) and the tools used to develop it, such as programs like code editors and hardware like a laptop. The facilities used are university installations, so they are not included in the scope of the project.

For the human resources, some roles where identified in order to carry each of the tasks in the full development of the project. Identifying this roles helps to make a more accurate break down of the expenses. All those roles are covered by me. Next the responsibilities of each role are described and related to the tasks of the project. The tasks are referenced by their identifier, but the full name of the task can be seen in Figure 4.4.

- **Technical Writer:** Is the person responsible of writing technical documents and documentation in general. This responsibilities can be translated in my project as the tasks PM1 - PM4 and DC1 as they are the ones that require documentation and technical knowledge of the project and how to manage it.
- **System Analyst:** Is the person who collects the requirements from the product owner and defines the objectives for the project. The System Analyst is also responsible of writing the verification criteria for the aforementioned requirements. It is responsible of all the tasks made during the inception phase (IN1-IN3) as they are the ones where the requirements are collected and the objectives fixed.

- **Software Architect:** The Software Architect is responsible of designing the system architecture in all levels of detail. In the project, its responsibilities are the design phase tasks (SD1-SD5), which require a system architecture design, software architecture design and database design.

- **Programmer:** Is the person which is responsible of implementing the system. In the project, all the implementing tasks are assigned to the programmer as they are the ones that require writing code (GS1-GS4, OC1-OC3).

- **Tester:** The tester has to check that the system is robust, behaves as it should, and follows the requirements determined in the inception phase. In the project there are three testing-related tasks, TT1-TT3 and will be executed by the tester role.

- **DevOps:** Is the responsible of the deployment of the system and has to make the connections between the subsystem work. Its associated tasks are the deployment ones (DP1-DP3) and PV1 which is the verification of the whole system, checking that all the tests and the deployment made is correct.

There are also some extra expenses like hardware amortisation (only the laptop, as the servers will be fully amortized) or trips to Montsec in order to install and test the software locally. Some incidental costs have also been taken into account, such as software implementation delay and extra trips to Montsec in case of hardware failures. In Figure 4.4, those costs are listed in more detail.

4.1.3.2 Cost estimates

The total cost of the project can be estimated with CPA, General Costs, Incidental Costs, Contingency and unforeseen costs. In order to calculate the CPA, each task is assigned to one of the roles mentioned above. For each task, the total cost is the cost of the assigned role per hour multiplied by the total of hours. In Table 4.3, the aforementioned salaries are shown. The price per hour is calculated taking into account a 1780 hours/year contract. The salaries are taken from GlassDoor[3], which provides the average salary of people working nearby as a specific role. The total cost for the human resources is 14866 euros. The total cost of the project is shown in detail in Figure 4.4.

For the General Costs calculation, the cost that are taken into account are the costs of the software used (no cost, because they are all free to use), and the trips to Montsec, which are calculated as the gasoline needed supposing an average consumption of 8 liters per 100km and food for 4 people, with the average expenses of 35 euros each. Three trips to Montsec are estimated, one trip to install hardware and two more trips to test the software. Finally, the cost of a server to host the OpCen Software and the usage of the laptop with an average useful life of 4 years are added. The server cost will be fully amortised as it will be used for the next four years. The amortisation A_L of the

[3]Glassdoor: https://www.glassdoor.es

laptop is calculated considering an average useful life of four years, 1780 working hours per year, the cost of the laptop C_L, and the total project hours T_P using the formulae in the Eq. (4.1)

$$A_L = \frac{C_L \cdot T_P}{1780 \cdot 4} \tag{4.1}$$

All the other hardware was already there before the project, so it is already included in previous projects budget. The general costs are 887 euros.

With both CPA and CG calculated, the total cost ascends to 15753 euros. To this cost, a 15% of contingency margin is added, as it is the average margin used in software projects. With this margin, the total cost ascends to 18116 euros. Next, the incidental costs of the project are included.

In the incidental costs, there are two main factors of risk: software implementation delay and hardware failure. For the software implementation delay, two possible delays are estimated. The first one is a one-week delay and it has a bigger risk (30%) whereas the second is an extra week delay and has a lower risk (15%) as it is more probable that if it has been delayed once, there has been enough time to finish it and there is no need of another week of delay. For the hardware failure incidental, it is only taken into account the cost of going to Montsec and repair or replace the hardware as it is assumed that the components have warranty. The total incidental costs ascend to 97 euros. The final cost of the project is 18212 euros.

Table 4.3: Annual Salary of different project roles

Role	Annual Salary (euros)	Total including SS (euros)	Price hour	Total hours project	Role total cost
Technical Writer	26350.00[24]	35571.50	19.98	115	2297.70
DevOps	36347.00[25]	49068.45	27.56	70	1929.20
Programmer	23961.00[26]	32347.35	18.17	214	3888.38
System Analyst	28299.00[27]	38203.65	21.46	25	536.50
Tester	32236.00[28]	43518.60	24.45	120	2934.00
Software Architect	46240.00[29]	62424.00	25.98	39	1013.22

Activity	Import (€)	Comments
PM1 - Context and Scope	413.50	Technical writer, 15 hours
PM2 - Time Planning	413.50	Technical writer, 15 hours
PM3 - Budgets and Sustainability	413.50	Technical writer, 15 hours
PM4 - Project Definition	275.67	Technical writer, 10 hours
PM5 - Weekly Meetings	1494.10	Programmer, Software Architect, System Analyst, 20 hours
IN1 - Define objectives	107.31	System Analyst, 5 hours
IN2 - Define requirements	321.94	System Analyst, 15 hours
IN3 – Requirements verification methods	107.31	System Analyst, 5 hours
SD1 - System Architecture Design	175.35	Software Architect, 5 hours
SD2 - Ground Station Design	701.39	Software Architect, 20 hours
SD3 - Ground Station Database Design	70.14	Software Architect, 2 hours
SD4 - Operation Center Design	350.70	Software Architect, 10 hours
SD5 - Operation Center Database Design	70.14	Software Architect, 2 hours
GS1 - Ground Station Interface	181.73	Programmer, 10 hours
GS2 - Ground Station Logic	908.63	Programmer, 50 hours
GS3 - Ground Station Hardware comm	908.63	Programmer, 50 hours
GS4 - Ground Station Database	36.35	Programmer, 2 hours
OC1 - Operation Center GUI	908.63	Programmer, 50 hours
OC2 - Operation Center Backend	908.63	Programmer, 50 hours
OC3 - Operation Center Database	36.35	Programmer, 2 hours
TT1 - Ground Station Testing	1222.43	Tester, 50 hours
TT2 - Operation Center Testing	1222.43	Tester, 50 hours
TT3 - System Level Testing	488.97	Tester, 20 hours
DP1 - Ground Station Deployment	551.33	DevOps, 20 hours
DP2 - Operation Center Deployment	551.33	DevOps, 20 hours
DP3 - System Connection	275.67	DevOps, 10 hours
PV1 - Project Verification	551.33	DevOps, 20 hours
DC1 - Writing documentation	1199.07	Technical writer, 60 hours
Total CPA	**14866.07**	
Laptop	117.97	Thinkpad t580, 700 hours
Draw.io software	0.00	Free to use
Overleaf software	0.00	Free to use
LibreOffice Calc software	0.00	Free to use
Visual Studio Code	0.00	Free to use
GitLab	0.00	Free to use
3 x Trips to Montsec	539.88	370km gasoline (8 liters/100 km) + food (4 people, 35€ each)
Operation Center Server	229.00	HP ProLiant MicroServer G8 Intel G1610T/4GB
Total CG	**886.85**	
Total Costes (Total CPA + Total CG)	**15752.92**	
Contingency	2362.94	Contingency margin = 15%
Total CD+CI +Contingency	**18115.85**	
Software implementation delay (1 week)	40.38	Cost: Programmer, 20 hours. Risk = 30%
Software implementation delay (2 weeks)	20.19	Cost: Programmer, 20 hours. Risk = 15%
Harware failure, trip to Montsec to repair it	35.99	Cost: Trip to Montsec. Risk = 20%
Total incidentals:	**96.57**	
TOTAL:	**18212.42**	

Figure 4.4: Break down of expenses

4.1.4 Management control

In order to check the deviations of the budget regarding human resources, the time needed to actually complete each task will be annotated. With the cost per hour of the worker (C_H), the estimated hours (H_E) and the total hours (H_T) we will know the deviation of costs in human resources (D_{HR}) in efficiency for each task described in the GANTT diagram. The aforementioned deviation can be calculated following the Eq. (4.2):

$$D_{HR} = C_H \cdot (H_E - H_T) \tag{4.2}$$

For the general costs, we have three types of deviation, the deviation in amortisation, the number of trips to Montsec and the cost of the servers.

The deviation in the amortisation of the laptop (D_{LA}) is calculated in Eq. (4.3), taking into account the total usage hours (H_T), the estimated usage hours (H_E) and the cost per hour (C_H) calculated in the Eq. (4.1):

$$D_{LA} = C_H \cdot (H_E - H_T) \tag{4.3}$$

The trips to Montsec can be counted in order to determine the total trips (NT_T) versus the estimated trips (NT_E) and check if there were more than predicted in order to calculate the travel expenses deviation (D_{TE}) by multiplying the difference for the cost of one trip (C_T), as seen in Eq. (4.4).

$$D_{TE} = C_T \cdot (NT_E - NT_T) \tag{4.4}$$

The deviation on the cost of the server (D_{CS}) is defined in the Eq. (4.5) as the subtraction of the real cost (C_{SR}) with the expected one (C_{SE}):

$$D_{CS} = C_{SE} - C_{SR} \tag{4.5}$$

To sum up, a the total costs deviation (D_C) is calculated in the Eq. (4.6) as the total of estimated costs (C_E) versus the total of the real ones (C_R).

$$D_C = C_E - C_R \tag{4.6}$$

Finally, for the incidental costs, there will be a list of incidental costs that will be updated every time some unexpected expense occurs. By doing so, it will be easy at the end to compare the expected incidental costs with the real incidental expenses. The value of this deviation can be seen in the number of trips to Montsec deviation and on the human resources efficiency deviation.

With these indicators, we will know where the deviations in the budget are. To get an overview of the project budget we can compare if the total cost of the project is larger than the total estimated cost and see if the contingency margin was estimated correctly.

4.2 Final Project Planning and Budget

4.2.1 New Tasks

The initial planning had to be changed because the implementation tasks took much longer than the originally estimated. The GS Software took about one hundred more hours to be completed at operational level.

After doing the OpCen design, the functionalities were divided in four different profiles, one for each of the roles an operator can cover and an extra one whose main purpose is to offer information to the rest of the personnel. It was estimated that from the four requested user profiles, the visualization and the management ones could be implemented to an operational level on time, while the other ones would be future work.

The chosen profiles were the OpCen and GS Management, and the Visualization profile, as they are critical in order to have an operational version of the OpCen. By doing so, six new design, implementation and test tasks are defined. Task SD4 is also redefined to make it more general. The new tasks can be defined as follows:

- **SD4 - OpCen Design:** To design the backend server architecture and the different user interfaces functionalities and look and feel.
- **SD6 - OpCen Management Software Design:** To design in a detailed manner the Management Software and User Interface, using software architecture patterns, REST Application Programming Interface (API) protocols and database synchronization libraries.
- **SD7 - OpCen Visualization Software Design:** To design in a detailed manner the Visualization Software, User Interface and use of external services and libraries.
- **OC1 - Management Software Implementation:** Implementation of the WebApp and backend API for the Management Software
- **OC2 - Visualization Software Implementation:** Implementation of the WebApp and backend API for the Visualization Software and integration with external libraries and APIs.
- **TT2 - Management Software Testing:** To test the Management Software API and User Interface.
- **TT3 - Visualization Software Testing:** To test the Visualization Software API and User Interface.
- **TT4 - UI/UX Testing:** To test the user interface of the management and the visualization softwares. It requires testers and both softwares to be completed and running.

The updated tasks are shown in Table 4.4. As a result of the changes made, the total number of hours dedicated to the project has risen to 963. This results in an extension of the schedule and an increase in costs, as it will be seen below.

Table 4.4: Summary of the final tasks to be performed

ID	Name	Time(h)	Predecessors	Resources
	Project Management			
PM1	Context and Scope	15		PC, *draw.io*, *Overleaf*
PM2	Time Planning	15		PC, GANTTproject, *Overleaf*, calc
PM3	Budgets and Sustainability	15		PC, *Overleaf*, calc
PM4	Project Definition	10	PM1, PM2, PM3	PC
PM5	Weekly Meetings	40		Work Reports
	Inception			
IN1	Define objectives	5		PC
IN2	Define requirements	15	IN1	PC
IN3	Establish requirements verification methods	5	IN1	PC
	Software Design			
SD1	System Architecture Design	5	IN1, IN2, IN3	PC, *draw.io*
SD2	GS Design	20	SD1	PC, *draw.io*
SD3	GS Database Design	2	SD2	PC, *draw.io*
SD4	OpCen Design	15	SD1	PC, *draw.io*
SD5	OpCen Database Design	2	SD4	PC, *draw.io*
SD6	OpCen Management Software Design	20	SD4	PC, *visual studio*
SD7	OpCen Visualization Software Design	5	SD4	PC, *visual studio*
	Software Implementation			
	GS Software			
GS1	GS Interface	50	SD3	PC, visual studio code, design diagrams, git
GS2	GS Logic	80	SD3	PC, visual studio code, design diagrams, git
GS3	GS Hardware communication	80	SD2	PC, visual studio code, design diagrams, git
GS4	GS Database	2	SD3	PC, visual studio code, design diagrams, git
	OpCen Software			
OC1	Management Software Implementation	200	SD5	PC, visual studio code, design diagrams, git
OC2	Visualization Software Implementation	75	SD5	PC, visual studio code, design diagrams, git
OC3	OpCen Database	2	SD5	PC, visual studio code, design diagrams, git
	Testing			
TT1	GS Testing	50	GS1, GS2, GS3, GS4 *	PC, visual studio code, git, hardware access
	OpCen Testing			
TT2	Management Software Testing	30	OC1, OC2	PC, visual studio code, git, postman
TT3	Visualitzation Software Testing	20	OC1, OC3	PC, visual studio code, git, postman
TT4	UI/UX Testing	15	OC1, OC2, OC3	Software running, Test plan
TT5	System Level Testing	20	DP3	Access to both softwares
	Deployment & Conclusion			
DP1	GS Deployment	20	TT1	PC, git, GS remote access
DP2	OpCen Deployment	20	TT2	PC, git, OpCen server access
DP3	System Connection	30	DP1, DP2	Access to both softwares
PV1	Project Verification	20	DP1, DP2	OpCen access
DC1	Writing documentation	60	ALL **	PC, *Overleaf*, *draw.io*, project resources
	Total:	**963**		

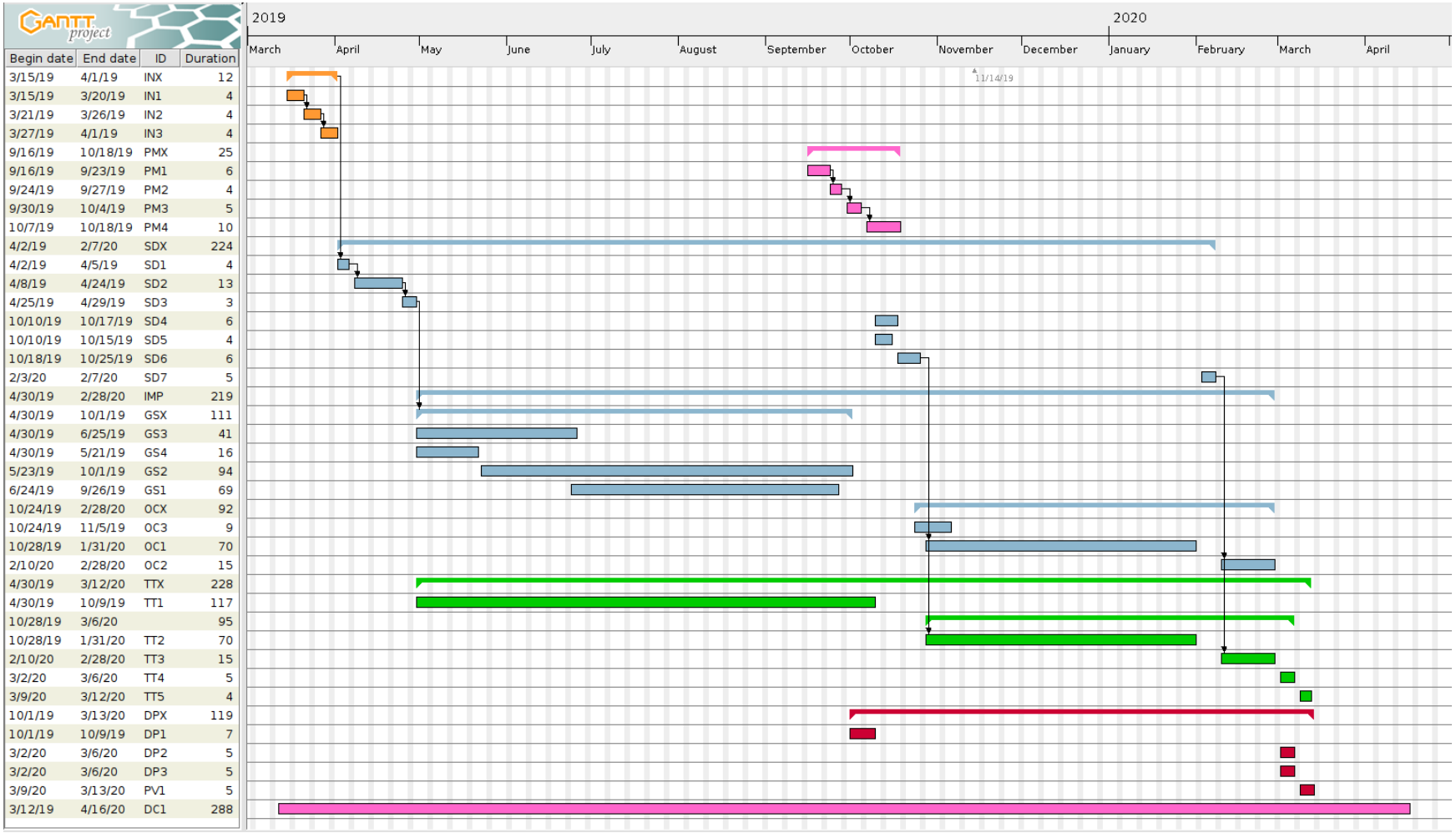
4.2.2 Final Schedule

As seen in the *New Tasks* section, the GS implementation time increased by 100 hours, which in the end, did not cause any delay in the schedule, but the OpCen implementation and testing did, as it has in total, 175 more hours than expected.

The implementation and testing tasks were extended for three weeks and the deployment and verification ones, were moved five weeks ahead. For this reason, the schedule has been extended from January 16th to April 16th. In Figures 4.5, 4.6 the updated schedule is shown in more detail.

Name	Begin date	End date	Coordinator	Risk	ID	Duration
Establish requirements verification methods	3/15/19	4/1/19			INX	12
Inception	3/15/19	3/20/19	System Analyst	Low	IN1	4
Collect requirements	3/21/19	3/26/19	System Analyst	Low	IN2	4
Collect objectives	3/27/19	4/1/19	System Analyst	Low	IN3	4
Project Management	9/16/19	10/18/19			PMX	25
Context and Scope	9/16/19	9/23/19	Technical Writer	Low	PM1	6
Time Planning	9/24/19	9/27/19	Technical Writer	Low	PM2	4
Budgets and Sustainability	9/30/19	10/4/19	Technical Writer	Low	PM3	5
Project Definition	10/7/19	10/18/19	Technical Writer	Low	PM4	10
Software Design	4/2/19	2/7/20			SDX	224
System Architecture Design	4/2/19	4/5/19	Software Architect	Low	SD1	4
Ground Station Design	4/8/19	4/24/19	Software Architect	Low	SD2	13
Ground Station Database Design	4/25/19	4/29/19	Software Architect	Low	SD3	3
Operation Center Design (General)	10/10/19	10/17/19	Software Architect	Low	SD4	6
Operation Center Database Design	10/10/19	10/15/19	Software Architect	Low	SD5	4
OpCen Management Software Design	10/18/19	10/25/19	Software Architect	Low	SD6	6
OpCen Visualitzation Software Design	2/3/20	2/7/20	Software Architect	Low	SD7	5
Software Implementation	4/30/19	2/28/20			IMP	219
Ground Station Software	4/30/19	10/1/19			GSX	111
Ground Station Hardware communication	4/30/19	6/25/19	Programmer	High	GS3	41
Ground Station Database	4/30/19	5/21/19	Programmer	Low	GS4	16
Ground Station Logic	5/23/19	10/1/19	Programmer	Low	GS2	94
Ground Station Interface	6/24/19	9/26/19	Programmer	Low	GS1	69
Operation Center Software	10/24/19	2/28/20			OCX	92
Operation Center Database	10/24/19	11/5/19	Programmer	Low	OC3	9
Management Software Implementation	10/28/19	1/31/20	Programmer	Low	OC1	70
Visualitzation Software Implementation	2/10/20	2/28/20	Programmer		OC2	15
Testing	4/30/19	3/12/20			TTX	228
Ground Station Testing	4/30/19	10/9/19	Tester	Low	TT1	117
Operation Center Testing	10/28/19	3/6/20	Tester	Low		95
Management Software Testing	10/28/19	1/31/20	Tester	Low	TT2	70
Visualitzation Software Testing	2/10/20	2/28/20	Tester	Low	TT3	15
UI/UX Testing	3/2/20	3/6/20	Tester	Low	TT4	5
System Level Testing	3/9/20	3/12/20	Tester	Low	TT5	4
Deployment	10/1/19	3/13/20			DPX	119
Ground Station Deployment	10/1/19	10/9/19	DevOps	High	DP1	7
Operation Center Deployment	3/2/20	3/6/20	DevOps	Medium	DP2	5
System Connection	3/2/20	3/6/20	DevOps	Medium	DP3	5
Project Verification	3/9/20	3/13/20	DevOps	Low	PV1	5
Writing Documentation	3/12/19	4/16/20	Technical Writer	Low	DC1	288

Figure 4.5: Final GANTT tasks list



Begin date	End date	ID	Duration
3/15/19	4/1/19	INX	12
3/15/19	3/20/19	IN1	4
3/21/19	3/26/19	IN2	4
3/27/19	4/1/19	IN3	4
9/16/19	10/18/19	PMX	25
9/16/19	9/23/19	PM1	6
9/24/19	9/27/19	PM2	4
9/30/19	10/4/19	PM3	5
10/7/19	10/18/19	PM4	10
4/2/19	2/7/20	SDX	224
4/2/19	4/5/19	SD1	4
4/8/19	4/24/19	SD2	13
4/25/19	4/29/19	SD3	3
10/10/19	10/17/19	SD4	6
10/10/19	10/15/19	SD5	4
10/18/19	10/25/19	SD6	6
2/3/20	2/7/20	SD7	5
4/30/19	2/28/20	IMP	219
4/30/19	10/1/19	GSX	111
4/30/19	6/25/19	GS3	41
4/30/19	5/21/19	GS4	16
5/23/19	10/1/19	GS2	94
6/24/19	9/26/19	GS1	69
10/24/19	2/28/20	OCX	92
10/24/19	11/5/19	OC3	9
10/28/19	1/31/20	OC1	70
2/10/20	2/28/20	OC2	15
4/30/19	3/12/20	TTX	228
4/30/19	10/9/19	TT1	117
10/28/19	3/6/20		95
10/28/19	1/31/20	TT2	70
2/10/20	2/28/20	TT3	15
3/2/20	3/6/20	TT4	5
3/9/20	3/12/20	TT5	4
10/1/19	3/13/20	DPX	119
10/1/19	10/9/19	DP1	7
3/2/20	3/6/20	DP2	5
3/2/20	3/6/20	DP3	5
3/9/20	3/13/20	PV1	5
3/12/19	4/16/20	DC1	288

Figure 4.6: New GANTT diagram

4.2.3 Effect on project cost

As a result of the increase in total hours required for the execution of the project, the total cost of this has been increased. This increase is due solely to human resources costs, since the hardware and software used remains the same. This can be easily seen in Figure 4.7 where the updated budget is shown in detail.

Activity	Import (€)	Comments
PM1 - Context and Scope	413,50	Technical writer, 15 hours
PM2 - Time Planning	413,50	Technical writer, 15 hours
PM3 - Budgets and Sustainability	413,50	Technical writer, 15 hours
PM4 - Project Definition	275,67	Technical writer, 10 hours
PM5 - Weekly Meetings	1494,10	Programmer, Software Architect, System Analyst, 20 hours
IN1 - Define objectives	107,31	System Analyst, 5 hours
IN2 - Define requirements	321,94	System Analyst, 15 hours
IN3 – Requirements verification methods	107,31	System Analyst, 5 hours
SD1 - System Architecture Design	175,35	Software Architect, 5 hours
SD2 - Ground Station Design	701,39	Software Architect, 20 hours
SD3 - Ground Station Database Design	70,14	Software Architect, 2 hours
SD4 - Operation Center Design	526,04	Software Architect, 15 hours
SD5 - Operation Center Database Design	70,14	Software Architect, 2 hours
SD6 - OpCen Management Software Design	701,39	Software Architect, 20 hours
SD7 – OpCen Visualization Software Design	175,35	Software Architect, 5 hours
GS1 - Ground Station Interface	908,63	Programmer, 50 hours
GS2 - Ground Station Logic	1453,81	Programmer, 80 hours
GS3 - Ground Station Hardware comm	1453,81	Programmer, 80 hours
GS4 - Ground Station Database	36,35	Programmer, 2 hours
OC1 - Management Software Implementation	3634,53	Programmer, 200 hours
OC2 – Visualisation Software Implementation	3634,53	Programmer, 200 hours
OC3 - Operation Center Database	1362,95	Programmer, 75 hours
TT1 - Ground Station Testing	1222,43	Tester, 50 hours
TT2 - Management Software Testing	733,46	Tester, 30 hours
TT3 – Visualization Software Testing	488,97	Tester, 20 hours
TT4 – UI/UX Testing	366,73	Tester, 15 hours
TT5 - System Level Testing	488,97	Tester, 20 hours
DP1 - Ground Station Deployment	551,33	DevOps, 20 hours
DP2 - Operation Center Deployment	551,33	DevOps, 20 hours
DP3 - System Connection	275,67	DevOps, 10 hours
PV1 - Project Verification	551,33	DevOps, 20 hours
DC1 - Writing documentation	1199,07	Technical writer, 60 hours
Total CPA	24880,56	
Laptop	151,69	Thinkpad t580, 900 hours
Draw.io software	0,00	Free to use
Overleaf software	0,00	Free to use
LibreOffice Calc software	0,00	Free to use
Visual Studio Code	0,00	Free to use
GitLab	0,00	Free to use
3 x Trips to Montsec	539,88	370km gasoline (8 liters/100 km) + food (4 people, 35€ each)
Operation Center Server	229,00	HP ProLiant MicroServer G8 Intel G1610T/4GB
Total CG	**920,57**	
Total Costes (Total CPA + Total CG)	**25801,12**	
Contingency	3870,17	Contingency margin = 15%
Total CD+CI +Contingency	**29671,29**	
Software implementation delay (1 week)	40,38	Cost: Programmer, 20 hours. Risk = 30%
Software implementation delay (2 weeks)	20,19	Cost: Programmer, 20 hours. Risk = 15%
Harware failure, trip to Montsec to repair it	35,99	Cost: Trip to Montsec. Risk = 20%
Total incidentals:	**96,57**	
TOTAL:	**29767,86**	

Figure 4.7: Updated budget

Comparing it with the old one, the final effect on project costs has an increase of the total cost of 8000 euros. Such an increase was not contemplated in the incidental costs. The cause of this difference is an underestimation of the time that some tasks required. In order to improve it, in future projects, the defined tasks should be more granular because it is easier to estimate the time a small task takes than a big one.

4.2.4 Project feasibility and objectives

Due to the quantity of work required for the completion of the project, the objectives had to be prioritized. As seen in the requirements table, each requirement had a level of priority, where *high priority* meant that the requirement had to be completed in order to have an operational version, and *medium* and *low* were requirements that needed to be completed at a design level but could be implemented in the future.

For that reason, some of the requirements are going to be future work, which means that not all the objectives mentioned will be completed. Some of the objectives affected by the schedule are the Uplink communication with the satellites or allowing external companies to use our network of GSs. The second one will also be future work as there has not been any agreement yet on how this process should be handled.

The GS and OpCen objectives are all on plan except the communication with the satellite, as it is not bidirectional at the moment because the software is not ready yet. For an operational version, it is enough to be able to receive data although it cannot be sent, so this objective is not very critical if it is not on plan.

External companies will not be able to access scheduling functionalities by now, as it is not scheduled to be implemented yet, but the design is flexible enough to incorporate this functionality in the future. Furthermore, we can still schedule passes for them using some other kind of communication, such as an email or phone call, and sending them the recorded file afterwards.

A summary of the objectives status can be seen in Table 4.5, where they are all listed along with their current completion status.

Finally there are some requirements that are not met, such as health tests implementation for the GSs or having a data visualisation software, but it can be done through other ways, such as testing the antennas manually or using external softwares respectively. During the Project Verification(PV1) we will analyse the requirements validation in more detail.

Table 4.5: Objectives completion review

Objective	Completion
OB 1: Design a software that guarantees that all the roles of the operators can be covered	On plan
OB 1.1: The GS manager can check the well-functioning of the GSs	On plan
OB 1.2: The GS manager can schedule passes	On plan
OB 1.3: The downlink manager will be able of receiving data in real time	On plan
OB 1.4: The uplink manager will be able to plan the uplink commands	On plan
OB 1.5: The uplink manager will be able to send commands to the satellite	On plan
OB 1.6: All the personnel is kept informed during the operations	On plan
OB 2: Control GSs from one single location in order to operate satellites	On plan
OB 2.1: An OpCen connected to a number of GSs	On plan
OB 2.2: GSs that can communicate with satellites	Partially on plan (downlink only)
OB 2.3: Build an infrastructure and its required links to become capable of sharing data between the GSs and the OpCen, with the latter acting as a central hub.	On plan
OB 3: Schedule satellite passes automatically	On plan
OB 3.1: Schedule passes from the OpCen.	On plan
OB 3.2: Having a GS software which is capable of executing the pass automatically.	On plan
OB 4: Having the data centralised in the OpCen	On plan
OB 4.1: Find a method of synchronising the data received in the GSs with the OperationCenter.	On plan
OB 5: Provide the capability for the operator to enable external companies to have access to scheduling functionalities.	Future Work
OB 6: Communicate with the existing GS Hardware.	On plan
OB 6.1: Create an interface to the existing UHF / VHF chain with the new software structure.	On plan
OB 6.2: Create a brand new interface to the new S-Band chain.	On plan

5 | Software architecture

The system design has been focused on creating a modular system architecture which is easily implantable and extensible to different GSs. With this approach, the interfaces to the GSs are unified, and the OpCen can control them in a secure and controlled manner. The system consists of two main software blocks, the OpCen and the GS software, each of them located in separate locations. These two softwares are connected through a connection block, which includes the technologies needed to communicate them. An overview of the system diagram is shown in Figure 5.1.

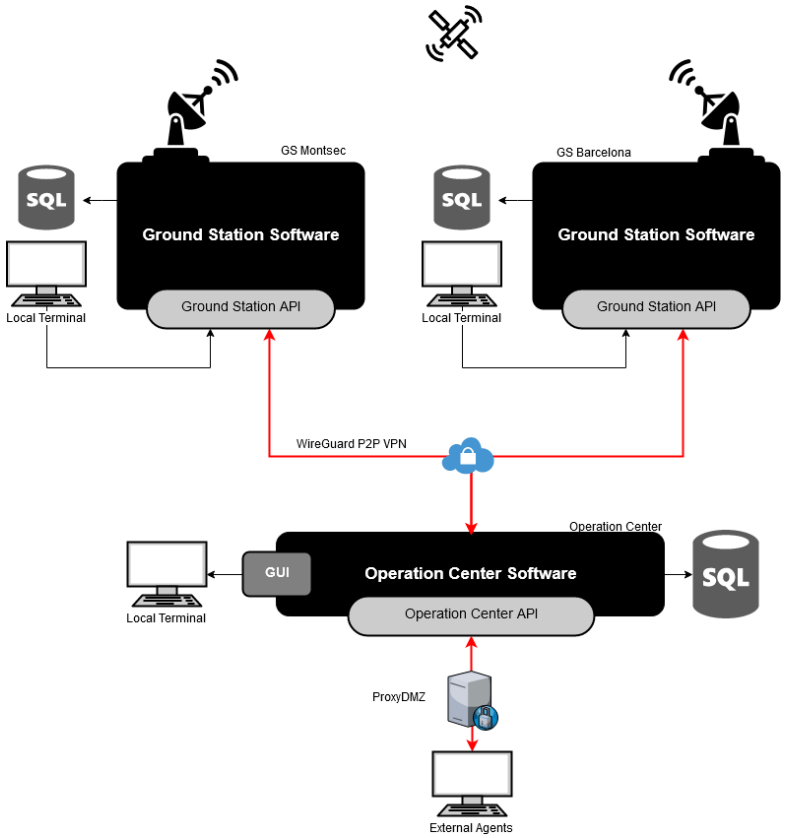


Figure 5.1: System design overview

The GS software can be installed in many GSs in order to operate them. It has

a common API so that all the GSs can be accessed from the OpCen using the same interface. It also has access to a Database which will share the same data schema amongst GSs. This Database has information about the working schedule of that GS and the data retrieved from satellite passes. The OpCen software consists of a user interface that will allow the operator to perform all the functionalities described in the objectives, a database with information of the ground stations and received data, and access to the GSs' API. It also has an external API so that external users can request the service of the GSs. There is also the connection block. This block includes a secure connection (based on IP security (IPSec) protocol) between the OpCen and the GSs, as well as a controlled access to the external API. It also provides data synchronization methods between the GSs and the OpCen, including files and Database synchronization.

6 | Ground Station Design

6.1 Ground Station overview

As mentioned before, the GS has to act in two different scenarios, each of them corresponding to a different **entry point** of the software.

- **Operator request:** The GS can be accessed by the operator locally or using the OpCen software through the same API.
- **Scheduled task:** The GS has to handle those requests automatically by calling the necessary methods at the required time (usually moving antennas at the requested time).

Any of those entry points trigger different functionalities that are managed by the **controllers**, which have all the logic of the system. The controllers have access to the resources of the GS. They can access the database and the adapters, which are the direct connection with the GS hardware. The access to the adapters is given by a software multiplexer, which manages concurrent calls to the hardware.As the hardware is specific of each GS's chain, the **adapters** have to be implemented specifically for each chain, but all of them will share a common interface to make it compatible with the controllers. A schema of the GS architecture overview is shown in Figure 6.1.

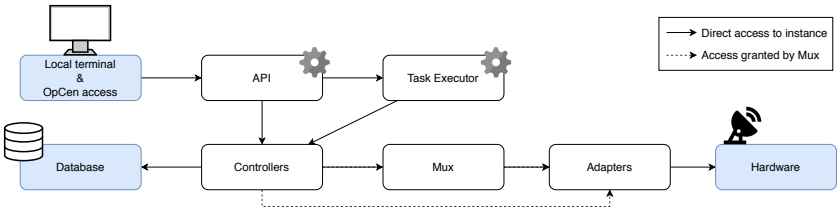
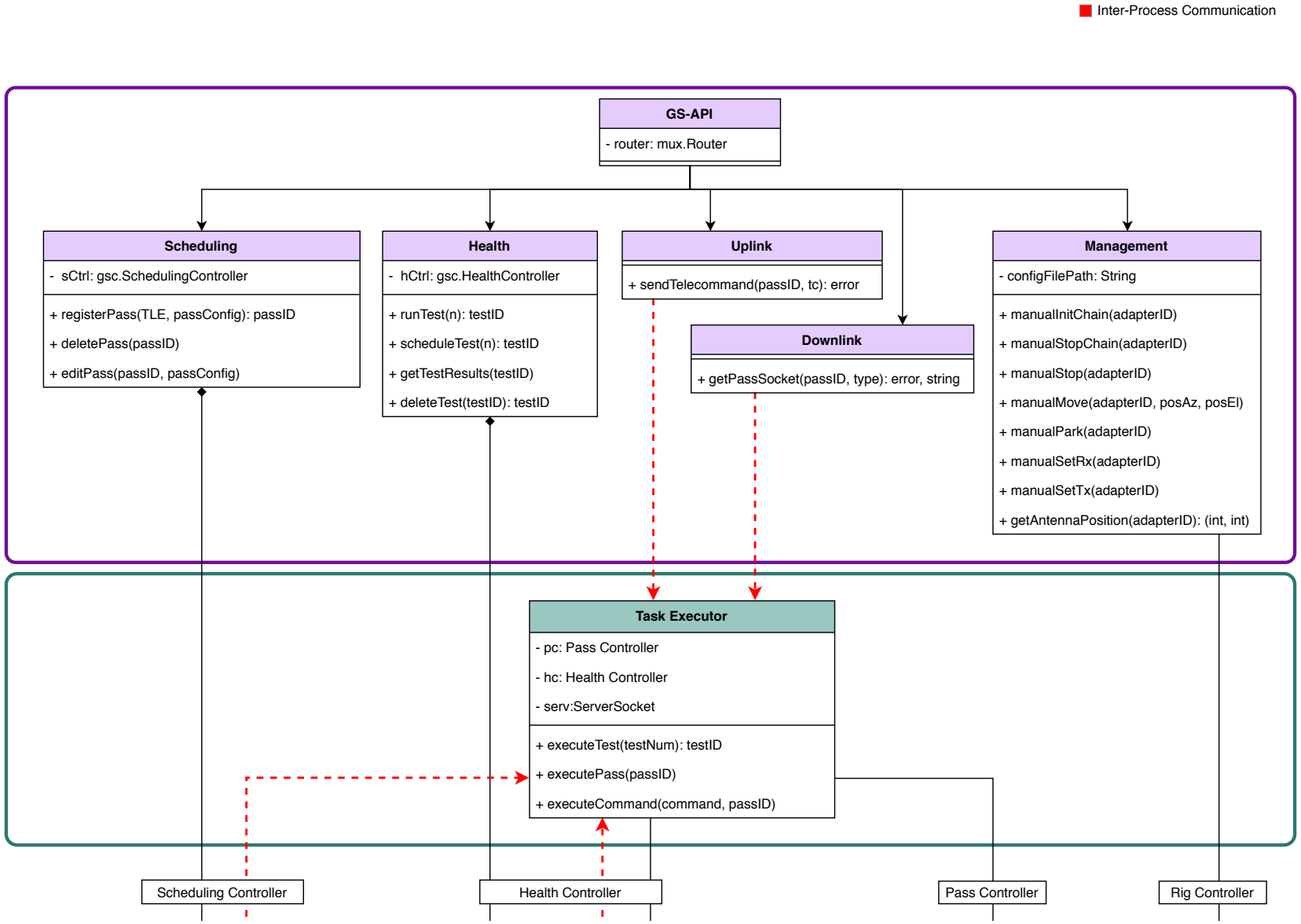
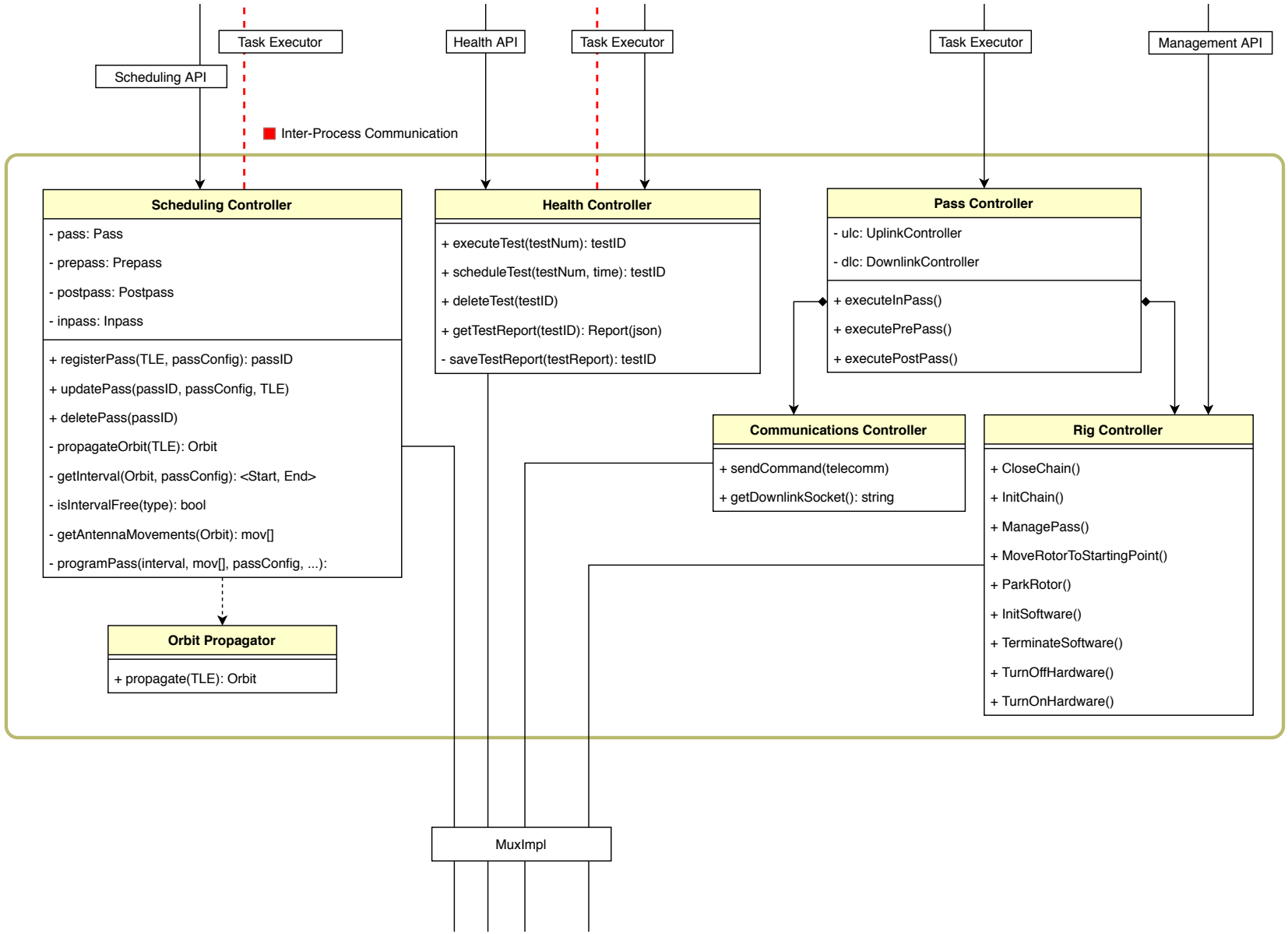


Figure 6.1: GS software overview

6.2 Detailed design

In the following sections, each of the aforementioned components are explained in detail. Furthermore, the UML specification of each component is shown in detail in Figure 6.2.


Inter-Process Communication
GS-API
- router: mux.Router
Scheduling
- sCtrl: gsc.SchedulingController
+ registerPass(TLE, passConfig): passID
+ deletePass(passID)
+ editPass(passID, passConfig)
Health
- hCtrl: gsc.HealthController
+ runTest(n): testID
+ scheduleTest(n): testID
+ getTestResults(testID)
+ deleteTest(testID): testID
Uplink
+ sendTelecommand(passID, tc): error
Downlink
+ getPassSocket(passID, type): error, string
Management
- configFilePath: String
+ manualInitChain(adapterID)
+ manualStopChain(adapterID)
+ manualStop(adapterID)
+ manualMove(adapterID, posAz, posEl)
+ manualPark(adapterID)
+ manualSetRx(adapterID)
+ manualSetTx(adapterID)
+ getAntennaPosition(adapterID): (int, int)
Task Executor
- pc: Pass Controller
- hc: Health Controller
- serv:ServerSocket
+ executeTest(testNum): testID
+ executePass(passID)
+ executeCommand(command, passID)
Scheduling Controller
Health Controller
Pass Controller
Rig Controller


Scheduling API
Task Executor
Health API
Task Executor
Task Executor
Management API
Inter-Process Communication
Scheduling Controller
- pass: Pass
- prepass: Prepass
- postpass: Postpass
- inpass: Inpass
+ registerPass(TLE, passConfig): passID
+ updatePass(passID, passConfig, TLE)
+ deletePass(passID)
- propagateOrbit(TLE): Orbit
- getInterval(Orbit, passConfig): <Start, End>
- isIntervalFree(type): bool
- getAntennaMovements(Orbit): mov[]
- programPass(interval, mov[], passConfig, ...):
Health Controller
+ executeTest(testNum): testID
+ scheduleTest(testNum, time): testID
+ deleteTest(testID)
+ getTestReport(testID): Report(json)
- saveTestReport(testReport): testID
Pass Controller
- ulc: UplinkController
- dlc: DownlinkController
+ executeInPass()
+ executePrePass()
+ executePostPass()
Communications Controller
+ sendCommand(telecomm)
+ getDownlinkSocket(): string
Rig Controller
+ CloseChain()
+ InitChain()
+ ManagePass()
+ MoveRotorToStartingPoint()
+ ParkRotor()
+ InitSoftware()
+ TerminateSoftware()
+ TurnOffHardware()
+ TurnOnHardware()
Orbit Propagator
+ propagate(TLE): Orbit
MuxImpl

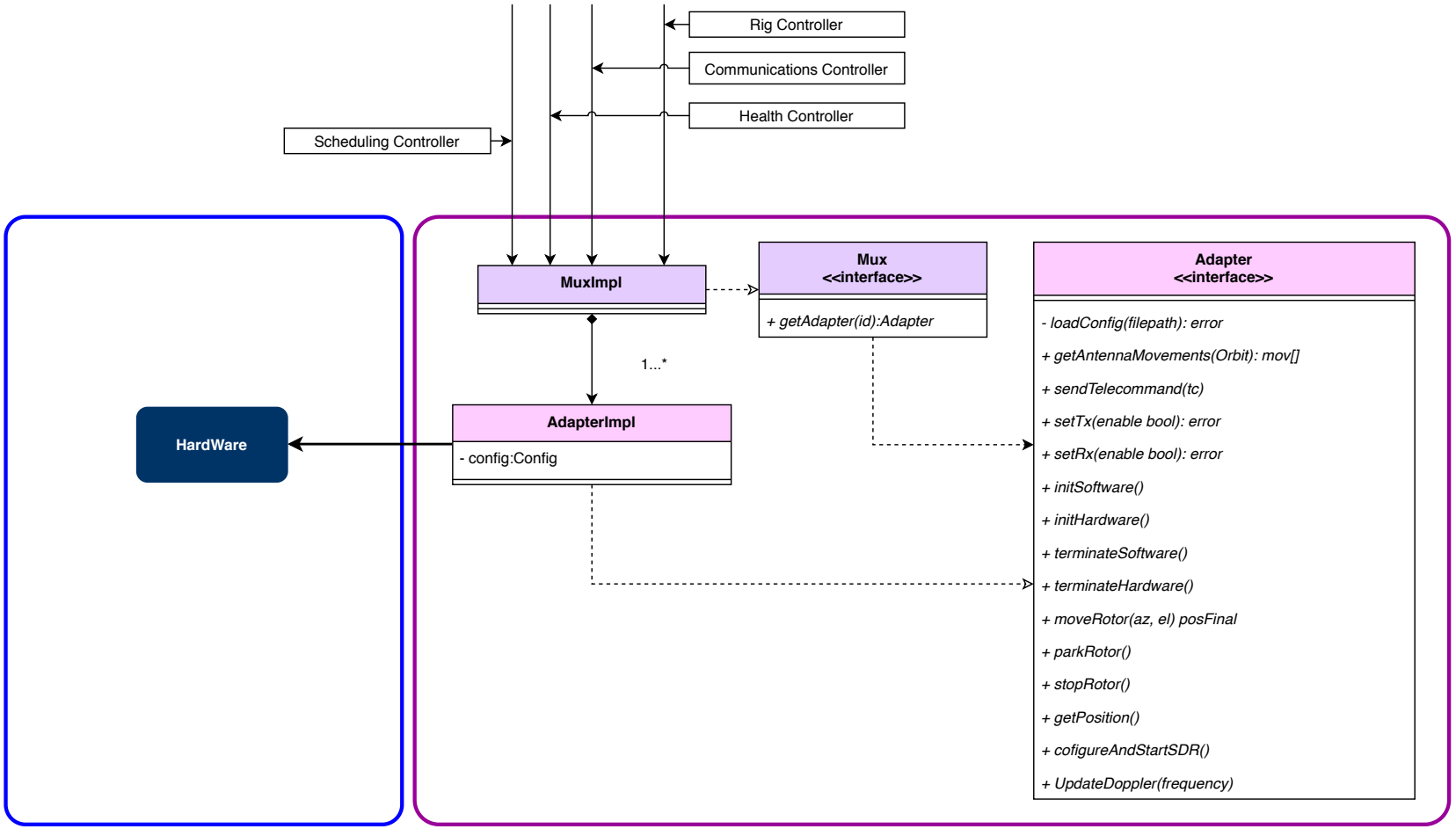


Figure 6.2: UML Diagram of the GS Software

6.2.1 Communication with the Hardware

As mentioned in the previous section, the GS software communicates with the chain hardware using specific **Adapters**. This adapters are a hardware-specific implementation of the *Adapter Interface*, which defines all the operations that the hardware should be capable of performing. Those operations are divided in some groups according to the involved components:

- **Init/Stop chain:** This group of functions include turning on/off the hardware and executing any external script or program that is required in order to be able to use the chain.
- **Antenna movement:** Functions regarding the movement of the antenna, including not only physical movement but also moving it to its parking position or knowing its movement range and limitations. The parking position is the operator-configured position in which the antenna rests and its purpose is to avoid any damage that the bad weather could do to the antenna, such as snow accumulating on the dish.
- **Transceivers control:** Functions related to changing the state of the communication (Transmission or Reception or both) via hardware.
- **SDR control:** It includes the functions for starting and stopping the required GNURadio blocks in order to transform the received signal to raw data and to transform commands to radio signal. Furthermore, it is responsible of creating files where the downloaded data can be stored, so the OpCen can copy and access them at any time.

As there is physically one hardware chain per implementation, there can only be one instance of the adapter accessing the real hardware. For that reason, the adapter class must be a *Singleton*, which is a software design pattern that restricts the instantiating of that class to one single instance.

6.2.2 Managing Adapters access

Even if the adapters are singleton classes, their instance could be retrieved and used by many other controllers instances at the same time, which may compete to get the same resources. For that reason, the *Multiplexer interface* is required. The *Mux interface* provides methods of managing the access to the adapters, as its instance is only given through them. This interface has to specifically be implemented for each GS as it has to instantiate all the available adapters.

The **Mux** provides control on whether the instance of an adapter should be given to a controller or not. This mechanism helps to control concurrent accesses to an instance that may not always accept it.

6.2.3 The Controllers logic

The controllers encapsulate the logic of the software. The main idea is to separate them by functionality. There are controllers specialized in scheduling and executing tasks, accessing data and interacting with the hardware.

The **Scheduling Controller** contains all the logic to schedule, edit and delete passes. It knows how to use the *Orbit Propagator* in order to determine when there is a pass and knows how to correct the Doppler frequencies. It also has access to the *Adapters* configurations in order to assign the one that fits the tracked satellite. Finally, it also has to manage the schedules of each chain and know when one is busy and can not schedule a specific pass.

The **Health Tests Controller** has access to all the specific tests that each chain can execute. It also has access to the adapters in order to run the aforementioned tests. Finally it also manages the health tests schedule and the generated reports. Both the *Scheduling Controller* and the **Health Tests Controller** require read and write access to the information. The *Scheduling Controller* needs to know the schedule of health tests and passes and write access to the passes schedules. The *Health Tests Controller* needs access to the available health tests and the passes and health tests schedules. For that reason, another layer of extraction is added to the software. There is an extra **Database Controller** whose job is to effectuate all database accesses and provide the controllers and the Task Executor an interface to the information. Having all the database accesses in a different controller make it easy to replace the database structure or technology in the future, without needing to modify the core software.

The **Rig Controller** and the **Communications Controller** are the controllers that interact with the hardware. They use the Adapter functions that produce changes in the state of the hardware. On one hand, *Rig Controller* is the one in charge of moving the antennas, and configuring the SDR to, for example, adjust the Doppler frequency shift during a pass. It is also responsible of turning on and off the software and hardware necessary to perform the aforementioned operations. On the other hand, the *Communications Controller* is the one that manages the connections with the uplink and downlink sockets. It is also responsible of turning the transmission mode to reception or transmission, and transmitting a command if requested.

Finally, the **Pass Controller** aims to encapsulate the pass execution process. In order to make sure that each function is executed at the required time. It has three main functions. The first one is executing pre-pass operations which include starting the hardware and moving the antennas to the starting position. The second one is executing the pass itself. It moves the antennas at the position they need to be at each instant of time and updates the Doppler when necessary. Finally, post pass operations include moving the rotor to parking position and stopping the hardware. The *Pass Controller* also has access to the *Communication Controller* in order to send commands to the satellite dur-

ing a pass and redirect the data flow to the operators. It is important to point out that the *Pass Controller* implements a timeout pattern to access the hardware resources. When starting the hardware, it checks that everything started correctly and retries a number of times before timing out in case it fails. The aim of this implementation is to give more sturdiness to the software.

6.2.4 The Task Executor

The **Task Executor** acts like the brain of the system. It is in charge of commanding the start of both passes and health tests. In order to do that, it has two main behaviours. The first one is reacting to the creation of new scheduled tasks, such as new scheduled passes or new scheduled health tests. The second one is executing those tasks at the required time. The first behaviour could be seen as setting an alarm to perform a task at a certain time and the second one refers to begin executing that task once the alarm rings. In order to know when a new task is scheduled, it uses an *Observer* pattern, where the corresponding controller (Scheduling or Health Test) notifies to the Task Executor that a new task was scheduled. When it receives that notification, it reads the new task through the Database Controller and creates an event at the specified time. Finally, the Task Executor reacts at the start of an event and executes the corresponding task. This is known as Event-Driven Architecture, where there exists an event generator (the operators requests and the scheduling start), and an event processor (task executor) that executes its reaction (executing the pass or the health test).

Each task is applied to a specific chain. This means that a chain cannot do more than one task at the same time, but multiple tasks on different chains can be performed. For that reason, the task executor has to execute them concurrently and trust the Mux and the Controllers will do their job to not have multiple accesses to the same chain.

6.2.5 Operations through the API

The API is the entry point to the software used by the operators, both from the OpCen and from GS if necessary. As in the controllers, its endpoints are divided in groups according to its functionality. The full design of all the API calls can be found in the Appendix B.

The **Scheduling endpoints** include scheduling, editing and deleting a pass. They all call the *Scheduling Controller* and if everything went as expected, return the user the id of the scheduled pass or an ok answer.

Next, the **Health endpoints** give the operator the option to schedule a new test calling directly the *Health Test Controller* or to execute one right at the moment, requesting it to the *Task Executor*, which will execute it if there is no pass using that chain. Both options will return the operator an identifier of the health test if everything went as expected. The endpoints also offer the options to get the results of the tests and

to delete a scheduled test.

The **Uplink** offers the option to send a command to a satellite if there is a pass at the moment. This will be managed by the *Task Executor* which is the one that controls the execution of passes and health tests. Similarly, the **Downlink endpoint** gives the connection parameters to the data flow sockets. This is also managed by the *Task Executor* as those sockets only exist during a pass. Finally, the **Management endpoints** allow the operators to access the hardware functionalities manually. This endpoints interact directly with the controllers, and should have priority over the *Task Executor*, as they should only be used when there is no pass or in case of something going wrong with the automatic process.

6.2.6 Database

In order to decide the database type, the type of data to be processed has been analyzed and the use of a **Relational Database** [30] has been determined. The data types used by the GS Software are the ones as follow:

- **Passes Scheduling data and Health Tests Scheduling data** is made up of a specific group of data types, which are the ones specified in the request to make a schedule, including date and time or the necessary parameters to execute a pass or a test successfully. For this same reason, since they are already defined specifically in the request interface, they can be exported to a tabular database.
- **Downloaded Satellite data** is saved in a raw data file, which can be processed later on. The database can save the path to that file in order to make it easy to find an specific pass Satellite's data. This kind of information can be stored in a relational database saving the path and the filename in a row.

The GS Database is composed of two independent parts, the tables related to the passes and the tables related to the health tests, as it can be seen in Figure 6.3.

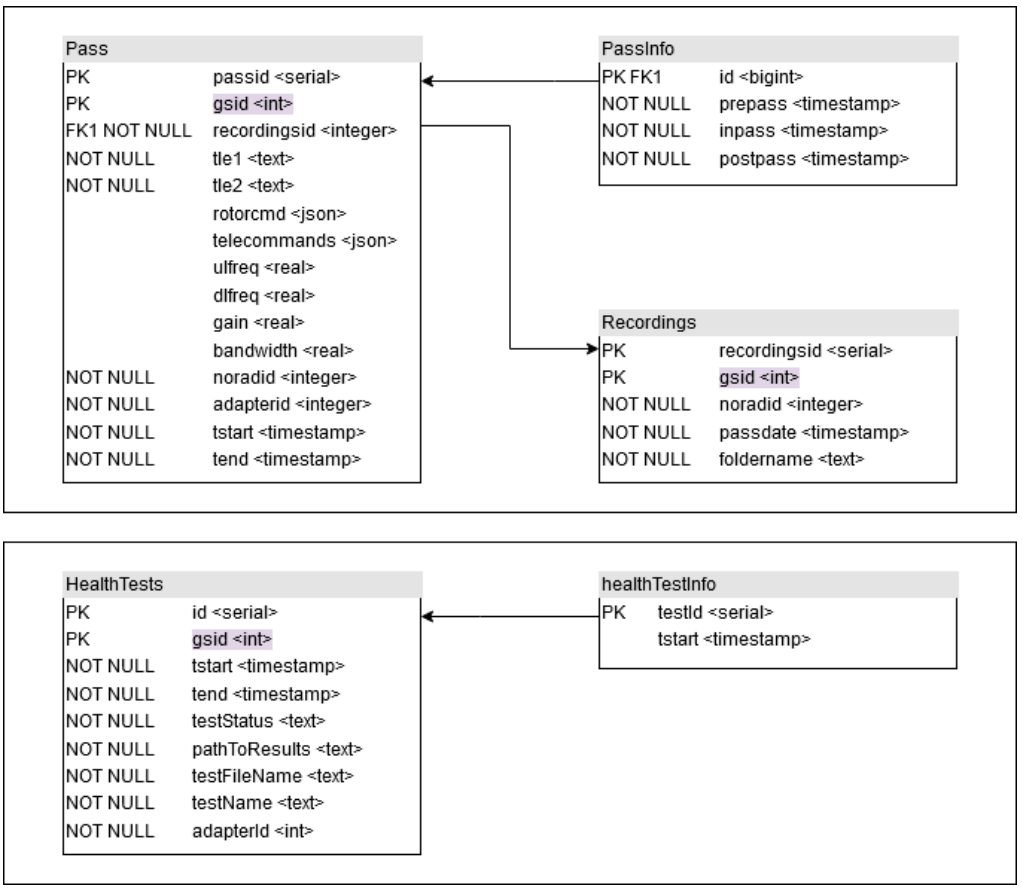


Figure 6.3: GS Database Structure

On the **passes** part there are three tables. The main one is the **Pass** table which includes all the information collected in the *Schedule Pass Request* explained above. It references the **Recordings** table which has the information of the paths and filenames to the Satellite's data generated files. Finally, it is related to the **Pass Info** table which is the one read by the *Task Executor* software in order to know when to trigger functions.

Similarly, in the **health tests** part, we have the main table **Health Tests** which contains the needed information to schedule an automated test, such as starting and end time or the test that needs to be executed. There is also the **Health Test Info** which is the equivalent to the *Pass Info* one.

6.2.7 Software robustness

The GS Software has to be robust as it needs to be able to recover from errors. Software errors should be managed in the implementation, but hardware ones should be taken into consideration such as the weather conditions in the GS causing power or internet outages. The most critical parts of the software executions that could lead to inconsistencies if some hardware accidents occur are:

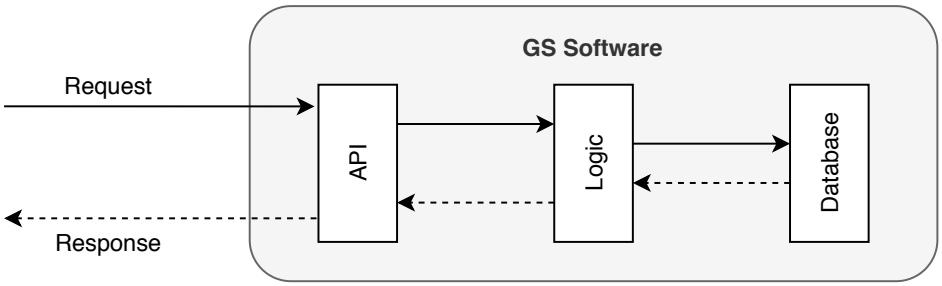


Figure 6.4: API call transaction

- **API calls:** It is important that when the API returns a response, all the corresponding changes in the software (such as scheduled passes or health tests) cannot be lost if an incident occurs.
- **During a pass:** The software has to be able to run the pass if there is internet outage (no connection with the OpCen) and to continue it if electricity fails and comes back.
- **When there is a future scheduled task:** All the tasks scheduled before the incident have to be scheduled again and executed on time once the incident is repaired.
- **Managing downloaded data:** All the received data should not be lost during power or internet outages.

Regarding the API calls execution, the design can guarantee that no data will be lost. As it can be seen in Figure 6.4, when a call is made, the software processes the request and stores its result to the database if necessary before returning an answer to the caller. In both internet or electricity outage, the API will not receive an answer that is not recorded.

The software is also resilient to incidents during the execution of passes and future pending tasks. As explained in the *Task Executor*, when the passes and health tests are scheduled, an event is stored in the *Pass Info* table. If there is no connection with the OpCen, the GS has a local copy of its schedule, so it can keep executing tasks automatically. Concerning power outages, when the *Task Executor* starts running, it loads all the events that are occurring at that specific time and all the future ones. Thanks to this mechanism if there is a power outage during a pass and it recovers in time, the *Task Executor* will resume the pass operations automatically.

Finally, the received data has to be available even if there is no connection. For that reason, the data is not only transmitted to the OpCen during the pass (using sockets) but it is also stored to a file which can be accessed at any time. If there is no internet these files can be accessed once the connection is recovered. In case of power outage,

the files will contain the part of the pass that could be recorded only. In Table 6.1, all this explanation is summarized.

Table 6.1: Hardware incidents management

	Power outage	Internet Outage
API calls	Results stored to the database before finishing the transaction	
During a pass	Pass resumes once electricity is available again	No internet required
Scheduled taks	Future tasks are rescheduled by the Task Executor	No internet required
Downloaded data	All received data while there was power is saved	Data saved to files

6.2.8 Software scalability

The software has separated layers for the interface, the logic, the access to the database and the access to the hardware. Thanks to this design, this software can be installed in all kinds of GSs as it is only required to implement the multiplexer and the adapters interfaces. The rest of the software works identically. Furthermore, having the database controller separately in a separate layer enables to easily swap the database structure or technology without having to modify all the code. Only the database controller should be modified to match the new structure or technology. Finally, by having a separate layer for the interface, enables the software to be expanded and to add more interfaces that are compatible with the same logic. For example, if in the future someone wanted to create a local program to control the antennas manually, it could directly call the controllers instead of the API.

7 | Operation Center Design

The OpCen is the responsible of the control of the satellite missions. It has to be able, with GSs' help, to manage the communication with the satellites. This management includes not only passes scheduling, but also commanding the uplink instructions and having access to the received data. As the central part of the system, it is also responsible of taking care of the GSs' status as well as turning them on/off in case of failure. Finally, from an external point of view, it has to be able to provide clients an easy way to schedule their own passes.

7.1 Operation Center Overview

The key point of the OpCen design is that it has to be used by the operators. As mentioned in the *Context and Motivation*, there are three types of operators in a satellite mission, related to the specific task they have to carry (uplink control, downlink control and GS manager). As each of them have an independent task, the software is divided in four additional softwares, each of them covering one of those tasks plus one public software displaying information about the passes in real time. As seen in Figure 7.1, these four softwares will communicate with a common backend through an internal API, each with its own entry points to access only what they need to.

The aforementioned backend is the part of the software which is in charge of three main tasks.

- The main one is serving the operators requests, which may involve communicating with the GSs, managing the GS and satellite's network or accessing received data stored in the local database and file-system. This includes all the API calls with its associated logic made by the aforementioned subsoftwares.
- Another task is serving external clients requests, such as scheduling passes or getting the data from those passes once they occur. This should be done using an authenticated external API which is open to those clients.
- Finally, for security reasons, the software will have to be capable of providing user sessions management through an authentication service in order to guarantee that each role accesses only the endpoints that are available to it.

It is basically composed of an authenticated backend server which has access to a local database and external data services. This backend can be accessed by external users using what we called the *external API* and by the operators using the different subsoftwares mentioned above, which have access to an *internal API*.

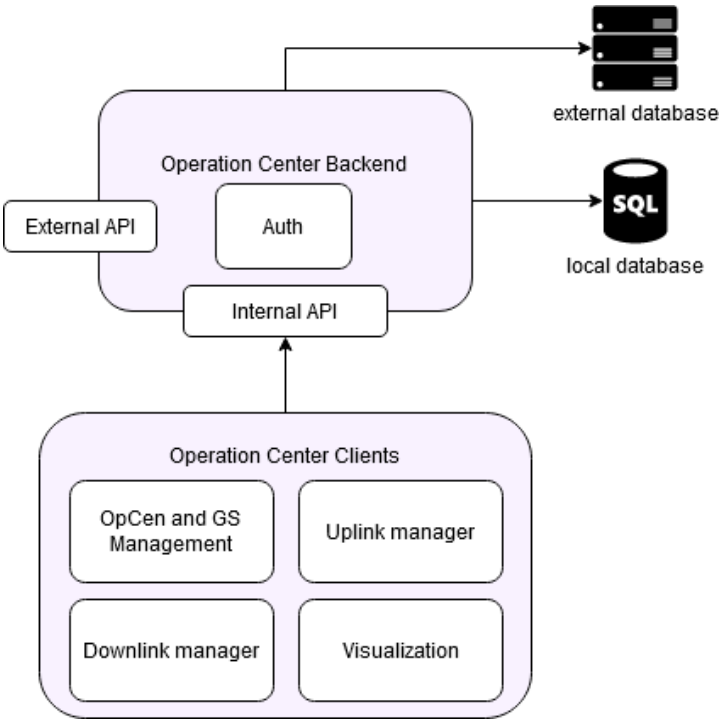


Figure 7.1: OpCen overview

7.2 OpCen GUI Design

7.2.1 Operation Center Management Client

The Operation Center Management Client is a dashboard-style software that enables the operator to interface with the GSs and the data and events related to it in an easy and organized manner. It is composed of a sidebar menu that grants access to the management of the infrastructure and a main window that displays the content of each menu entry dynamically.

Satellite's management

The satellite's management tab enables the operator to add new satellites when launched, update the information of the existing ones and deleting them. It also gives the operator a list of satellites automatically downloaded and auto-updated from Celestrak.

In order to make it accessible, the satellites are shown in card format, as it can be seen in Figure 7.3, which contains a summed up version of the full satellite's information. It includes shortcuts to the *set as favourite* and the *auto-update from Celestrak* features, which will be explained below. A detailed Mockup of the cards can be seen in Figure 7.2

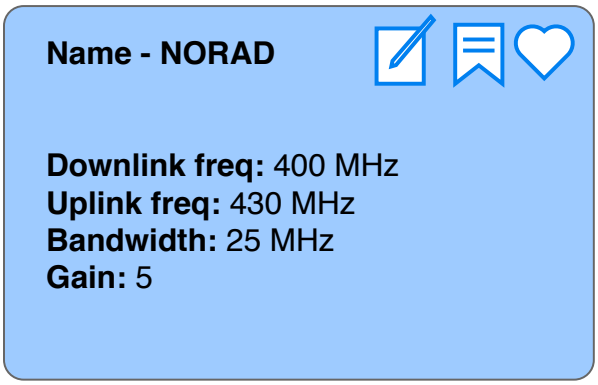


Figure 7.2: Satellites' Card Mockup

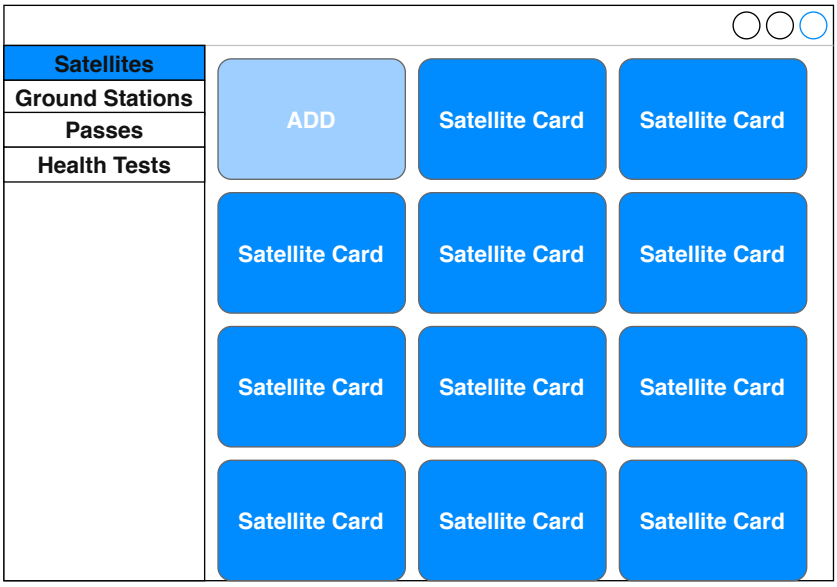


Figure 7.3: Satellites' Tab Main View Mockup

Clicking a card will show the satellite's full information, such as the working frequencies, gain and bandwidth. Adding a satellite is as simple as clicking the empty card, which opens a form in a modal on top of the main menu and filling it with the necessary information, as seen in Figure 7.4. Editing an existing satellite can be done through the edit button in the satellite's card or from the edit button located in the satellite's full information view. A satellite can be deleted from the edit form.

This tab is important because it will help the operator to schedule a pass as the satellite's information can be auto-completed instead of having to fill all the parameters for every pass.

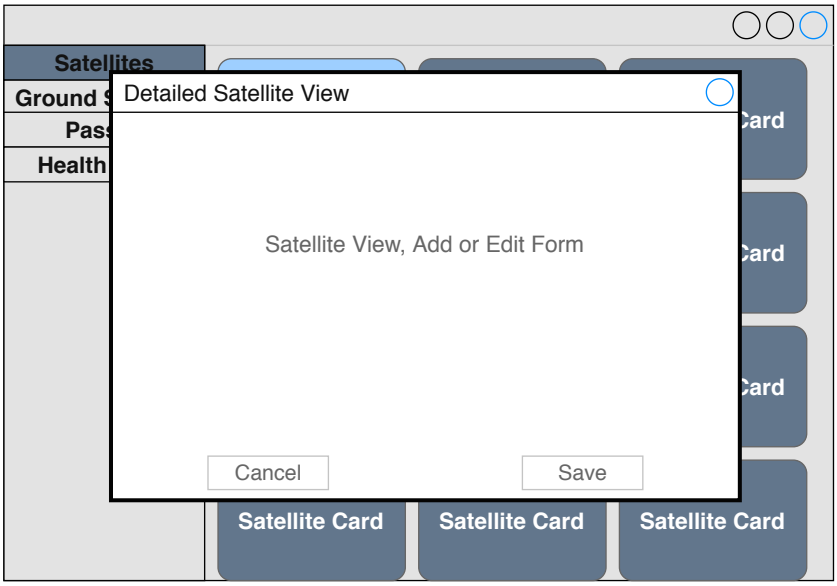


Figure 7.4: Satellites' Tab Forms Mockup

Ground Station's management

The Ground Station's management tab lets the operator to see which GSs are in the network and offers functionalities to test their correct functioning such as *manual control of the antennas* and *health test execution*.

The main window shows the GSs in a card with a picture of it. It contains the basic information such as the position of the GS and the chains it has. When clicking on a card, it redirects to the GS control panel. Figure 7.5 shows the main window while the control panel can be seen in Figure 7.6.

The GS control panel grants access to the GS chains (and antennas). When one chain is selected, the page shows the tests that the chain can run, information about that chain and video streaming of the antenna together with controls to move it in real time.

Using the buttons placed below the video streaming, the operator can move the antennas manually and its current position will be displayed at each moment. Selecting a health test from the list and clicking on the execute button, will automatically start the test.

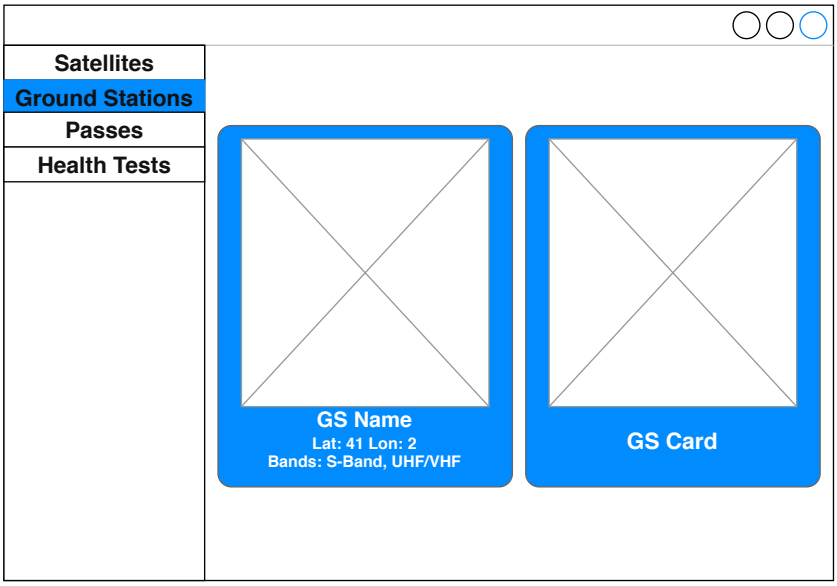


Figure 7.5: Ground Stations' Tab Main Window Mockup

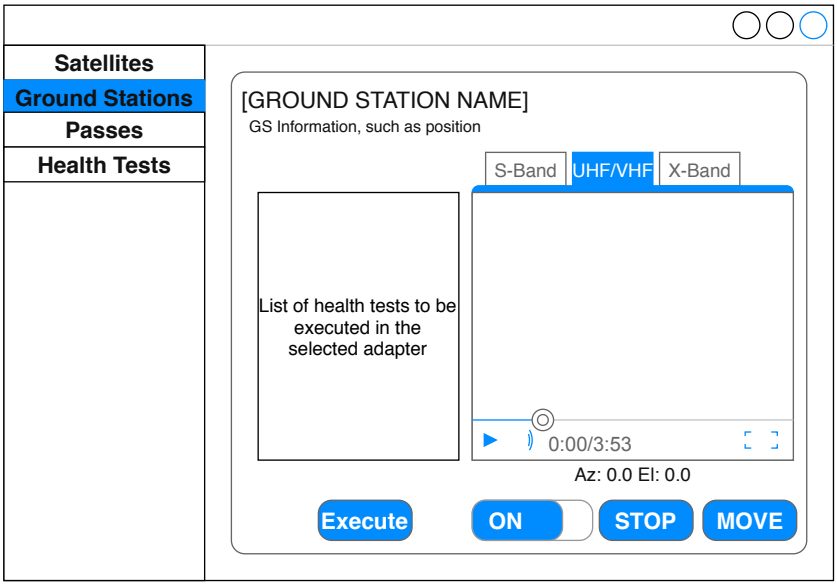


Figure 7.6: Ground Stations' Tab Control Panel Mockup

Passes scheduling

The Passes scheduling window is used to manage passes, both to visualize or create new ones. The main view consists of a table containing all the future passes with the essential information about them. This way, the operator can have a quick view of which passes are next and should be ready for. Figure 7.7 shows a first schema of this view. The table contains the following columns:

- **NORADID**: identifier of the satellite.
- **Satellite's name**: human readable form of identifying the satellite.
- **Start time**: time in which the GS will start preparing for the pass (starting software, moving hardware to starting position, etc.)
- **End time**: time in which the GS will completely have finished the pos-pass operations, including moving the atenna to the parking position or closing data files properly in order to be synchronized with the OpCen
- **Acquisition of Signal (AOS)**: time in which the GS starts receiving signal from the satellite.
- **Loss of Signal (LOS)**: time in which the GS stops receiving signal from the satellite. Along with the AOS, it delimiters the time in which there can be data exchange with the satellite, including uplink and downlink communication.
- **Maximum elevation**: Maximum position of the satellite in the Elevation coordinate. It is useful to calculate this as some antennas cannot track certain elevations for hardware reasons.

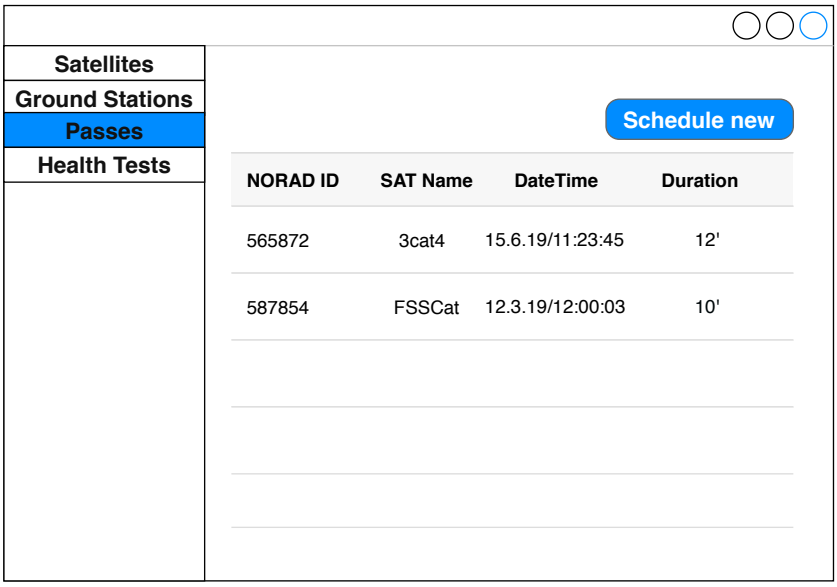


Figure 7.7: Passes' Tab Main View Mockup

In order to schedule new passes, there is also a button in the main view that when clicked opens a form with all the necessary parameters that need to be filled. Figure 7.8 shows where the form is displayed. It lets the operator schedule passes in different manners in order to make it more user-friendly. The available options are the ones as follow:

- **Single pass:** Schedules the next available pass, no extra information is needed.
- **N passes:** Schedules the next N available passes, the operator has to introduce the number of passes that need to be scheduled.
- **All passes until a specific date:** The operator introduces an ending date and the OpCen schedules all available passes until that date.
- **All passes in a time interval:** The operator introduces the time interval for which the passes need to be scheduled and the system schedules all the available ones between those two dates.
- **N passes from a starting date:** Similar to the one above, the operator introduces one starting date and a number of passes. The system schedules the next N available passes starting from the aforementioned date.

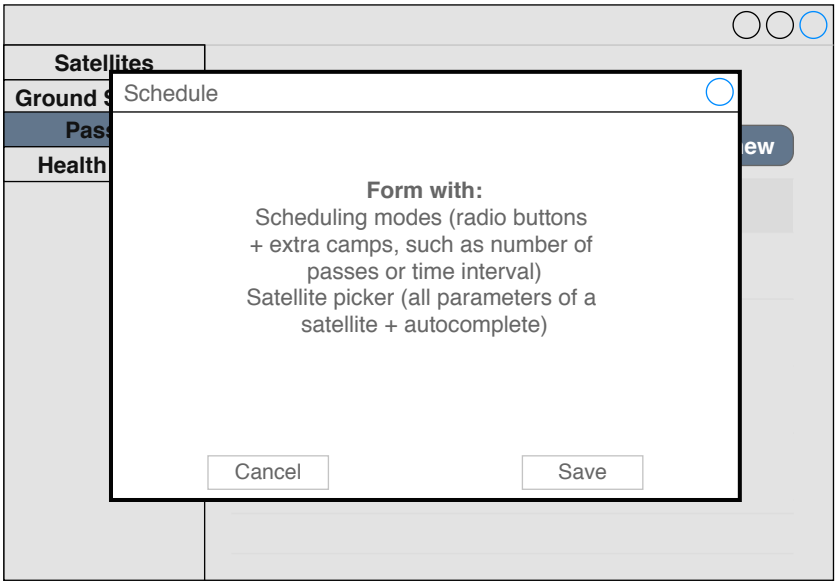


Figure 7.8: Passes' Tab Schedule Form Mockup

Finally the operator selects the satellite to be tracked and edits any satellite parameters if needed (i.e. updating TLE or adjusting frequency). After submitting the form, a loading ring will be displayed until all the passes have been scheduled. Once scheduled, a summary of the scheduled passes will be displayed on screen and a close button will redirect the operator to the main screen of the passes tab.

Health Tests scheduling

The health tests tab displays a calendar with the future health tests and a list of health tests reports. The first design of this view is shown in Figure 7.9. The list of health tests reports displays the last finished tests reports as they are expected to be the more

consulted. The reports can be filtered by date, test name or GS and adapter in order to get older reports that are not initially shown.

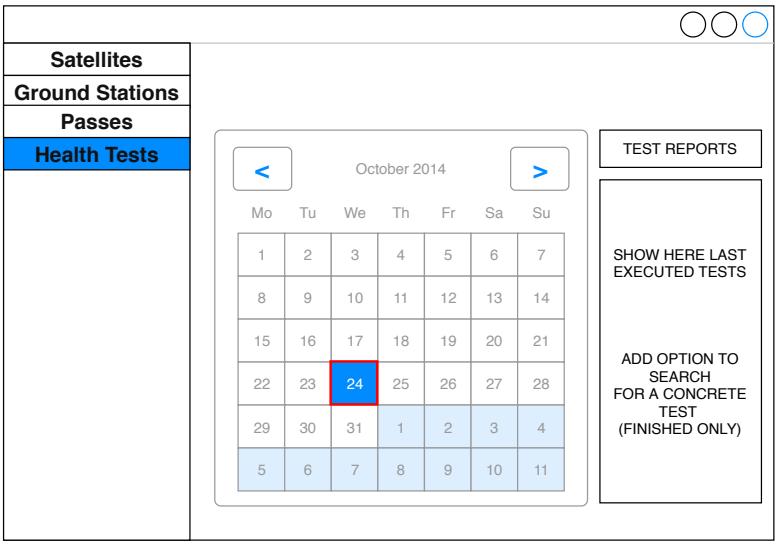


Figure 7.9: Health Tests' Tab Main Window Mockup

When clicking on a day on the calendar, a modal containing a form is opened. This form contains all parameters necessary to schedule a health test, including the GS and adapter in which it will run, the name of the test that needs to be scheduled, the creation date and the repetition period. When clicking on a scheduled test in the calendar, a similar form will open in order to edit or delete that test. This is shown in Figure 7.10

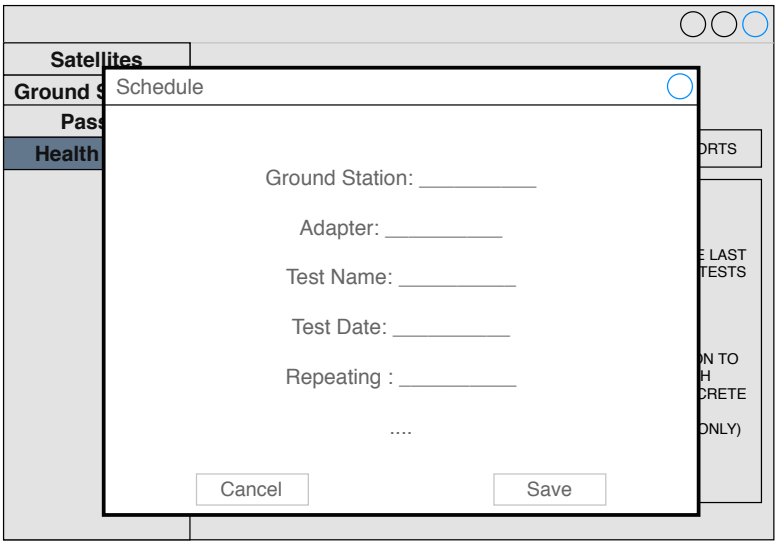


Figure 7.10: Health Tests' Tab Form Mockup

7.2.2 Downlink Control Client

The Downlink Control Client is used primarily to visualize data received from the satellites. This data should be retrieved from the GS which is currently tracking the pass using sockets or some other form of redirect through the OpCen backend services. It can include different kinds of graphs according to each specific satellite. It will also show general information about the pass, such as the current position of the satellite, the percentage of the pass which has been already covered, starting and ending times, etc. The first idea for the client interface can be seen in Figure 7.11.

Figure 7.11: Downlink Control Client Mockup

7.2.3 Uplink Control Client

The Uplink Control Client is used for managing the commands that might want to be sent during a pass. It has three main components. This components can be seen in Figure 7.12. The first one is a list of the known commands that a satellite can receive. As mentioned in the *Context and Motivation*, satellites have a set specific commands that can be sent to them, so it is easy to save them and show the ones that one satellite can understand.

The second component is a record of the previously sent commands during that pass. It should show the command sent, with the specific parameters and the time at which it has sent.

Finally, there is an editable graph of commands in which the operator can describe a flow of commands. This graph should be editable before the pass and during the pass.

Thanks to this graph, the operator can organize the communication flow and know at which point it is at each moment of the pass.

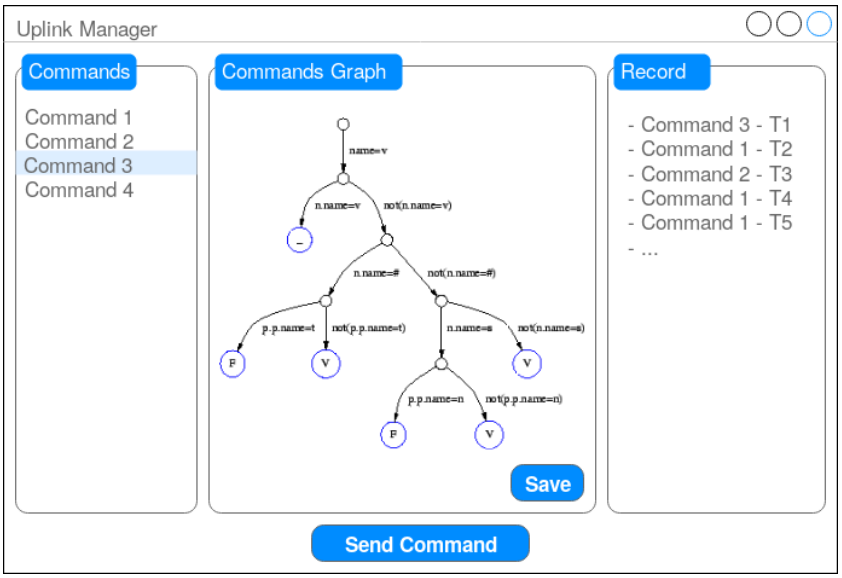


Figure 7.12: Uplink Control Client Mockup

7.2.4 Viewer Client

The Viewer Client is intended for serving as a displayable static page that contains informational data for the public. It should be displayed in a big screen and be seen by everyone who is not an operator. It should give enough information to keep everyone up to date whilst not interfering with the operational tasks.

The design consists of four components displayed in a grid layout, ordered as in Figure 7.13. It has a text area which displays information of the current or next pass and its satellite, such as its progress or the time until it starts. It also has a map with the current or next pass satellite's orbit to make it visual to know where it is at each time. A video stream of the antennas help the viewers see that the antenna's are moving during the pass and that everything is working as expected. Finally, there is a graph that displays the signal received by the antenna to make sure that data downloading is working properly.

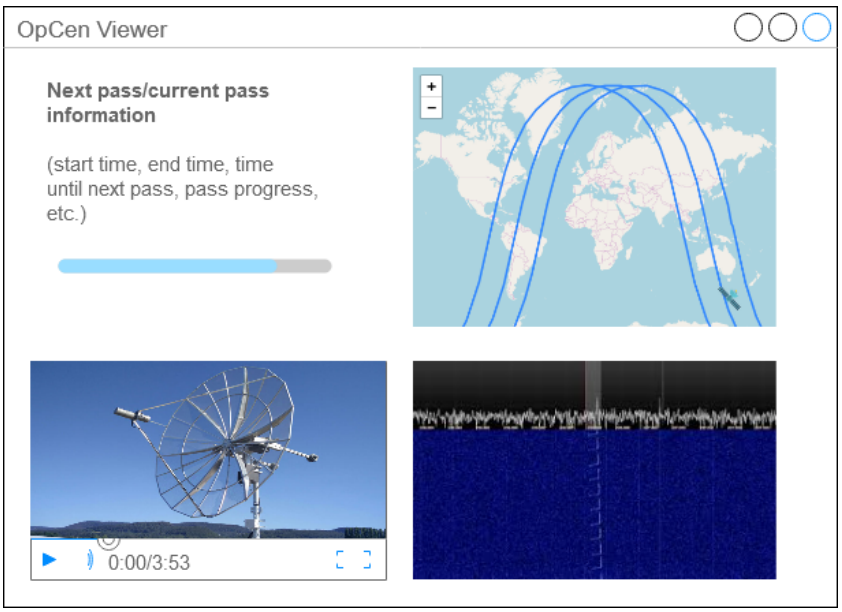


Figure 7.13: Viewer Client Mockup

7.2.5 UI/UX and Brand Image

When developing a GUI for a company, it is important that it reflects its brand. In this project it is even more important, as there is not only one GUI, but four (when the project is complete). For that reason, some design guidelines were defined.

Firstly, in order to create cohesiveness amongst the softwares, the components used should have the same look and feel. For that reason, all the softwares use Bootstrap components with its default Cascading Style Sheets (CSS). This helps the user to visually correlate the softwares as similar and makes them more user-friendly.

Secondly, it is important to give the software a brand image. For that reason, a color palette was created based on the color of the UPC NanoSat Lab Logo, which can be seen in Figure 7.14. The generated color palette is the one seen in Figure 7.15 and is applied as follows:

- **Dark accent #011d42:** This is the original logo color and is used as an accent color, for example in icons, checkboxes and buttons.
- **Background #849cbc:** This is the color used in the software background in order to highlight the components with white background.
- **Light accent #e0ecfc:** This color is used as a background of components that need to be perceived different from the others. One clear example of its usage could be the *Add Satellite* card, which should have different background than the existing cards.

- **Light background #ffffff:** White color is used for the background of menus, modals and cards.
- **Accent danger #f5365c:** Reddish color used for permanent destructive actions such as delete buttons.

Figure 7.14: NanoSat Lab Logo

Color	Hex	RGB
	#011d42	(1,29,66)
	#849cbc	(132,156,188)
	#e0ecfc	(224,236,252)
	#ffffff	(255,255,255)
	#f5365c	(245,54,92)

Figure 7.15: GUI Color Palette

7.3 Backend and API

The API is the entry point to the OpCen Software Backend. It includes endpoints to all the necessary calls the GUIs or external clients have to perform. In terms of security, all the endpoints of the API that are open for external clients should have an authentication middleware, in order to prevent unauthorized people from accessing the services of our

software.

The endpoints are divided according to the type of data they have to access or functionalities they have to cover, for example, a group of endpoints could be the ones related to satellite's data and a group based on functionalities could be the uplink management endpoints. The full design of the OpCen API can be found in the Appendix C.

The Backend software covers all the functionalities mentioned above. It is accessed through the internal and external APIs which call different controllers for each group of endpoints they have. Each controller has a specific functionality related to a specific part of the model, which is accessed only by one endpoint group. Thus, we have for example one controller for the Uplink functionalities or one for managing the GSs for instance.

Each public controller has two private controllers which can only be accessed by it. Those two controllers cover the two main behaviours of the OpCen, which are accessing the local database, and making calls to the GSs' API. Finally, some controllers might have access to external libraries such as a TLE downloader. This design separates the logical operations from the data access operations, making the software more robust and easy to change in case, for example, that we want to change the database technology or schema.

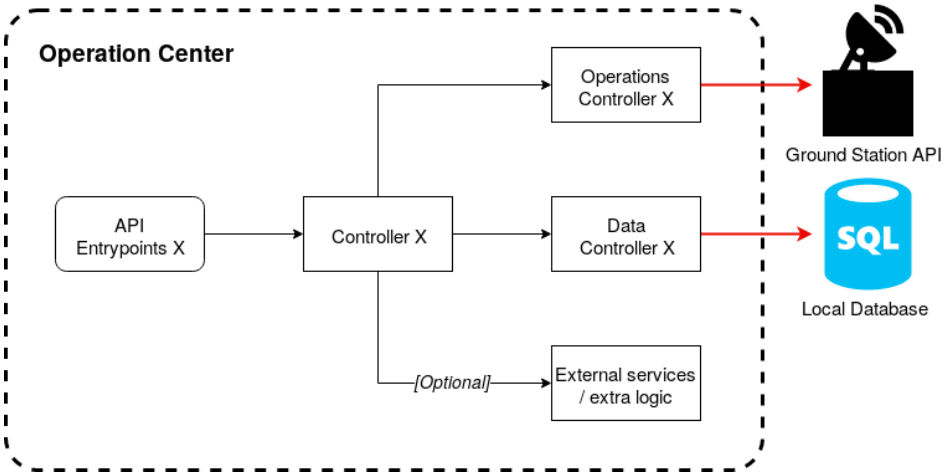


Figure 7.16: Backend Structure

The controllers are the ones as follow:

- **Ground Station Controller:** Enables the operator to retrieve the information of all GSs and do manual maintenance jobs, such as moving or stopping the antennas. It also has the required parameters to be able to see the cameras in real time.
- **Health Tests Controller:** Lets the operator schedule single or repeating health

tests to the GSs such as testing that the antennas move correctly or that the SDR is working properly. It also retrieves the results of the tests if those have finished.

- **Passes Controller:** This controller provides different ways of scheduling passes. It also retrieves all passes information in order to be displayed later.
- **Satellites Controller:** Has options to add, view, edit and delete information about satellites such as its frequency, bandwidth or TLE. It also includes functions to auto-update the TLE file.
- **Uplink Controller:** Includes all the functions related to editing commands graphs, consulting the commands history or sending the commands in order to transmit them.
- **Downlink Controller:** Grants access to the data flows received from the GSs during a pass.
- **Visualization Controller:** Provides the Visualization Control Client the endpoints to retrieve all the information it needs, including URLs to video streaming, passes information or calculated orbit points.

As all the only way to access the OpCen is through the API, all the requests have to be completed quickly. Most of the requests do not have much processing behind, as they are usually Hypertext Transfer Protocol (HTTP) GET requests or simple HTTP POSTS, so that is not usually a problem.

However, other requests such as executing a health test or scheduling multiple passes can not be done quickly. For that reason, those requests do not directly obtain the result but they receive an identifier. Using that identifier, they can poll the API until the result is already processed.

Using this request format for the more demanding requests offer two main benefits: (1) the user can do other jobs while the software is working and (2) the request does not take too long which means that no HTTP timeouts will be sent.

Finally, it also has some routines that are running periodically that help maintain consistency between the OpCen and the GSs, and keep all the satellite's information updated. They also call different controller's functions. The running routines are the ones as follow:

- **Celestrak TLE Auto-updater:** This routine periodically downloads the newest version of the TLE files used to track the satellite's position and updates the database's entries of the satellites that have the *auto-update* variable set to true. If there is a TLE file corresponding to a non-existing satellite, it also creates the database entry for it in order to have an updated copy in our database. The updates are made once a week as it is the refresh rate of Celestrak page for most of the satellites.

- **Ground Station Manager:** It sends periodically all the health tests repeating requests to the GSs in order to retrieve some workload from the GSs.

7.4 Database

Following the decision taken in the GS Database design, the OpCen also uses a Relational Database. The data types used by the OpCen Software are the ones as follow:

- GSs Network's information
- Satellite's information
- Passes Scheduling, Commands and Recordings
- Health Tests Scheduling and Results

The database of the OpCen has a more complex schema as it includes more objects to model and has tables synchronized from other databases, such as the GS database. The complete schema is shown in Figure 7.17 and is useful to follow the upcoming paragraphs.

The GSs Network's information is stored in two tables, the **Ground Station** and the **Adapter** tables. The first one contains the basic parameters to model a GS, such as its name, position and connection parameters. The second one includes more detailed information of the hardware capabilities of each band offered by each GS, such as its working frequencies or its maximum position in Azimuth and Elevation.

The Satellite's information is stored in the **Satellite** table, which has all the parameters needed to perform a pass, including uplink and downlink frequencies, bandwidth and gain and each unique satellite identifier, the NORADID. It is also important to highlight the auto-update boolean field since it indicates if the Celestrak TLE auto-updater has to overwrite the TLE of this satellite in case a newer one is found.

A pass can cross multiple GSs, and each of them will return a different pass identifier. This is handled in the **General To Specific Pass** which is an associative table that relates the GS pass identifier with the general pass one, which is stored in the **General Pass** table. The General Pass table includes TLE camps, which may seem redundant as it references the Satellite table, but it was made intentional in case the operator wants to use a different TLE from the one stored in the Satellite's table but does not want it to be overwritten. Finally, the **GS Pass Table** and the **GS Recordings Table** are copies of the GS database tables using Database Synchronization which will be explained later.

The Passes Scheduling requests are managed using the **Scheduling Request** table. It stores a request that may include multiple passes, as seen in the GS Management

Software part. The requests are related with the **General Pass** table using the associative table **Scheduled Pass** which helps knowing when each pass was requested.

For the Uplink, it is important to mark that the **General Pass** table has a column to store the commands graph. The available commands for a specific satellite are stored at the **Command** table and the history of sent commands is stored at the **Commands History** table.

Finally, for the Health Tests scheduling, three tables where created. The first one is the **Health Test Event** table, which contains the required parameters to execute a repeating health test, such as its name, the target GS and Adapter, the creation time and its repetition period. Having the creation time and its repetition period enables the OpCen Backend to calculate when a test needs to be executed instead of saving thousands of copies of the same test.

There is also the **Health Test History** which connects the Health Test Events with the **GS Health Tests Table**, and the **Cancelled Health Test** which has the ids of the health tests that have been cancelled and do not have to be executed, including the ones cancelled by the operators or the ones cancelled because there is a pass using that chain at the same time.

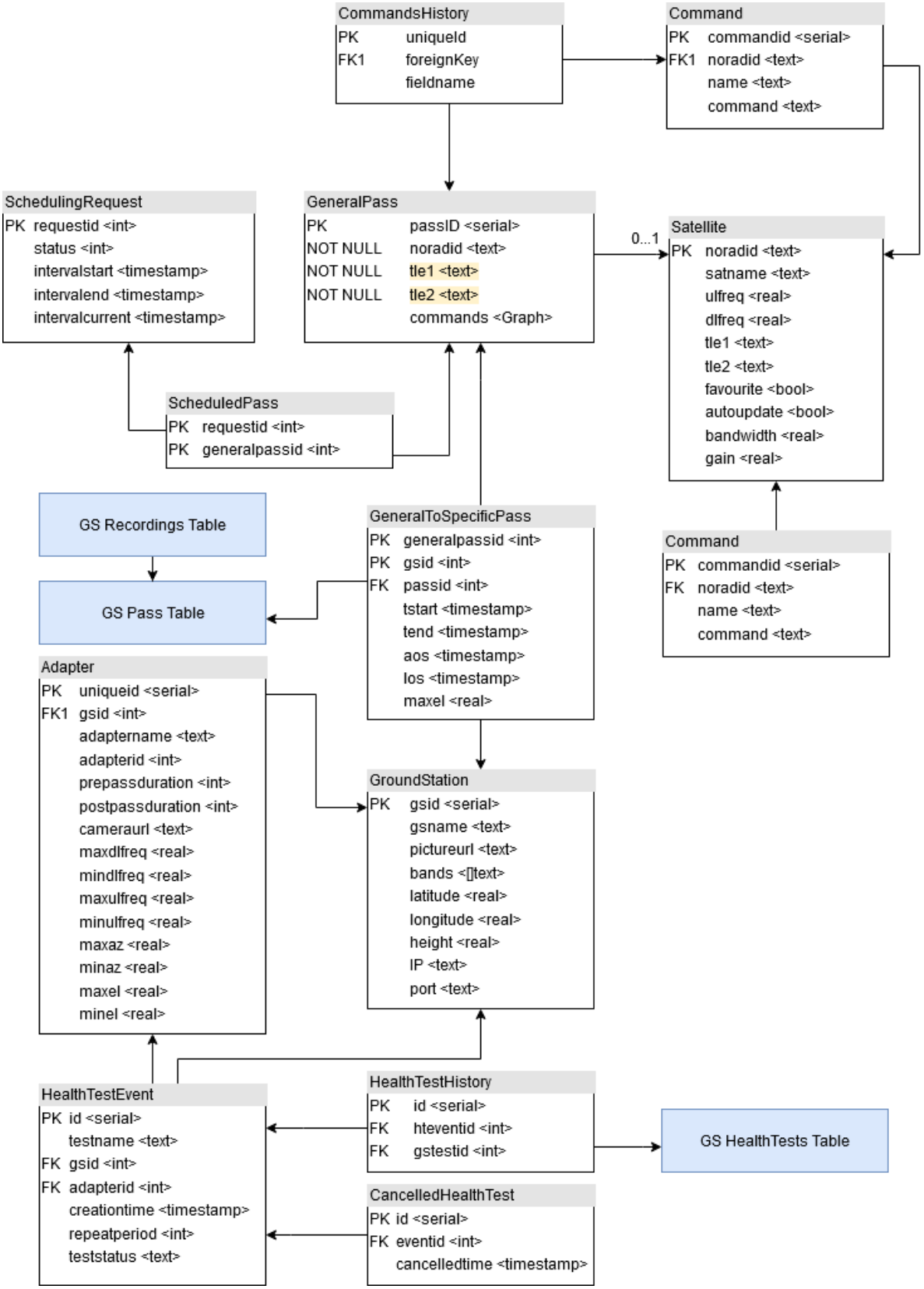


Figure 7.17: OpCen Database

8 System Communication Design

The main architecture requirement of the software is that it needs to be modular and extensible. For that reason, the GS was designed to make it really easy to add new adapters, and the OpCen design provides a simple method to add new GSs to the network. Both methods do not require any changes to the core software, as it will be seen in the following sections.

8.1 Adding adapters to a GS

In order to have adapters for the GS software, an implementation of the Mux Interface is needed. This interface will have access to all instances of the adapters that the operators want to use, and will switch between them depending on the requested frequency and the knowledge of whether it is in use or not. For each adapter, a specific implementation of the Adapter Interface needs to be developed according to the hardware needs.

8.2 Adding GS to the OpCen Network

The OpCen will know which GS are in its network by checking the database tables *Ground Station* and *Adapter*. So, in order to add a GS to the network it is required to add one row to the GS table with the connection parameters of it and a row for each adapter that the GS has. Finally, the GS direction needs to be accessible from the OpCen.

8.3 Data Synchronization

The GS and the OpCen have information in common, such as the scheduling of passes, the recordings made, and the scheduling and results of health tests. For that reason, some of the GS tables have to be replicated in the OpCen. These tables include Pass, Recordings and HealthTests.

The replication should be done from the GS to the OpCen, as one of the requisites for the GS was that they should be operable locally. This leaves us with a replication architecture of Many-to-One, as seen in Figure 8.1, which requires the masters to identify

themselves in each table so no autoincremented id gets duplicated, causing a duplicate primary key problem. For that reason, in the GS Database, there was a column called *gsid* in all the tables that required replication.

In order to manage file replication, the files have to be stored using the same relative path. For that reason, the Recordings table in the GS Database, stores a relative path to the recording file instead of an absolute one, so when it is replicated in the OpCen, the files can be accessed using the same path and not getting any file-system error. The root of the path can be stored in a configuration file. By having a system that can replicate from */rootGS/path* to */rootOpCen/path*, it is easy to keep the files well organized and following the same structure regardless of the storage structure of each of the GS.

This synchronization should be done in real time in order to not have inconsistencies between the files stored in the GSs and the OpCen.

As the data is synchronized using logical database replication, there will not be any inconsistencies in the stored data. Each table is either written by the GS software or the OpCen one, but never by both. Even if the OpCen reads the GSs tables, it does never modify them, so each GS will have only the data that it has generated and it will be replicated *exactly* as it is.

The OpCen does not generate any files, so the same reasoning applies to the file synchronization mechanisms. Each GS replicates its own data to the OpCen and does not read other GSs' data, so they are isolated from each other.

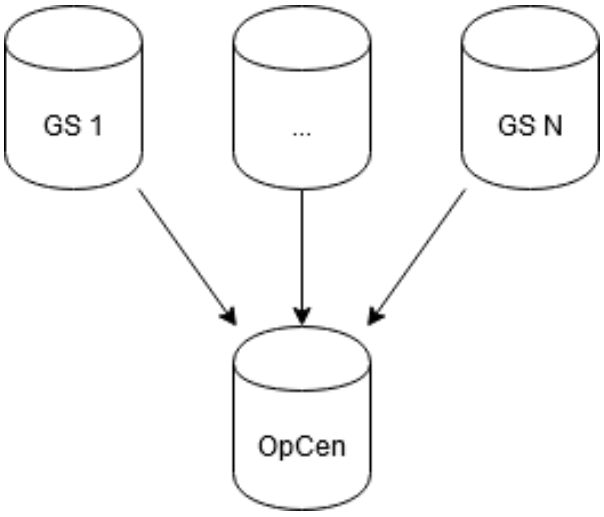


Figure 8.1: Many-To-One Replication Architecture

8.4 API

The OpCen interacts with the GSs through API calls. Some of the calls include assigning a pass to a GS or moving the antennas manually, as explained in the GS software design section. The GS, however, never begins any communication with the OpCen as it only responds to the received requests.

8.5 Secure Connection

The connection between the GSs and the OpCen has to be private and secure. The GS should only be accessible through the OpCen and the OpCen does not require any incoming communication with the exterior. For that reason, everything should be installed under local networks interconnected with point-to-point connections and making use of a Virtual Private Network (VPN).

The VPN [31] extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network. In Figure 8.2 the behaviour of a VPN is shown.

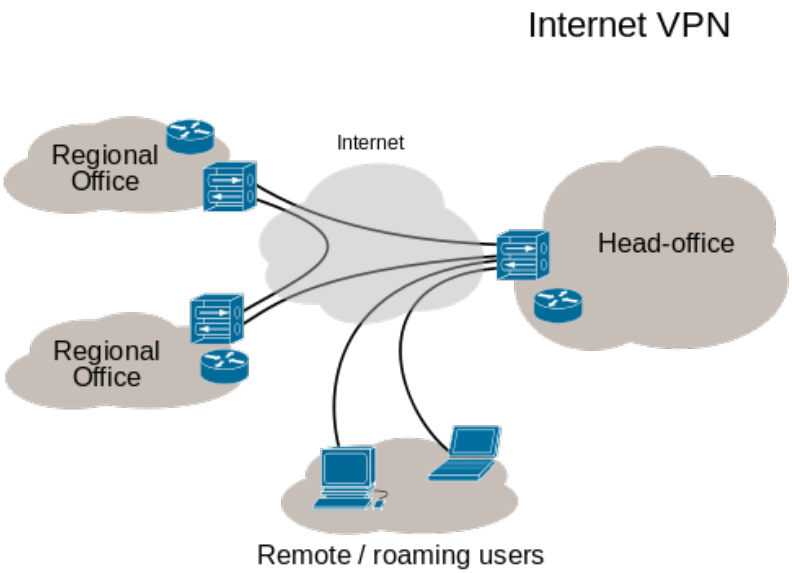


Figure 8.2: VPN example

9 | Selected Technologies

9.1 Ground Station technologies

The Ground Station logic has some technical requirements that need to be met by the technology implementing it, which are the ones that put hard requirements on the used technology in order to follow them.

- **Concurrent API:** The language has to support APIs creation mechanisms as the entry point to the software will be done through an API.
- **Good error handling mechanisms:** As one of the key points of the GS is that it can manage automatically what it has to do at each instant of time, it is of special importance that in case of any errors that may occur during the software execution, the GS knows how to handle it and not crash.
- **Software concurrency:** The software has to be capable of doing different things at the same time. For example, it can receive requests for scheduling a pass or a health test while another pass is executing. Another example is that it has to be able to execute two different passes from two different chains. Due to this concurrency in the GS responsibilities, it has to be able to work concurrently.
- **Threads communication:** As the software has to be robust (as mentioned in the error handling point) and has to be able to work concurrently, it also needs to have a good communication between related threads in order to assure that their status is the expected at each instant of time.
- **Multiple adapters support:** It has to be capable of working with different adapters, meaning that it is needed to be able to override functions according to the called adapter.
- **Access to PostgreSQL database:** As mentioned before, the technology used for the database is PostgreSQL as a relational database is needed. The technology used for the logic of the GS will need to be able to work with that database in order to interact with the data.
- **Good performance:** As the user entry point is an API working with HTTP requests, the ground station has to give a fast answer as the server will timeout

otherwise. During the pass scheduling request, the GS has to propagate the satellite's orbit in order to determine the time in which it will be visible for the GS. The orbit propagation and calculation of the directions in which the antennas have to point require many operations so it is important to have a fast language.

- **Compatible with GNU Radio scripts:** As the transmission chains of the GS are configured and controlled by GNU Radio scripts, which are written in Python, the software has to be capable of managing the execution of the aforementioned scripts.
- **Access Linux shell:** It has to be able to execute and manage the status of different shell commands as they are be required for the correct functioning of the hardware.
- **Easy to test:** As the software needs to be reliable, it has to be tested properly. For that reason, the technology used must provide good testing options, including unit, integration and system tests support, and code coverage tools.

Two languages were proposed to develop the GS logic: Golang or Go[1], and Python[2]. In order to make a decision, these two languages were analysed to see which one would bring better results for the project.

Both languages provide a way of creating APIs. Golang has *net/http* which, along with *gorilla/mux*, offer a really easy-to-use request router and dispatcher. On the other hand, Python has Flask framework which helps in the creation of web applications and APIs.

In the error handling camp, Go and Python differ. Python uses exceptions to handle the wrong flow of the program with try/except clauses. Go uses errors, which are returned as a valor in functions calls. Both methods provide good error handling mechanisms but personally, I find errors to be more comfortable to work with.

As it has to be able to execute different functionalities concurrently, Go has Goroutines [32] and Channels. Goroutines are functions that run concurrently, and can be thought as lightweight threads that cost less resources to be crated. Those Goroutines can communicate with other parts of the software through Channels, which can be seen as pipes that connect different Goroutines. This provide an easy way to handle concurrency [33] and threads communication in a inbuilt manner. Python, on the other hand has no inbuilt methods to manage concurrency, which can only be done via system threads and queues, which can be difficult to manage and to debug.

In order to deal with multiple implementations of the functions that require access to the chains, the language has to provide some inheritance mechanism. In the case of

[1]**Golang:** https://golang.org/project/
[2]**Python:** https://www.python.org/

Go, that mechanism are interfaces, which are implemented by multiple objects. With Python, there is subclass Inheritance. Both are valid options to implement inheritance of functions.

Both of the alternatives have libraries to work with PostgreSQL databases. In the case of Go it is a third party library widely used called *lib/pq* [34] and in Python an adapter called *Psycopg* [35].

Regarding the access to Linux shell, both Go and Python have tools to call different Operating System (OS) commands. This feature enables the compatibility with the GNU Radio scripts as they can be initiated and controlled through shell commands. In the case of Golang, the library is called *os* [36] and is one of the predefined libraries. For Python, there is a module called *os* [37].

Both languages provide Table-Driven Tests (TDTs) and Code Coverages (CCs) tools which are very similar. Go offers a program called *cover [38]* and has the *testing* library [39]. Python also has TDTs using the *Pytest* library [40] and CC tools using *Coverage.py* [41].

Finally, Golang was the chosen language because it has a very well-implemented concurrency system and everything needed is included in the basic libraries, so there are no third party softwares dependencies except from the orbit propagator. Table 9.1 shows a summary of the characteristics described above.

Table 9.1: Go vs. Python comparison summary

	Go	**Python**
Concurrent API	Internal library: net/http	Flask framework
Error handling mechanisms	Inbuilt errors	Try/except clauses
Software concurrency	Inbuilt	-
Threads communication	Channels	Queues
Multiple adapters support	Use of interfaces	Inheritance
Access to SQL database	lib/pq Driver	Psycopg
Compatible with GNU Radio scripts	Through OS library	Through OS module
Access Linux shell	OS library	OS library
Easy to test	TDT and CC	TDT and CC

9.2 Operation Center technologies

The OpCen has two separate parts, the GUI and the backend. As the backend follows a similar structure as the GS's backend (API, logic and database access), Golang is also

used here. For the GUI, another technology is needed.

It was implemented as a Web App for two reasons. The first one is that it does not require to store any data or access the hardware of the device, which is the main strength of native applications versus Web Apps. Secondly, it is interesting to make it cross-platform as the only requirement to use the software is to be in the NanoSat Lab network.

As for the technology itself, some frameworks where put on the table such as React, Angular or Vue. Vue was chosen for many reasons. Some of them are the ones as follow:

- **Previous experience with the framework.** Previous experience with Vue in some other projects that required GUIs.
- **Extensive documentation.** The framework is well-documented and has a lot of examples to set up your project quickly.
- **Good community support.** There is a big community around it.
- **Many premade templates ready to use.** There are a lot of community-made components and templates which are free to use.

9.3 Database technologies

In order to determine which database technology can be used for our software, there are some requirements that can not be negotiated, which are:

- It has to be a **Relational Database**. As explained in the *Software design section*, all the data that needs to be stored is tabular.
- It has to be **compatible** with Golang. As it is the chosen language for the backend, it is important that there are easy ways to communicate it with the database.
- It has to have **replication tools**, as some of the information stored in the GSs' databases have to be replicated to the OpCen one.

Those three requirements are met by PostgreSQL[3] and MySQL[4], a comparison of both technologies was looked for, in order to take the final decision. A very specific comparison was found in the 2nd quadrant website [42] so the choice was made based on some points of this comparative.

Following the comparative mentioned above, the first thing to point out is the compliance of ACID properties [43]. PostgreSQL meets all the ACID properties while MySQL

[3]**PostgreSQL:** https://www.postgresql.org/
[4]**MySQL:** https://www.mysql.com/

only fulfills them if using InnoDB, otherwise some data could be lost. Regarding compatibility with other SQL technologies, PostgreSQL is more adapted to the standard than MySQL, which means that if in the future the database technology is changed for another SQL one, it will be easier to swap if we use PostgreSQL.

Another important factor is concurrency, because the backend may have concurrent calls to the API, it is important that the database can manage them properly if data is requested. PostgreSQL supports concurrency whereas MySQL only supports it if using InnoDB. Finally, PostgreSQL has a stronger community which is a very important factor when one does not know which difficulties may face during the implementation of the solution. For all the reasons mentioned above, PostgreSQL was the chosen technology to implement the databases for both the GSs and the OpCen.

9.4 System communication

9.4.1 Networking

The connection between the GSs and the OpCen has to be secure and private, as requested in the non-functional requirements, an VPN connection is needed. Some technologies that provide this service are OpenVPN[5] or WireGuard[6].

WireGuard was the chosen technology as it is a fast and secure VPN tunnel. It is really easy to set up and offers much better performance than other technologies in the market such as OpenVPN. Furthermore the NanoSat Lab already had experience using this technology and offers everything that was needed.

9.4.2 Database synchronization

All GS databases should be synchronized to a single database in the OpCen. In order to achieve this, the databases require some technology that could work with the postgreSQL databases and accept the many-to-one replication model.

The use Londiste or PgLogical by 2ndQuadrant[7] was studied. Londiste is Skytools' asynchronous primary/secondary replication system, built atop PGQ. PgLogical is a logical replication system implemented entirely as a PostgreSQL extension. Fully integrated, it requires no triggers or external programs.

At first, Londiste was installed but it did not work properly due to installation errors. As there is not a big community around it, it was hard to find a valid solution, so PgLogical was installed instead. PgLogical documentation is extensive and has great

[5]**OpenVPN:** https://openVPN.net/
[6]**WireGuard:** https://www.wireguard.com/
[7]**2ndQuadrant:** https://www.2ndquadrant.com/es/

community support online. It was easy to set up and did everything needed for the project, so in the end, PgLogical was the chosen database replication tool.

9.4.3 File synchronization

Like the connection for exchanging requests, the connection for synchronizing files must also be secure and private. In addition, this has to be done automatically in real time. In order to achieve this, Syncthing[8] was used. Syncthing is a continuous file synchronization program. It offers private, encrypted and authenticated synchronization in real time. As the rest of the technologies used for the development of the project, it is open source.

[8]**Syncthing:** https://syncthing.net/

10 | Additional software

10.1 Go-Satellite

As seen in the design, the Scheduling Controller interacts with an orbit propagator. An already implemented orbit propagator was used, the package Go-Satellite[1]. When comparing the results of it with a trustable propagation software, the final outputs were not the same. The output of every function was debugged and it was found that the conversion from ECI to Azimuth and Elevation was wrong. A new implementation based on the following Celestrak pseudo-code[2] was written and the results were pushed to the original creator.

10.2 GS Adapters

In order interact with the chains in the Montsec GS, two adapters were implemented, one for the S-Band and another one for the UHF/VHF chain. The structure of the connection with both hardware structures can be seen in Figure 10.1 which is explained in the following paragraphs.

Each component of the **UHF/VHF** hardware is accessible through Transmission Control Protocol (TCP) sockets. Each of them have an interface of the commands that they can understand and its format. All this communication was already implemented in a previous software, so it was adapted to Go and refactored in order to follow the *adapter interface* functions structure.

The **S-Band** hardware is more simple as it is all contained in the rotor controller. It is also accessible through a TCP socket with a fixed interface. A Go library was created in order to encapsulate all the calls to the rotor controller. While developing the adapter for the S-Band chain, the controller of the antenna was not accessible yet. For that reason, a **simulator** was developed too. It mimicked the interface of the controller and answered the requests with stub values, following the structure indicated in their documentation.

[1] **Go-satellite:** https://github.com/joshuaferrara/go-satellite
[2] **ECI to AzEl:** http://celestrak.com/columns/v02n02/

Finally, all those components' power can be switched using NETIO PowerPDU. The components can be turned on and off by making calls to the NETIO Interface, similarly to an API. Once again, all this calls have been encapsulated to a Go library which is called by both of the adapters implementations.

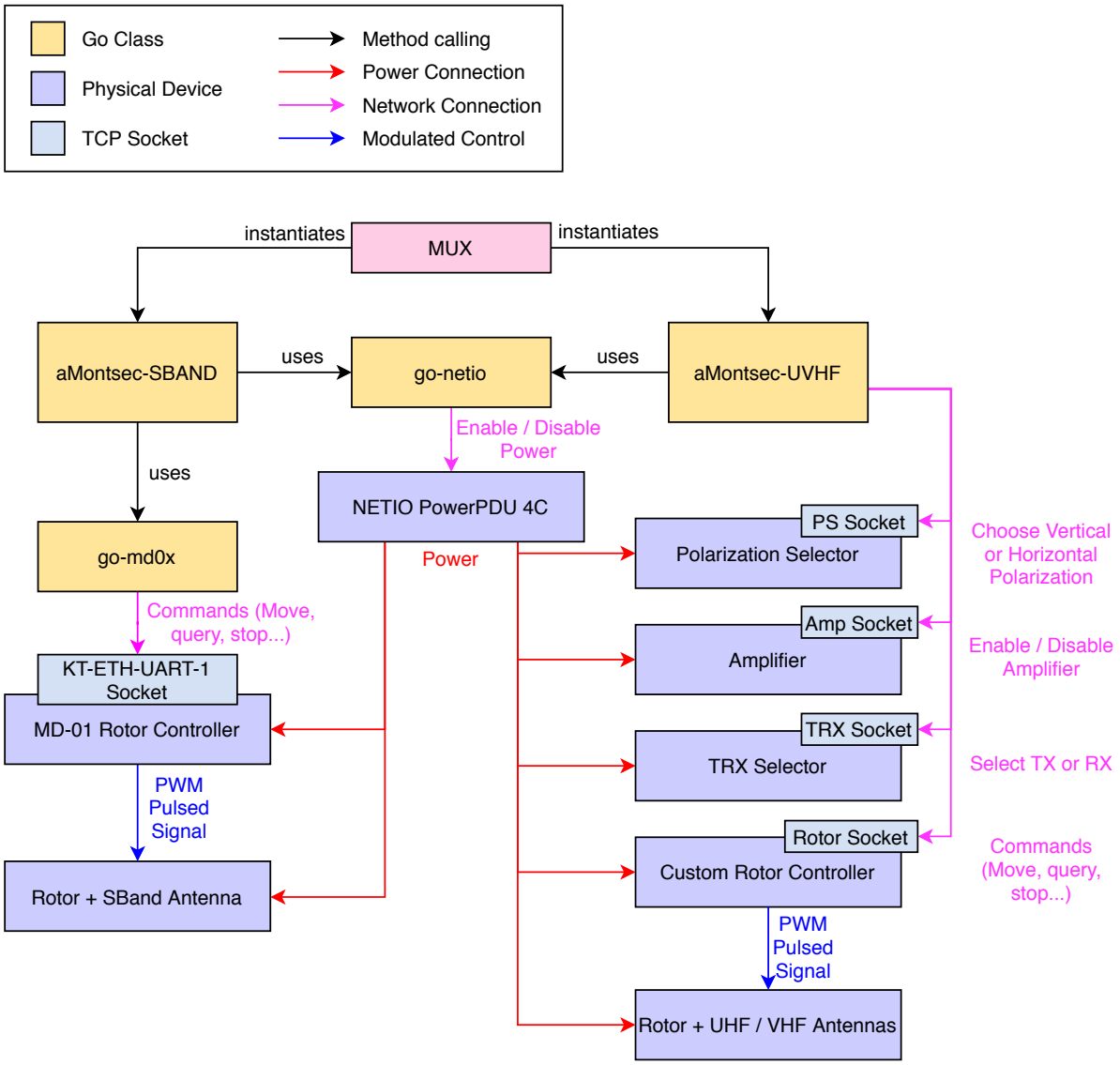


Figure 10.1: Hardware access configuration

11 | Deployment

11.1 System set up

Although the system can work using multiple GS, at the moment we only have one. The system is composed of two computers, one situated in the NanoSat Lab and the other one in the Montsec Observatory. In Figure 11.1 it is shown how those two computers interact.

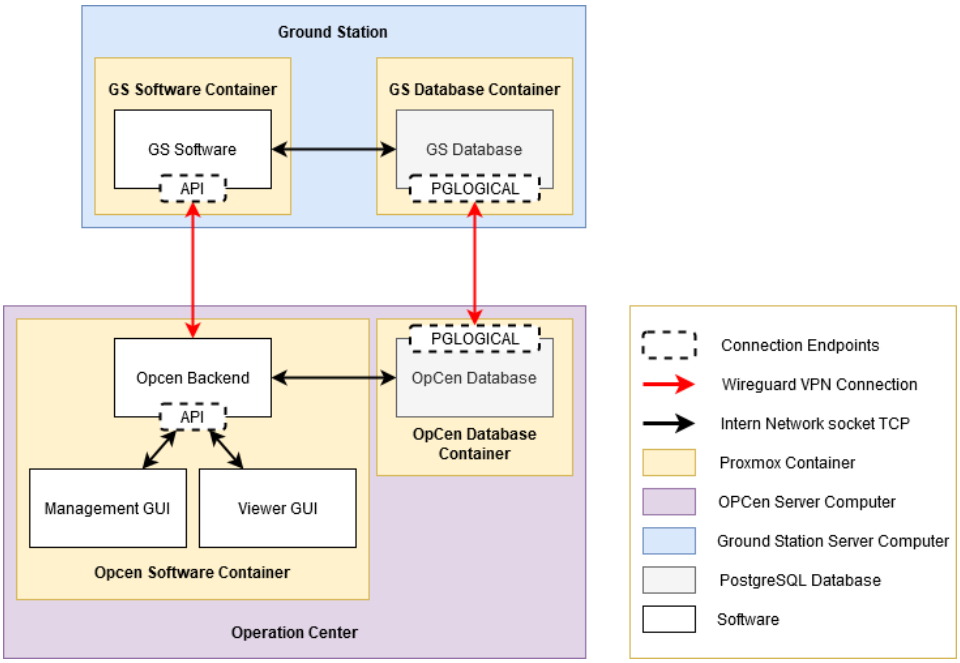


Figure 11.1: Final system connection

Firstly, each computer has Proxmox[1] installed, in order to create containers that will contain the software. For each computer, two containers were created: one that runs the software and a second one that contains the database. Using two different containers makes maintenance jobs much easier and if some part fails, one does not have to be careful to not break the other one. In the Software container from the OpCen, there are both the GUIs and the backend. The GUIs connect with the backend using a connection

[1]**PROXMOX:** https://www.proxmox.com/en/

inside the local network with socket TCP. The backend connects with the database in the same way, both in the OpCen and the GS.

All the connections between the OpCen and the GS are made using a Wireguard VPN point-to-point connection. This is made to keep the connection secure and to not have the public ip of the GS open. The OpCen software accesses the GS software using the GS API and both databases are connected through PgLogical.

11.2 Deployment of the backend

In order to deploy the GS and the OpCen software, an installation script was written. This script contains bash instructions in order to install the program in its corresponding container, as a Linux service, which is started on computer start up.

The database for the OpCen and the GS is also created in its corresponding container using database creation scripts.

After installing the programs, the configuration files were modified in order to match with the corresponding database credentials and antennas configuration parameters. Finally, in the OpCen database, the necessary rows in the *Ground Stations* and *Adapters* table were inserted so the OpCen software could recognize all the GSs in the network.

11.3 Deployment of the OpCen GUI

The OpCen GUI is deployed creating a production build of the software and installing it to a NGINX[2] server. This process is done for each of the GUI, the management and the Viewer one. The url of the OpCen API is also configured for the production build in order to point to the new deployed direction.

The deployed clients can be seen in the Appendix F. They show the satellites, passes and ground stations tabs from the management client and the visualization client. All of them follow the design guidelines that were provided.

[2]**NGINX:** https://www.nginx.com/

12 System Verification

12.1 Software Testing

The GS logic was tested using TDT. They are a variation of unit tests where the programmer establishes a series of test cases which contain an expected input and output. The test function is executed for each of those tests cases in order to ensure that everything is working as expected in all possible scenarios. In Figures 12.1 and 12.2 the results of those tests is shown. The last line of Figure 12.2 shows how all tests passed correctly.

```
=== RUN   Test_getPassesInterval
=== RUN   Test_getPassesInterval/Returns_1_pass
=== RUN   Test_getPassesInterval/Returns_2_passes
=== RUN   Test_getPassesInterval/Returns_no_passes
--- PASS: Test_getPassesInterval (0.12s)
    --- PASS: Test_getPassesInterval/Returns_1_pass (0.00s)
    --- PASS: Test_getPassesInterval/Returns_2_passes (0.00s)
    --- PASS: Test_getPassesInterval/Returns_no_passes (0.00s)
=== RUN   Test_getPassesIntervalSat
=== RUN   Test_getPassesIntervalSat/Returns_1_pass
=== RUN   Test_getPassesIntervalSat/Returns_2_passes
=== RUN   Test_getPassesIntervalSat/Returns_no_passes
--- PASS: Test_getPassesIntervalSat (0.13s)
    --- PASS: Test_getPassesIntervalSat/Returns_1_pass (0.00s)
    --- PASS: Test_getPassesIntervalSat/Returns_2_passes (0.00s)
    --- PASS: Test_getPassesIntervalSat/Returns_no_passes (0.00s)
=== RUN   Test_checkOverlapsWithPasses
=== RUN   Test_checkOverlapsWithPasses/Intervals_overlap
=== RUN   Test_checkOverlapsWithPasses/Different_adapter,_same_interval
--- PASS: Test_checkOverlapsWithPasses (0.12s)
    --- PASS: Test_checkOverlapsWithPasses/Intervals_overlap (0.00s)
    --- PASS: Test_checkOverlapsWithPasses/Different_adapter,_same_interval (0.00s)
=== RUN   Test_DeletePass
=== RUN   Test_DeletePass/existing_pass
=== RUN   Test_DeletePass/non_existing
--- PASS: Test_DeletePass (0.12s)
    --- PASS: Test_DeletePass/existing_pass (0.00s)
    --- PASS: Test_DeletePass/non_existing (0.00s)
=== RUN   Test_getRecordingsID
=== RUN   Test_getRecordingsID/Pass_exists
=== RUN   Test_getRecordingsID/Pass_does_not_exist
--- PASS: Test_getRecordingsID (0.12s)
    --- PASS: Test_getRecordingsID/Pass_exists (0.00s)
    --- PASS: Test_getRecordingsID/Pass_does_not_exist (0.00s)
```

Figure 12.1: TDT Results I

It was also tested at a system level. For each endpoint of the API or functionality of the Task Executor, a test case with the correct scenarios and the error scenarios was defined. Each of this scenario was executed using postman and if there was any error, it was fixed.

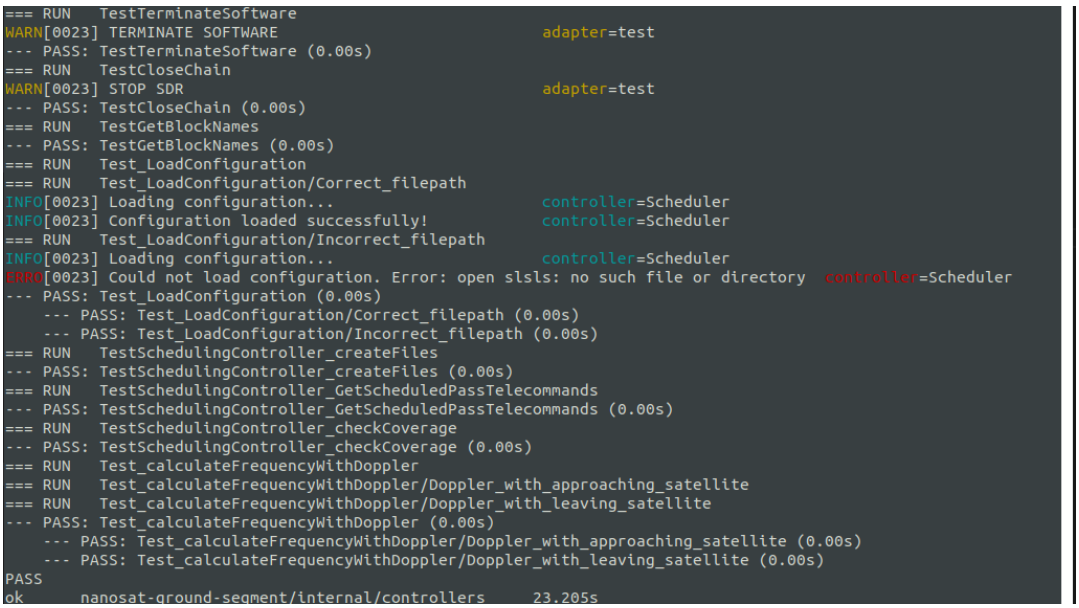


Figure 12.2: TDT Results II

The OpCen Backend was tested directly using System Tests as it does not have any complicate logic behind, apart from making calls to the GS API and accessing the database. Finally, the OpCen GUI was also tested using System Level Tests by testing that each of the implemented functionalities work as expected. The specification of all those system level tests, including the GS and OpCen ones, can be seen in the Appendix D.

12.2 Requirements Verification

As some parts of the system were required to be completed only at a design level and others needed to be implemented, the requirements verification is made using three verification levels.

- **Design Verification:** indicates if the requirement is verified at a design level, meaning that it exists a component in the design that once implemented would cover the required feature.
- **Unit Tests:** tells whether the functions that complete the requirement are tested using unit tests. Unit tests apply for the GS logic, as it is the most complex part of the system.
- **System Test:** verifies that the requirement is met at a system level. The functionality works properly once all the functions are added to the full system. It verifies that the software and installation work at a system level.

The complete verification of the system can be found in the Appendix A. As mentioned in the requirements specification, the ones with *high* priority were the ones needed in order to have a software at an operational level. In the aforementioned appendix is shown how those requirements are met, which leave us with a first stable and production-ready version of the software.

12.3 Non-Functional Requirements Justification

The system has to perform the tasks defined in the objectives and accomplish the defined requirements, but it is also important that it fulfills the non-functional ones, as they are the ones that measure the quality of the solution. In this project there were six different non-functional requirements which were taken into account in the system design.

NF-REQ 1: **Data exchanges should be done through a secure connection.** The *System Communication Design* takes this requirement into account and defines the use of a VPN to connect the GS nodes to the OpCen. Both in the *Selected Technologies* and the *Deployment*, this requirement is taken into account and a VPN technology was selected and deployed to the final version of the system.

NF-REQ 2: **The GS software should be scalable and modular.** As explained in the *Ground Station Design* Chapter, the software is scalable thanks to the facilities it provides by separating the access to the hardware from the rest of the software, making it accessible with the use of a common interface (adapter and mux interfaces). The software is modular as the entry point interface and the access to the database are separate from the logic, and can be replaced without affecting the core.

NF-REQ 3: **The GS software has to be reliable and should be able to recover from errors, such as internet or electricity outage.** The GS software has all the logic and information necessary to meet its schedule during periods when the connection may be lost. Thus, it also has mechanisms to avoid losing generated information, such as saving the recorded data in files. Finally, if there is a power outage, it has mechanisms to resume everything from where it was left.

NF-REQ 4: **There has to be data integrity between the Ground Stations and OpCen.** As explained in the *System Communication Design*, each database table is either written by the GS or the OpCen and the copying mechanisms just go one way (GS to OpCen) so there is never inconsistencies in the data. Files are also generated only in the GSs and copied to the OpCen, obtaining the same results as with the databases.

NF-REQ 5: **The OpCen's software has to be usable, as it is the one managed by real users.** The OpCen software has different GUI so each of the

operators can do its own tasks without interference. Furthermore, the GUI uses simple components and standard interactions so it is easy to pick up for any user, as explained in the *OpCen Design*.

NF-REQ 6: **The OpCen software GUIs have to keep the same look and feel and brand image.** All the GUIs of the OpCen software use the same style of components and a color palette was designed in order to match with the NanoSat Lab logo color.

13 | Sustainability report

13.1 Introduction

The *sustainability matrix* is a way of evaluating the sustainability of a project. The evaluation is made for the three following blocks: the Project Put in Production, the Lifetime and the Risks that the project presents. For each block, the evaluation is made according to an environmental, economic and social point of view. In Table 13.1, the aforementioned matrix is shown. In the next sections, each of the evaluations will be made.

Table 13.1: Sustainability matrix

	PPP	Lifetime	Risks
Environmental	Design consumption	Ecological footprint	Environmental risks
Economic	Initial budget	Viability plan	Economic risks
Social	Personal impact	Social impact	Social risks

13.2 Project Put in Production

13.2.1 Design consumption, Environmental dimension

The project is developed using a low-consumption laptop, which uses far less energy than a tabletop computer. The only negative impact it may have (and this is not improvable), is the electricity and resources cost for the creation and use of the hardware. This negative impact can be easily paid off by the fact that the project will enable the NanoSat Lab to operate with satellites that do Earth Observation tasks, which provide data used for environmental studies.

13.2.2 Initial budget, Economic dimension

The estimated cost of the project can be seen in the Planning and budget section. It includes materials and human resources costs. In order to reduce costs, we could only optimize the number of times required to go to Montsec, which we did by spending some

days there instead of going multiple times. Human resources costs can not be lowered as the project cannot be done in less time.

At the end of the project, the cost had increased a lot because we underestimated the hours needed to implement certain parts of the software, and as mentioned in the Effects on project cost section, the initial budget could have been estimated more accurately if the defined tasks were more granular instead of so generic.

13.2.3 Personal impact, Social dimension

This project has taught me about Space missions and software design for the GSs. I learned how to apply the software design skills learned during the degree to a completely new environment. I have also learned that there is a lot of people willing to help you in the development of projects, both people from inside the laboratory and strangers from internet communities.

13.3 Lifetime

13.3.1 Ecological footprint, Environmental dimension

Before the deployment of my project, the NanoSat Lab used specific programs to communicate with their GS but did not have a defined infrastructure to do so.

During the first year of lifetime of the system, the GS and the OpCen computers need to be maintained. Each computer consumes an average of 42 kWh, taking into account the maximum and minimum usage time. The total Ecological footprint is of 84 kWh on average, which is less than having two people working on a laptop on a full-time job.

The project will have a worse ecological footprint (as the software is running all day), compared to their previous solution, but as a counterpart, it will enable the laboratory to make a better use of their GS, which reflects on getting more scientific data related to Earth Observations missions, which provide information to improve the environment and make a better use of the natural resources. Thus, we can say that, globally, the system will improve the ecological footprint as the deterioration of the ecological footprint is invested in research to improve it way more.

13.3.2 Viability plan, Economic dimension

When NanoSat Lab had a satellite mission they wished to control, they used a simple software that controlled the GS directly in order to download the data but was too simple for future missions. This software was created internally, so it had no external costs rather than the electricity used by the hardware of the GS and the computer used

in the laboratory.

Without my solution, the NanoSat Lab would have had to resort to external services in order to control more complex missions. This external companies would have charged them for each pass they wanted to follow. With my system, the only added cost is the electricity consumed by the OpCen server and the reparation of the computers. Furthermore, they can also rent the GS usage in order to cover costs.

As the software created is modular and reusable it is really easy to expand by adding new GSs that the NanoSat Lab might own in the future. By reusing the software in the new GSs, there will not be an extra cost for developing new software, as this one will still fit. In case of expanding, the costs will be linear to the number of GSs possessed and the electricity/repairs costs of each of them. All this information help society know more about the health of Earth and where measures should be taken in order to improve it.

13.3.3 Social impact, Social dimension

As mentioned above, the result from Earth Observation missions is information that helps improving the environment. With the climate crisis, this information is really important for the society. It also helps preventing fires and contributes to the monitoring of crop fields

The NanoSat Lab has the intention of participating in more Satellite missions in the future, which means that an Operation Center and their Ground Stations will still be of use in order to operate the aforementioned missions. In the short term, they are participating in FSSCat [8] and 3Cat-4 [7], as mentioned in the *Context and Motivation* section. In the mid-term, they are working on the Remote sensing and Interference detector with radiomeTry and vegetation Analysis (RITA). It is one of the Remote Sensing payloads selected by the 2nd GRSS Student Grand Challenge and it will fly inside a 3U CubeSat integrated by the NSSTC (UAE)[1]. Furthermore, with the expansion of CubeSat missions, having a network of GS gives a good investments opportunity.

The project will directly benefit the operators in the NanoSat Lab and everyone who works with the satellites tracked by them, as it will give them a much stronger software to control the GSs and manage satellite's missions operations.

Furthermore, it will also indirectly benefit the scientific community as the downloaded data can be processed and used for scientific studies related to Earth Observation. Society can take actions according to those studies results in order to improve the environment, so it will also benefit everyone indirectly.

[1] **NSSTC: https://www.uaeu.ac.ae/ar/**

The project could not achieve all the proposed objectives, but it offers a first working version that covers all the high priority requirements. From the sixteen proposed objectives, fourteen of them are met (87.5%). The design of all the remaining parts is defined and the architecture is very extensible, making easy to continue the project from where it was left.

13.4 Risks

13.4.1 Environmental risks, Environmental dimension

The energy consumption of the project is linear to the number of GSs that are operating in the network, and the quantity of active time of each of them. The more they are used, the more energy they consume. Although theoretically this could happen, in reality building a GS takes time and money so it is not something that can scale up very quickly in the short and mid term.

13.4.2 Economic risks, Economic dimension

The only risk that could worsen the project viability would be that the technologies used for the project were no longer maintained. During the design of the software, we chose technologies that are widely used and that have a good community support, so the risk of them being unmaintained is low.

13.4.3 Social risks, Social dimension

The project does not have any scenario where it can be harmful for anyone nor create any dependence to the users or to society. It is an internal software used for educational purposes and does not have any social component.

14 | Laws and Regulations

The developed software is primary a control software for space missions and Ground Stations hardware control. Stored data is not personal information nor critical data, so it does not need to be treated in any special manner.

The system depends on some third-party software, such as user interface components or scientific libraries. This softwares include the SGP4 orbit propagator, the Operations Center dashboard template or HTTP handlers implementations amongst others.

All of those technologies have permissive free software licences, including having BSD-2, BSD-3 and MIT, as seen in Table 14.1. Those three licenses allow us to use the software for commercial use, modification, distribution and private use. As a counterpart it does not offer us liability and warranty. In order to use those softwares, there must be a copy of the copyright notice and the clauses mentioned in the BSD-2 and BSD-3 licenses in our software license.

Table 14.1: Techonology licenses' summary

Technology	Description	Licence
Go-satellite	SGP4 library in Golang for orbit propagation	BSD-2
Gorilla-Mux	HTTP request multiplexer (used to create the API endpoints and redirect traffic)	BSD-3
Moment	JavaScript library for parsing, validating, manipulating, and formatting dates.	MIT
Leaflet	JavaScript library for mobile-friendly interactive maps.	BSD-2
Bootstrap-vue	Implementation of Bootstrap v4 components and grid system for Vue.js	MIT
vue-argon-dashboard	Dashboard template with many components	MIT

15 | Conclusions

The main objective of the project was to design an OpCen that accepted a network of multiple GS that shared the same software and data structures. The design of that prototype had to guarantee that all the roles that the operators cover during a CubeSat mission can be performed correctly. There were three main roles, which cover different tasks:

- The *Operation Manager* had to be able to schedule passes on the GS and check its current functioning. This is achieved in the Management client and the Scheduling and Health endpoints in the GS software, where the operator will be able to schedule both passes and health tests.
- The *Uplink Manager* had to be able to plan the commands flow that should be sent during the pass operations, and send those commands to the satellite. This is accomplished in the design of the Uplink client and the Uplink functionalities in the GS, as they have the infrastructure to propagate those commands from the OpCen to the antennas.
- The *Downlink Manager* had to be able to receive the satellite's data which is achieved using a Downlink client which once implemented, will display the data received by the GS Downlink sockets.

Finally, all the personnel will be informed of the operations thanks to the Visualization software that provides them with all the necessary information to know that everything is going as expected, including proof that the antennas are moving, that the satellite is passing over the GS and that we are receiving signal.

Regarding the objectives that required implementation, most of them are achieved. We have an OpCen with access to the current GSs and all the data is replicated to its database using PgLogical replication. The GS can download data from satellites and redirect it to the OpCen although no uplink operations can be performed yet.

Passes can be scheduled from the OpCen and from the GS locally. The GS executes the pass automatically thanks to the Task Executor that reacts to events. This is a very clean solution as it does not consume much CPU when there is no events occurring.

All the adapters of the Montsec GS are implemented, deployed and tested, which means that the GS is fully operative in S-Band and in UHF/VHF.

Another objective that is not completely met, is allowing access to the scheduling functionalities to external companies. It is defined at a design level, that some of the scheduling API endpoints will be public for them to use, but it is not yet implemented as there has not been an agreement yet on how this should be done. However, they can request a schedule using other ways such as an email and then we can send them the recorded files, which in the end, accomplish the objective of scheduling passes for them.

To summarize, there were seven design objectives which were accomplished and nine implementation objectives of which seven were completely completed while the remaining two were partially completed.

Concerning future work, the project has many opportunities to be expanded upon. There is still a lot of things to implement, including the health tests part, the uplink and downlink clients and data processing algorithms. Offering a good interface for external clients can give the project a lot of external visibility and make it profitable for the NanoSat Lab.

In general terms, I am very happy with the result of the project and I think that the NanoSat Lab will find it very useful now that the number of CubeSat missions they have is increasing. I would really like to see all the design implemented and working as I think it is very complete and has a lot of potential.

I am glad that I could do this project because it has helped me consolidate many software design knowledge. Having the opportunity to learn a language like Go, which has so many useful features such as go-routines and channels, has also helped me grow as a developer and learn a lot about concurrency management and implementation. Finally, working in the space sector is something that I always had curiosity about and has made this whole project very enjoyable.

Bibliography

[1] Keyvan Aghababaiyan, Reza Zefreh, and Vahid Shah-Mansouri. 3d-omp and 3d-fomp algorithms for doa estimation. *Physical Communication*, 31:87–95, 10 2018. doi: 10.1016/j.phycom.2018.10.005.

[2] David A. Vallado and Paul Crawford. 2008) "sgp4 orbit determination. In *in Proceedings of AIAA/AAS Astrodynamics Specialist Conference and Exhibit, August 18 - 21*.

[3] Eci coordinates. `https://gssc.esa.int/navipedia/index.php/Conventional_Celestial_Reference_System`. Last accessed: 31/03/2020.

[4] NanoSat Lab. `https://nanosatlab.upc.edu/en/presentation`, . Last accessed: 23/09/2019.

[5] Cubesats. `http://www.cubesat.org/`, . Last accessed: 11/10/2019.

[6] Montsec observatory. `https://www.oadm.ieec.cat`. Last accessed: 20/02/2020.

[7] Nanosat lab 3cat4. `https://nanosatlab.upc.edu/en/missions-and-projects/3cat-4`, . Last accessed: 26/03/2020.

[8] Fsscat. `https://nanosatlab.upc.edu/en/missions-and-projects/fsscat`. Last accessed: 07/04/2020.

[9] Open Cosmos. `https://open-cosmos.com/`. Last accessed: 2/09/2019.

[10] SatNOGS. `https://satnogs.org/`, . Last accessed: 22/09/2019.

[11] Amazon Ground Stations. `https://aws.amazon.com/es/ground-station/`, . Last accessed: 22/09/2019.

[12] Kongsberg Ground Stations. `https://www.kongsberg.com/ksat/services/ground-station-services/`, . Last accessed: 22/09/2019.

[13] Genso. `https://www.esa.int/Education/How_GENSO_works`. Last accessed: 14/10/2019.

[14] Orbit Propagation Models. `https://en.wikipedia.org/wiki/Simplified_perturbations_models`. Last accessed: 11/10/2019.

[15] Celestrak SGP4 canonical implementation. http://www.celestrak.com/NORAD/documentation/spacetrk.pdf. Last accessed: 11/10/2019.

[16] Python SGP4 implementation. https://pypi.org/project/sgp4/, . Last accessed: 11/10/2019.

[17] JavaScript SGP4 implementation. https://github.com/shashwatak/satellite-js. Last accessed: 11/10/2019.

[18] Golang SGP4 implementation. https://github.com/joshuaferrara/go-satellite, . Last accessed: 11/10/2019.

[19] C++ SGP4 implementation. https://www.danrw.com/sgp4/. Last accessed: 11/10/2019.

[20] Chartist-js. https://gionkunz.github.io/chartist-js/. Last accessed: 11/10/2019.

[21] NASA. OpenMCT, 2019. https://nasa.github.io/openmct/.

[22] Ramón Martínez Rodríguez-Osorio. Educational ground station based on software defined radio. http://oa.upm.es/3599/2/INVE_MEM_2008_56070.pdf. Last accessed: 3/10/2019.

[23] Iterative and Incremental development. https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/, . Last accessed: 23/09/2019.

[24] Technical writer salary. https://www.glassdoor.es/Sueldos/espa~na-technical-writer-sueldo-SRCH_IL.0,6_IN219_KO7,23.htm. Last accessed: 3/10/2019.

[25] Devops salary. https://www.glassdoor.es/Sueldos/barcelona-devops-sueldo-SRCH_IL.0,9_IM1015_KO10,16.htm. Last accessed: 3/10/2019.

[26] Programmer salary. https://www.glassdoor.es/Sueldos/barcelona-programmer-sueldo-SRCH_IL.0,9_IM1015_KO10,20.htm. Last accessed: 3/10/2019.

[27] System analyst salary. https://www.glassdoor.es/Sueldos/espa~na-system-analyst-sueldo-SRCH_IL.0,6_IN219_KO7,21.htm. Last accessed: 3/10/2019.

[28] Tester salary. https://www.glassdoor.es/Sueldos/espa~na-tester-sueldo-SRCH_IL.0,6_IN219_KO7,13.htm. Last accessed: 3/10/2019.

[29] Software architect salary. https://www.glassdoor.es/Sueldos/espa~na-software-architect-sueldo-SRCH_IL.0,6_IN219_KO7,25.htm. Last accessed: 3/10/2019.

[30] Relational Databases. https://www.ibm.com/cloud/learn/relational-databases/. Last accessed: 18/12/2019.

[31] Virtual private networks. https://courses.lumenlearning.com/wm-introductiontobusiness/chapter/virtual-private-networks-vpns/. Last accessed: 07/04/2020.

[32] gobyexample goroutines. https://gobyexample.com/goroutines. Last accessed: 07/04/2020.

[33] Nicolas Dilley and Julien Lange. Concurrency in go. https://www.cs.kent.ac.uk/people/staff/jl703/papers/dilley-lange-saner19-preprint.pdf. Last accessed: 17/10/2019.

[34] Golang postgresql. https://github.com/lib/pq. Last accessed: 17/10/2019.

[35] Python postgresql. http://initd.org/psycopg, . Last accessed: 17/10/2019.

[36] Golang OS. https://golang.org/pkg/os/, . Last accessed: 17/10/2019.

[37] Python OS. https://docs.python.org/3/library/os.html, . Last accessed: 17/10/2019.

[38] Go Code Coverage. https://golang.org/cmd/cover/. Last accessed: 17/10/2019.

[39] Writing table driven tests in go. https://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go. Last accessed: 17/10/2019.

[40] Python tdt. https://docs.pytest.org/en/latest/, . Last accessed: 17/10/2019.

[41] Python Code Coverage. https://coverage.readthedocs.io/en/v4.5.x/, . Last accessed: 17/10/2019.

[42] Postgres vs MySQL. https://www.2ndquadrant.com/es/postgresql/postgresql-vs-mysql. Last accessed: 21/03/2020.

[43] Acid. https://es.wikipedia.org/wiki/ACID. Last accessed: 22/03/2020.

[44] WireGuard. Wireguard VPN, 2019. https://www.wireguard.com/.

[45] Ground Segment. https://en.wikipedia.org/wiki/Ground_segment, . Last accessed: 23/09/2019.

[46] NASA. How do satellites communicate. https://www.nasa.gov/directorates/heo/scan/communications/outreach/funfacts/txt_satellite_comm.html. Last accessed: 19/09/2019.

[47] Iterative and Incremental development diagram. https://en.wikipedia.org/wiki/Iterative_and_incremental_development. Last accessed: 23/09/2019.

[48] Python vs go. https://www.educba.com/python-vs-go/, . Last accessed: 17/10/2019.

[49] Go vs python. https://dzone.com/articles/golang-vs-python-which-one-to-choose. Last accessed: 17/10/2019.

[50] Golang timeouts. https://medium.com/@simonfrey/go-as-in-golang-standard-net-http-config-will-break-your-production-environment-1360871cb72b, . Last accessed: 17/10/2019.

[51] Python inheritance. https://www.w3schools.com/python/python_inheritance.asp, . Last accessed: 17/10/2019.

A System Requirements

A.1 GS Requirements

API-HC-001: API Health Checks	
Description	The operator shall be able to assign predefined tests to the GS schedule
Verification	Adding the tests to the testsConfig file ensures that it contains the new test
Priority	Medium

API-HC-002: API Health Checks	
Description	The operator shall be able to execute predefined tests
Verification	Try to execute some of the predefined tests
Priority	Medium

API-HC-004: API Health Checks	
Description	The API shall provide the operator with a test identificator after a test is scheduled
Verification	Receiving a testID that can be later consulted after scheduling or executing a tests
Priority	Medium

API-HC-005: API Health Checks	
Description	The operator should be able to get a report of the executed test through the API
Verification	Executing a tests and accessing its report later shall give a report file in the correct format
Priority	Medium

API-HC-006: API Health Checks	
Description	The operator shall be able to consult available tests of a concrete GS
Verification	Calling the get available tests function shall return a tests file in the correct format
Priority	Medium

API-SC-001: API Scheduling	
Description	The API shall register a pass automatically when given a TLE file and a configuration file
Verification	Sending a TLE+config file should register pass with the given ID which can be consultable
Priority	High

API-SC-002: API Scheduling	
Description	The operator shall obtain a pass identificator after registering a pass
Verification	Test with API-SC-001
Priority	High

API-SC-003: API Scheduling	
Description	The operator shall be able to edit and upload a pass configuration or its TLE file
Verification	Try uploading TLE and config file and check that the API receives it. Check that the database has the new file uploaded.
Priority	Medium

API-SC-004: API Scheduling	
Description	The operator shall be able to cancel a pass indicating its pass ID
Verification	Calling the delete pass function with a correct ID shall delete it from the DB
Priority	High

API-SC-005: API Scheduling	
Description	The operator should be able to consult the schedules
Verification	Schedules data should be available on the local database or replicated to the OpCen
Priority	High

GS-AST-001: GS Automated Satellite Tracking	
Description	The GS shall propagate orbit from a TLE file for the next pass
Verification	Check with GS-AST-003
Priority	High

GS-AST-002: GS Automated Satellite Tracking	
Description	The adapter shall know how to correct orbits according to the specific antenna
Verification	Check with GS-AST-003
Priority	High

GS-AST-003: GS Automated Satellite Tracking	
Description	The adapter shall calculate commands to follow the orbit
Verification	All calculated antenna movements are physically viable
Priority	High

GS-AST-004: GS Automated Satellite Tracking	
Description	The adapter shall execute antenna movements commands at the specified timestamp
Verification	The antenna follows the specified orbit at the specified times
Priority	High

GS-AST-005: GS Automated Satellite Tracking	
Description	The adapter shall execute requested telecommands at the specified timestamp
Verification	All telecommands are transmitted on time
Priority	Low

GS-AST-006: GS Automated Satellite Tracking	
Description	The adapter shall send commands to move the antenna to the starting point before the pass
Verification	All commands are send and the antennas move to the expected positions
Priority	High

GS-AST-007: GS Automated Satellite Tracking	
Description	The adapter shall execute prepass configuration automatically
Verification	All and only the specified configuration and GNURadio blocks are initiallized before a pass and this configuration does not take longer than expected
Priority	High

GS-AST-008: GS Automated Satellite Tracking	
Description	The adapter shall execute postpass configuration automatically
Verification	The post configuration starts right after the pass and executes everything specified
Priority	High

GS-DA-001: GS Data Access	
Description	Have a DB user to access it locally
Verification	Being able to login using the defined user from the GS
Priority	High

GS-GSA-001: GS GS Control	
Description	The operator should be able to acces the API locally
Verification	Having acces to all API functionalities from the GS Terminal
Priority	High

GS-HW-001: GS Hardware	
Description	The adapter shall move antennas through software
Verification	Running a predefined test or executing a test software that talks to the adapter functions makes the antennas move to the specified positions
Priority	High

GS-HW-002: GS Hardware	
Description	The software should be able to send packets to be transmitted through GNU Radio
Verification	Packets at the uplink api and at the GNU Radio blocks are equivalent and are not lost
Priority	Medium

GS-MD-001: GS Modular Design	
Description	The operator shall be able to access the GS via an API with predefined commands
Verification	Having access to all API functionalities of each GS
Priority	High

GS-MD-002: GS Modular Design	
Description	API shall make calls to the controllers and the executing tasks
Verification	Check the API code that it communicates with the task executor correctly
Priority	High

GS-MD-004: GS Modular Design	
Description	Database structure shall be shared amongst GSs
Verification	Having the same DB structure in all GSs
Priority	High

GS-MD-005: GS Modular Design	
Description	Predefined adapter shall be implemented specifically for each GS
Verification	Code: having different implementations for each configuration
Priority	High

GS-RFC-001: GS RF Chain	
Description	RF Chain shall be capable of getting raw data
Verification	Checking the output should show raw data
Priority	High

GS-RFC-002: GS RF Chain	
Description	RF Chain shall be capable of demodulating data if modulation is known
Verification	Checking the output should show demodulated data for a known modulation
Priority	High

GS-RFC-003: GS RF Chain	
Description	RF Chain shall be capable of decoding data if codification is known
Verification	Checking the output should show decoded data for a known coding
Priority	Low

GS-RFC-004: GS RF Chain	
Description	RF Chain shall be capable of saving collected data to data files
Verification	Being able to access the files and checking they contain expected data
Priority	High

GS-RFC-005: GS RF Chain	
Description	RF Chain shall be capable of streaming collected and processed data
Verification	Tested with DF-DD-002
Priority	Medium

A.2 OpCen Requirements

OPC-DA-001: OpCen Data Access	
Description	The OpCen Operator should be able to acces raw saved mission data
Verification	Data files should be accessible after the pass
Priority	High

OPC-DA-002: OpCen Data Access	
Description	The OpCen Operator should be able to acces demodulated saved mission data
Verification	Data files should be accessible after the pass
Priority	High

OPC-DA-003: OpCen Data Access	
Description	The OpCen Operator should be able to acces decoded saved mission data if decoding is known
Verification	Data files should be accessible after the pass
Priority	Medium

OPC-DA-004: OpCen Data Access	
Description	The OpCen Operator should be able to acces past health tests reports of a concrete GS
Verification	Reports files should be accessible after the test
Priority	Low

OPC-GSC-002: OpCen GS Control	
Description	The operator should have access to the GS's cameras
Verification	Being able to access the cameras when managing the GS
Priority	High

OPC-GSN-001: OpCen GS Network	
Description	An authorized operator should be able to add a new GS to the network
Verification	Adding a GS grants access to its API functionalities
Priority	High

OPC-SC-002: OpCen Scheduling	
Description	The OpCen Software shall distribute GS's coverage correctly when scheduling a pass
Verification	Tested with OPC-SC-001
Priority	Low

OPC-SC-003: OpCen Scheduling	
Description	The OpCen Software shall be capable of scheduling one or more passes according to the number of passes or time interval specified by the operator
Verification	Being able to schedule one or multiple passes according to the available options.
Priority	High

OPC-DA-001: OpCen Data Access	
Description	Have a DB user to access it locally
Verification	Being able to login using the defined user from the OpCen
Priority	High

OPC-OPCA-001: OpCen OpCen Control	
Description	The operator should be able to acces the API locally
Verification	Having acces to all API functionalities from the OpCen Terminal
Priority	High

OPC-GUI-001: OpCen GUI Management	
Description	The OpCen should have a User Interface to see all the Ground Stations information
Verification	Having access to all the GS in the network information and its chains
Priority	High

OPC-GUI-002: OpCen GUI Management	
Description	The OpCen should have a User Interface to schedule passes
Verification	Being able to schedule a pass using a user interface and see which ones are scheduled
Priority	High

OPC-GUI-003: OpCen GUI Management	
Description	The OpCen should have a User Interface to schedule tests
Verification	Being able to schedule a test using a user interface and see which ones are scheduled
Priority	Medium

OPC-GUI-005: OpCen GUI Management	
Description	The OpCen should have a User Interface to communicate effectively with the satellite during a pass
Verification	Having access to all the functionalities requested in the Uploading telecommands section
Priority	Low

OPC-GUI-006: OpCen GUI View	
Description	The data retrieved from the satellite should be displayed visually
Verification	Displaying recieved data in a user interface
Priority	High

OPC-UP-001: OpCen Uploading telecommands	
Description	An operator shall see the available commands for a satellite when planning the uplink tasks
Verification	The OpCen should store the commands a satellite can recieve in a database and display them to the operator's GUI
Priority	Low

OPC-UP-002: OpCen Uploading telecommands	
Description	The Operator shall be able to see the telecommands sent during a pass
Verification	The OpCen keeps track of the sent telecommands storing them in a database and displays it to the GUI so the operator can see them
Priority	Low

OPC-UP-003: OpCen Uploading telecommands	
Description	The Operator shall be able to send telecommands during a pass
Verification	The OpCen has to send the telecommand to the necessary GS API
Priority	Low

OPC-UP-004: OpCen Uploading telecommands	
Description	The operator shall be able to add telecomands to be sent during the pass
Verification	Sending a telecommands file in the correct format shall add it to the pass entry in the DB
Priority	Medium

OPC-UP-005: OpCen Uploading telecommands	
Description	The operator shall be able to edit and upload the telecomands of a pass
Verification	Sending a telecommands file in the correct format shall rewrite the pass entry in the DB with the new telecommands
Priority	Medium

OPC-UP-006: OpCen Uploading telecommands	
Description	The operator shall be able to delete the predefined telecomands of a pass
Verification	Deleting telecommands giving its passID deletes the DB entry
Priority	Medium

A.3 System Communication Requirements

DF-DC-001: Data flow Data Sync	
Description	Scheduling data shall be replicated automatically from the GS to the OpCen
Verification	OpCen should include all defined GSs sync data
Priority	High

DF-DC-002: Data flow Data Sync	
Description	Pass recordings shall be replicated from the GS to the OpCen
Verification	OpCen should include all defined GSs sync data
Priority	High

DF-DC-003: Data flow Data Sync	
Description	Logs and statistics shall be replicated from the GS to the OpCen
Verification	OpCen should include all defined GSs sync data
Priority	High

DF-DC-004: Data flow Data Sync	
Description	GS data should be erased periodically
Verification	All data erased should be previously replicated to the OpCen and erasing should be done at the specified time
Priority	Low

DF-DD-001: Data flow Downloading data	
Description	The GS software should redirect satellite data in real time from the GS to the operation center
Verification	When in a pass, satellite data should be received at the OpCen
Priority	High

DF-DD-002: Data flow Downloading data	
Description	The OpCen should get sockets to download data during the pass
Verification	Having available sockets and being able to connect to them and receive data through them
Priority	Medium

DF-DD-003: Data flow Downloading data	
Description	The OpCen should receive in real time raw data via socket
Verification	Data received is raw
Priority	Medium

DF-DD-004: Data flow Downloading data	
Description	The OpCen should receive in real time demodulated data via socket if modulation is known
Verification	Data received is demodulated if demodulation is known, no data otherwise
Priority	Low

DF-DD-005: Data flow Downloading data	
Description	The OpCen should receive in real time decoded data via socket if codification is known
Verification	Data received is decoded if decoding is known, no data otherwise
Priority	Low

DF-UT-001: Data flow Uploading telecommands	
Description	Authorized operators should have acces to the pass socket in order to commicate with the satellite
Verification	As an authorized user, you shall be able to access the socket and check you can send data through it
Priority	Medium

DF-UT-002: Data flow Uploading telecommands	
Description	Authorized operators should be able to send telecommands manually when there are no scheduled telecommands
Verification	Sending commands makes the adapter send them as soon as it receives them
Priority	Medium

DF-UT-003: Data flow Uploading telecommands	
Description	Authorized operators should be able to interrupt scheduled telecommands in order to send them manually
Verification	Sending commands when there are scheduled commands, stops the scheduled commands transference and sends the manual ones as soon as they are received
Priority	Medium

DF-UT-004: Data flow Uploading telecommands	
Description	The OpCen shall validate telecommands for the known telemetry passes
Verification	Sending an incorrect command shall cause a vallidation failure and sending a correct command shall pass the validation filter
Priority	Medium

DF-UT-005: Data flow Uploading telecommands	
Description	The operator shall get status feedback during the communication from the GS
Verification	Status files shall be received periodically in the correct format
Priority	High

N-AIN-001: Networking Acces to internal Network	
Description	Acces to the NanoSat internal network must be done through an external API
Verification	There should not be any other way to access the NanoSat internal network from the outside
Priority	Low

N-AIN-002: Networking Acces to internal Network	
Description	External operators shall access through a proxy located between external and internal network traffic
Verification	Design, checking that there is a configured proxy and testing that the traffic goes through it
Priority	Low

N-AIN-003: Networking Acces to internal Network	
Description	External API must be only accessible to authorised users, handled by the proxy
Verification	Trying to access from an unauthorized user shall not be possible
Priority	Low

N-AIN-004: Networking Acces to internal Network	
Description	The communication with the outside has to be able to be shot down
Verification	Not being able to connect when the connection is closed
Priority	Low

N-AIN-005: Networking GS-OpCen communication	
Description	The OpCen shall connect with the ground stations via WireGuard VPN, using Noise protocol framework, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF, and secure trusted constructions
Verification	Having wireguard initialized makes possible the connection with the GS, not having it initialized grants no access to the GSs API
Priority	High

N-AIN-006: Networking GS-OpCen communication	
Description	The GS shall grant acces to the uplink and downlink sockets for authorised users
Verification	Coding, creations of sockets at the specified cases
Priority	Low

B | GS API Documentation

GS API

Scheduling endpoints

Register Pass

Description: Schedule a pass for the given satellite that fits in the given time interval if the given configuration (frequencies, bandwidth and gain) are supported for any of the chains in the ground station.

Method: POST

Path: /pass/request

Body:

RegisterPass.json

```
{
  "NORAD": "25544",
  "TLE1": "1 25544U 98067A   20090.85749228  .00001576  00000-0
36869-4 0  9991",
  "TLE2": "2 25544  51.6462  16.0529 0005034  72.5825  33.5339
15.48932713219876",
  "DlFreq": 244000000,
  "UlFreq": 244000000,
  "Gain": 5,
  "BandWidth": 250000,
  "TStart": 2020-03-31T14:52:13+00:00,
  "TEnd": 2020-03-31T15:02:13+00:00
}
```

RegisterPassWithNow.json

```
{
  "NORAD": "25544",
  "TLE1": "1 25544U 98067A   20090.85749228  .00001576  00000-0
36869-4 0  9991"
  "TLE2": "2 25544  51.6462  16.0529 0005034  72.5825  33.5339
15.48932713219876",
  "DlFreq": 244000000,
  "UlFreq": 244000000,
  "Gain": 5,
  "BandWidth": 250000,
  "TStart": "now",
  "TEnd": 2020-03-31T15:02:13+00:00
```

```
}
```

Answers:

- 200 OK

answerOKExample.json

```
{
  "Passid": 12,
  "StartTime": 2020-03-31T14:52:13+00:00,
  "EndTime": 2020-03-31T15:02:13+00:00,
  "MaxEl": 52.4,
  "AOS": 2020-03-31T14:55:13+00:00,
  "LOS": 2020-03-31T15:00:13+00:00
}
```

- 400 Bad Request

badRequestExample.json

```
{
  "error_tle1":"Error decoding the data, check the format of TLE"
}
```

Example:

example.sh

```
curl -d @RegisterPass.json -X POST http://X.X.X.X:YYYY/pass/request
```

Edit Pass

Description: Edits pass configuration parameters if possible for an already created pass.

Method: POST

Path: /pass/{id}

Body:

EditPass.json

```
{
```

```
  "NORAD": "25544",
  "TLE1": "1 25544U 98067A   20090.85749228  .00001576  00000-0
36869-4 0  9991"
  "TLE2": "2 25544  51.6462  16.0529 0005034  72.5825  33.5339
15.48932713219876",
  "DlFreq": 244000000,
  "UlFreq": 244000000,
  "Gain": 5,
  "BandWidth": 250000,
  "TStart": 2020-03-31T14:52:13+00:00,
  "TEnd": 2020-03-31T15:02:13+00:00,
}
```

Answers:

- 200 OK

answerOKExample.json

```
{
  "Passid": 12
  "StartTime": 2020-03-31T14:52:13+00:00,
  "EndTime": 2020-03-31T15:02:13+00:00,
  "MaxEl": 52.4,
  "AOS": 2020-03-31T14:55:13+00:00,
  "LOS": 2020-03-31T15:00:13+00:00
}
```

- 400 Bad Request

badRequestExample.json

```
{
  "error_tle1":"Error decoding the data, check the format of TLE"
}
```

Example:

example.sh

```
curl -d @EditPass.json -X POST http://X.X.X.X:YYYY/pass/12
```

Delete Pass

Description: Deletes an existing pass identified by the ID parameter.

Method: DELETE

Path: /pass/{id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"Pass not found"
}
```

Example:

example.sh

```
curl  -X DELETE http://X.X.X.X:YYYY/pass/12
```

Management endpoints

Manually Start Chain

Description: Starts all the hardware and software necessary to control the requested chain.

Method: POST

Path: /maintenance/manual/rotor/{id}/init

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl  -X POST http://X.X.X.X:YYYY/maintenance/manual/rotor/1/init
```

Manually Terminate Chain

Description: Stops all the hardware and software necessary to control the requested chain.

Method: POST

Path: /maintenance/manual/rotor/{id}/terminate

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl  -X POST http://X.X.X.X:YYYY/maintenance/manual/rotor/1/terminate
```

Manually Move Antennas

Description: Given a valid Azimuth and Elevation, tries to move the requested antenna to that position.

Method: POST

Path: /maintenance/manual/rotor/{id}/move

Body:

MoveAntennas.json

```
{
  "Az":120.0,
  "El":12.0
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -d @MoveAntennas.json -X POST
http://X.X.X.X:YYYY/maintenance/manual/rotor/1/move
```

Manually Move Antenna to Parking Position

Description: Moves the requested antenna to its parking position, as defined in the antenna configuration file.

Method: POST

Path: /maintenance/manual/rotor/{id}/park

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/maintenance/manual/rotor/1/park
```

Manually Stop Antennas Movement

Description: Stops the movement of the requested antenna if any.

Method: POST

Path: /maintenance/manual/rotor/{id}/stop

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/maintenance/manual/rotor/1/stop
```

Get Antenna Position

Description: Returns the antenna position in Azimuth and Elevation.

Method: GET

Path: /maintenance/manual/rotor/{id}/getposition

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "az":120.0,
  "el":12.0
}
```

- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/maintenance/manual/rotor/1/getposition
```

Set Transmision Mode (Tx)

Description: Sets the tranceivers of the requested chain to transmission mode.

Method: POST

Path: /maintenance/manual/rf/{id}/setTx

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/maintenance/manual/rf/1/setTx
```

Set Transmision Mode (Rx)

Description: Sets the tranceivers of the requested chain to reception mode.

Method: POST

Path: /maintenance/manual/rf/{id}/setRx

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no chain with the specified ID"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/maintenance/manual/rf/1/setRx
```

Health Tests endpoints

Get Health Tests

Description: Returns a list of all the health tests available for that gs.

Method: GET

Path: /healthtests

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "sband": ["moveantennas", "testtransmission"],
  "uvhf": ["moveantennas"]
}
```

</code>

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/healthtests/
```

Run Test

Description: Runs a concrete health test on the specified adapter.

Method: POST

Path: /healthtests/execute/{gsid}/{adapterid}/{testname}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "testID": 14,
}
```

</code>

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The test does not exist"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/healthtests/execute/1/1/moveantennas
```

Delete Health Test

Description: Deletes the health test event if it has not started.

Method: DELETE

Path: test/delete/{testID}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"There is no future test with the requested ID"
}
```

Example:

example.sh

```
curl -X DELETE http://X.X.X.X:YYYYtest/delete/14
```

Schedule Health Test

Description: Schedules one of the supported health test to a chain, in order to be performed at a certain time. The request returns the assigned ID in order to retrieve the results later.

Method: POST

Path: /test/schedule

Body:

ScheduleHealthTest.json

```
{
  "TestName": "moveantennas",
  "AdapterID": 1,
  "TStart": 2020-03-31T14:52:13+00:00
}
```

Answers:

- 200 OK

answerOKExample.json

```
{
  "testID": 12
}
```

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The adapter/test does not exist"
}
```

Example:

example.sh

```
curl -d @ScheduleHealthTest.json -X POST
http://X.X.X.X:YYYY/test/schedule
```

Get Test Results

Description: Returns the results of a health test if it has finished. The results format still needs to be defined and may vary amongst tests.

Method: GET

Path: /test/data/{testID}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "testID": 12,
  "testStatus": "finished",
  "testResults":{...}
}
```

answerOKExample2.json

```
{
  "testID": 14,
  "testStatus": "cancelled",
  "reason": "overlapping interval with a pass"
```

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The test has not finished its execution yet"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/test/data/14
```

Uplink endpoints

Send Telecommand

Description: Puts the chain in charge of the specified pass in transmission mode and transmits the command to the satellite. Once transmitted, the chain is put in reception mode again. If the pass is not happening at the time or it does not exist, it will return an error.

Method: POST

Path: /pass/tc/{passID}

Body:

Telecommand.json

```
{
  "telecommand": "hello"
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"The pass does not exist / is not happening yet"
}
```

Example:

example.sh

```
curl -d @Telecommand.json -X POST http://X.X.X.X:YYYY/pass/tc/14
```

Downlink endpoints

Get Pass Socket

Description: Returns the connection parameters to the socket from which the satellite's downloaded data is sent in real time during a pass. Only works when the (pre)pass has started.

Method: POST

Path: /pass/tc/{passID}

Body:

Telecommand.json

```
{
  "telecommand": "hello"
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"The pass does not exist / is not happening yet"
}
```

Example:

example.sh

```
curl -d @Telecommand.json -X POST http://X.X.X.X:YYYY/pass/tc/14
```

C | OpCen API Documentation

Operation Center API

Ground Stations endpoints

Get Ground Stations

Description: Returns a list of the Ground Stations in the network with its summarized information.

Method: GET

Path: /groundstations

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  [
    {
    "id": 1,
    "name": "Montsec",
    "lat": 14,
    "lon": 2,
    "height": 1300,
    "bands": ["UHF", "VHF", "S-Band"],
    "imgurl": "10.52.72.52:8000"
    },
    {...}
  ]
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/groundstations
```

Get Ground Station Detailed

Description: Returns all the information about a concrete ground station, including the short summary returned in the *Get Ground Stations* request, the connection parameters to the ground station API and the information of the available adapters.

Method: GET

Path: /groundstation/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "basic": {
    "id": 1,
    "name": "Montsec",
    "lat": 14,
    "lon": 2,
    "height": 1300,
    "bands": ["UHF", "VHF", "S-Band"],
    "imgurl": "10.52.72.52:8000"
  },
  "adapters": {
    [
      {
        "id": 1,
        "name": "S-Band"
      },
      {...}
    ]
  },
  "ip": "10.52.72.7",
  "port": 8000
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/groundstation/1
```

Move Antennas

Description: Move the antennas of the requested adapter to the position indicated in Azimuth and Elevation

Method: POST

Path: /groundstation/{gsid}/adapter/{adapterid}/move

Body:

MoveAntennas.json

```
{
  "az": 120.2,
  "el": 50
}
```

Answers:

- 200 OK
- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -d @MoveAntennas.json -X POST
http://X.X.X.X:YYYY/groundstation/1/adapter/1/move
```

Stop Antennas

Description: Stops the movement of the requested antenna if any.

Method: POST

Path: /groundstation/{gsid}/adapter/{adapterid}/stop

Body: -

Answers:

- 200 OK
- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/groundstation/1/adapter/1/stop
```

Get Antennas Position

Description: Retrieve the position of the requested antenna in Azimuth and Elevation

Method: GET

Path: /groundstation/{gsid}/adapter/{adapterid}/get

Body: -

Answers:

- 200 OK

answerOK.json

```
{
  "az": 120.2,
  "el": 50
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/groundstation/1/adapter/1/get
```

Get Antennas Camera URL

Description: Get the URL to the camera pointing to the requested antenna

Method: GET

Path: /groundstation/{gsid}/adapter/{adapterid}/camera

Body: -

Answers:

- 200 OK

answerOK.json

```
{
  "url":"10.52.25.62:2514"
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/groundstation/1/adapter/1/camera
```

Satellites endpoints

Get Satellites

Description: Returns a list of the Satellites stored in the database with its summarized information.

Method: GET

Path: /satellites

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  [
    {
    "noradid": "25544",
    "satname": "ISS",
    "ulfreq": 123000000,
    "dlfreq": 123000000,
    "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
    "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
    "favourite": true,
    "autoupdate": true,
    "bandwidth": 123000,
    "gain": 5
    },
    {...}
  ]
```

```
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/satellites
```

Toggle AutoUpdate

Description: Inverts the *autoupdate from Celestrak* property for the satellite identified with ID = noradID.

Method: GET

Path: /satellite/autoupdate/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "autoupdate": true
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
```

```
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/satellite/autoupdate/28533
```

Toggle Favourite

Description: Inverts the *favourite* property for the satellite identified with ID = noradID.

Method: GET

Path: /satellite/favourite/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "favourite": false
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/satellite/favourite/28533
```

Update Satellite

Description: Rewrites the satellite's information with the one provided in the request.

Method: POST

Path: /satellite/{id}

Body:

UpdateSatellite.json

```
{
  "noradid": "25544",
  "satname": "ISS",
  "ulfreq": 123000000,
  "dlfreq": 123000000,
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "favourite": true,
  "autoupdate": true,
  "bandwidth": 123000,
  "gain": 5
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the tle format"
}
```

Example:

example.sh

```
curl -d @UpdateSatellite.json -X POST
http://X.X.X.X:YYYY/satellite/25866
```

Add Satellite

Description: Creates a new satellite and stores it in the database.

Method: POST

Path: /satellite

Body:

AddSatellite.json

```
{
  "noradid": "25544",
  "satname": "ISS",
  "ulfreq": 123000000,
  "dlfreq": 123000000,
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "favourite": true,
  "autoupdate": true,
  "bandwidth": 123000,
  "gain": 5
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the tle format"
}
```

Example:

example.sh

```
curl -d @AddSatellite.json -X POST http://X.X.X.X:YYYY/satellite
```

Delete Satellite

Description: Deletes the satellite's indentified with noradID information.

Method: DELETE

Path: /satellite/{noradID}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the tle format"
}
```

Example:

example.sh

```
curl -X DELETE http://X.X.X.X:YYYY/satellite/25463
```

Passes endpoints

Get All Passes

Description: Returns a list of all the passes scheduled in the operation center.

Method: GET

Path: /passes

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  [
    {
      "Norad": "25682",
      "SatName": "satname"
      "Passid": 12,
      "StartTime": 2020-03-31T14:52:13+00:00,
      "EndTime": 2020-03-31T15:02:13+00:00,
      "MaxEl": 52.4,
      "AOS": 2020-03-31T14:55:13+00:00,
      "LOS": 2020-03-31T15:00:13+00:00
    },
    {...}
  ]
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes
```

Get All Future Passes

Description: Returns a list of all the passes scheduled in the operation center that are not yet complete.

Method: GET

Path: /passes/future

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  [
    {
      "Norad": "25682",
      "SatName": "satname"
      "Passid": 12,
      "StartTime": 2020-03-31T14:52:13+00:00,
      "EndTime": 2020-03-31T15:02:13+00:00,
      "MaxEl": 52.4,
      "AOS": 2020-03-31T14:55:13+00:00,
      "LOS": 2020-03-31T15:00:13+00:00
    },
    {...}
  ]
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes/future
```

Get Next Pass Time

Description: Returns the time at which the next pass starts.

Method: GET

Path: /passes/next

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "time": "2020-03-31T15:00:13+00:00"
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes/next
```

Schedule Passes

Description: Schedules passes according to the different modes:

- Schedule a single pass: Schedules the next pass available (with no occupation in the ground station)
- Schedule N passes: Schedules the next N passes available
- Schedule all passes until a concrete date: Schedules all available passes until the given date.
- Schedule N passes from start time: Schedules the next N passes available starting from a concrete date.
- Schedule passes in a time interval: Schedules all available passes for the given time interval.

As it might take some time to schedule many passes, the request returns an identifier which can be used to check the status of the processing of the request, explained in the next endpoint.

Method: POST

Path: /passes/schedule

Body: Schedule one pass:

OnePass.json

```
{
  "noradid": "25633",
```

```
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "ulfreq":123000000,
  "dlfreq":123000000,
  "bw": 123000,
  "gain":5,
  "mode": 1,
  "cont": 1
}
```

Schedule N passes:

MultiplePasses.json

```
{
  "noradid": "25633",
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "ulfreq":123000000,
  "dlfreq":123000000,
  "bw": 123000,
  "gain":5,
  "mode": 1,
  "cont": 5
}
```

Schedule all until date:

UntilDate.json

```
{
  "noradid": "25633",
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "ulfreq":123000000,
  "dlfreq":123000000,
  "bw": 123000,
  "gain":5,
  "mode": 2,
  "tend": "2006-01-02 15:04"
}
```

Schedule N passes from start time:

Interval.json


```
{
  "noradid": "25633",
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "ulfreq":123000000,
  "dlfreq":123000000,
  "bw": 123000,
  "gain":5,
  "mode": 4,
  "tstart": "2006-01-02 15:04",
  "cont": 10
}
```


Schedule passes in a time interval:

Interval.json


```
{
  "noradid": "25633",
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "ulfreq":123000000,
  "dlfreq":123000000,
  "bw": 123000,
  "gain":5,
  "mode": 3,
  "tstart": "2006-01-02 15:04",
  "tend": "2006-01-02 15:04"
}
```


Answers:

- 200 OK

answerOKExample.json


```
{
  "requestid": 14
}
```


- 400 Bad Request

badRequestExample.json

```
{
  "error":"the chosen scheduling mode does not patch the given
parameters"
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -d @interval.json -X POST http://X.X.X.X:YYYY/passes/schedule
```

Check scheduling request status

Description: Checks the status of a scheduling request. If it has already finished, it returns the results.

Method: GET

Path: /passes/schedule/status/{id}

Body: -

Answers:

- 200 OK

answerOK.json

```
{
  "passes": [
    {
      "pid": 14,
      "tstart": "01-02-2006 15:04",
      "tend": "01-02-2006 15:04"
    },
```

```
    {...}
  ],
  "pinit": "0",
  "pend": "10",
  "pcurrent": "10"
}
```

- 206 Partial Content: Passes have not already been scheduled

partialContent.json

```
{
  "passes": [],
  "pinit": "0",
  "pend": "10",
  "pcurrent": "5"
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"Error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes/schedule/status/14
```

Get Pass

Description: Returns the full information of a pass. If that pass crosses multiple ground stations, all the information of each of them is also displayed here.

Method: GET

Path: /passes/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "Norad": "25682",
  "SatName": "satname",
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "dlfreq": 123000000,
  "ulfreq": 123000000,
  "bandwidth": 123000,
  "start": "2020-03-31T15:00:13+00:00",
  "end": "2020-03-31T15:00:13+00:00"
  "gspass": [
    {
      "gsid": 1,
      "assignedadapter": 1,
      "gain": 5,
      "recordingsid": 4
      "Passid": 12,
      "StartTime": 2020-03-31T14:52:13+00:00,
      "EndTime": 2020-03-31T15:02:13+00:00,
      "MaxEl": 52.4,
      "AOS": 2020-03-31T14:55:13+00:00,
      "LOS": 2020-03-31T15:00:13+00:00
    },
    {...}
  ]
}
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes/14
```

Edit Pass

Description: Edits pass configuration parameters if possible for an already created pass. It affects to all the ground stations detailed passes that the satellite may cross during that pass.

Method: POST

Path: /passes/edit/{id}

Body:

EditPass.json

```
{
  "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
  "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
  "dlfreq": 123000000,
  "ulfreq": 123000000,
  "bandwidth": 123000,
  "start": "2020-03-31T15:00:13+00:00",
  "end": "2020-03-31T15:00:13+00:00"
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error_tle1":"Error decoding the data, check the format of TLE"
}
```

Example:

example.sh

```
curl -d @EditPass.json -X POST http://X.X.X.X:YYYY/passes/12
```

Delete Pass

Description: Deletes an existing pass identified by the ID parameter. It also deletes all the ground station passes.

Method: DELETE

Path: /passes/{id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"Pass not found"
}
```

Example:

example.sh

```
curl  -X DELETE http://X.X.X.X:YYYY/passes/12
```

Delete Ground Station Pass

Description: Deletes the part of a pass assigned to a concrete ground station.

Method: DELETE

Path: /passes/gs/{gs_id}/pass/{gs_pass_id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
```

```
  "error":"Pass not found"
}
```

Example:

example.sh

```
curl  -X DELETE http://X.X.X.X:YYYY/passes/1/pass/12
```

Get future ground station's chain passes

Description: Returns the information about the future passes of a concrete ground station's adapter. For example it can return all the future passes for the S-Band chain. It can be useful to display it in the adapters page of the user interface.

Method: GET

Path: /passes/future/gs/{gs_id}/adapter/{ad_id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "Norad": "25682",
    "SatName": "satname",
    "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
18422-4 0  9997",
    "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
    "dlfreq": 123000000,
    "ulfreq": 123000000,
    "bandwidth": 123000,
    "start": "2020-03-31T15:00:13+00:00",
    "end": "2020-03-31T15:00:13+00:00",
    "gain": 5,
    "recordingsid": 4,
    "Passid": 12,
    "StartTime": 2020-03-31T14:52:13+00:00,
    "EndTime": 2020-03-31T15:02:13+00:00,
    "MaxEl": 52.4,
    "AOS": 2020-03-31T14:55:13+00:00,
```

```
    "LOS": 2020-03-31T15:00:13+00:00
  },
  {...}
]
```

- 500 Internal Server Error

internalServerErrorExample.json

```
{
  "error":"error message"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/passes/future/gs/1/adapter/1
```

Get last ground station's chain passes

Description: Returns the information about the last N passes of a concrete ground station's adapter. For example it can return all the future passes for the S-Band chain. It can be useful to display it in the adapters page of the user interface.

Method: GET

Path: /passes/last/gs/{gs_id}/adapter/{ad_id}

Body:

N.json

```
{
  "n":10
}

**Answers**:
  * 200 OK
<code javascript answerOKExample.json>
[
  {
    "Norad": "25682",
    "SatName": "satname",
    "tle1": "1 25544U 98067A   20092.15450753  .00000562  00000-0
```

```
18422-4 0  9997",
    "tle2": "2 25544  51.6460   9.6389 0005110  77.4073  65.8523
15.48936344220071",
    "dlfreq": 123000000,
    "ulfreq": 123000000,
    "bandwidth": 123000,
    "start": "2020-03-31T15:00:13+00:00",
    "end": "2020-03-31T15:00:13+00:00",
    "gain": 5,
    "recordingsid": 4,
    "Passid": 12,
    "StartTime": 2020-03-31T14:52:13+00:00,
    "EndTime": 2020-03-31T15:02:13+00:00,
    "MaxEl": 52.4,
    "AOS": 2020-03-31T14:55:13+00:00,
    "LOS": 2020-03-31T15:00:13+00:00
  },
  {...}
]
```


- 500 Internal Server Error

internalServerErrorExample.json


```
{
  "error":"error message"
}
```


Example:

example.sh

```
curl -d @N.json -X GET http://X.X.X.X:YYYY/passes/last/gs/1/adapter/1
```

Health Tests endpoints

Get Health Tests

Description: Returns all health tests scheduled for the indicated time interval. It also includes the tests that have been cancelled, which are the ones containing a value in the *cancelledid* camp.

Method: GET

Path: /healthtests

Body:

GetHealthTests.json

```
{
  "start": "2020-03-31T15:00:13+00:00",
  "end": "2020-03-31T15:00:13+00:00"
}
```

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "eventid": 1, //identifier of the repeating event
    "testname": "moveantennas",
    "gsid": 1,
    "adapterid": 1,
    "time": "2020-03-31T15:00:13+00:00",
    "cancelledid": //only for cancelled health tests, usefull for
uncancelling it (see uncancel health test request)
  },
  {...}
]
```

- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the time format is correct"
}
```

Example:

example.sh

```
curl -d @GetHealthTests.json -X GET http://X.X.X.X:YYYY/healthtests
```

Cancel Health Tests

Description: Cancels the execution of a health test scheduled for a concrete date.

Method: POST

Path: /healthtests/cancel

Body:

CancelTest.json

```
{
  "eventid": 1, //identifier of the repeating event
  "time": "2020-03-31T15:00:13+00:00",
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the time format is correct"
}
```

Example:

example.sh

```
curl -d @CancelTest.json -X POST http://X.X.X.X:YYYY/healthtests/cancel
```

Schedule Health Tests

Description: Schedules the execution of health tests for a concrete chain of a ground station. This tests can be repeating or a single test, indicating the appropiate value in the *repeatperiod* camp.

Method: POST

Path: /healthtests

Body:

NewHealthTest.json

```
{
  "eventid": 1, //identifier of the repeating event
  "testname": "moveantennas",
  "gsid": 1,
  "adapterid": 1,
  "creationtime": "2020-03-31T15:00:13+00:00", //first time it has to
be executed
  "repeatperiod": 24 //repetition time in hours, 0 for a non-repeating
test
}
```

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"check the time format is correct"
}
```

Example:

example.sh

```
curl -d @NewHealthTest.json -X POST http://X.X.X.X:YYYY/healthtests
```

Delete Health Test Event

Description: Deletes, not only one health test but all its repetitions.

Method: DELETE

Path: /healthtests/{event_id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

badRequestExample.json

```
{
  "error":"the event does not exist"
}
```

Example:

example.sh

```
curl -X DELETE http://X.X.X.X:YYYY/healthtests/1
```

Get Health Tests History

Description: Returns a list with the information of all the executed health tests.

Method: GET

Path: /healthtests/history

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "eventid": 1, //identifier of the repeating event
    "testid": 43,
    "testname": "moveantennas",
    "gsid": 1,
    "adapterid": 1,
    "time": "2020-03-31T15:00:13+00:00",
    "testStatus": "complete"
  },
  {...}
]
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/healthtests/history
```

Get Health Tests Report

Description: Returns a report with the results of a test if it has finished.

Method: GET

Path: /healthtests/report/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  // content of the health report
}
```

- 423 Locked

LockedExample.json

```
{
  "message":"The report is not ready yet"
}
```

- 400 Bad Request

BadRequestExample.json

```
{
  "error":"There is no test for the especified id"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/healthtests/report/1
```

Activate Health Test

Description: Activates a cancelled health test in order to put it on schedule again.

Method: DELETE

Path: /healthtests/cancelled/{id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

BadRequestExample.json

```
{
  "error":"There is no test for the especified id"
}
```

Example:

example.sh

```
curl -X DELETE http://X.X.X.X:YYYY/healthtests/cancelled/1
```

Execute Health Test

Description: Runs a concrete health test on the specified adapter and ground station.

Method: POST

Path: /healthtest/execute/{testName}/{adapterID}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
```

```
  "testID": 14,
}
```

</code>

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The test does not exist"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/test/run/moveantennas/1
```

Get Available Health Test

Description: Gets the names of the tests available for a concrete adapter and ground station.

Method: GET

Path: /healthtest/names/{gsid}/{adapterID}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  ["moveantennas", "testSDR", ...]
}
```

</code>

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The adapter does not exist"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/healthtest/names/1/1
```

Get Last Health Tests

Description: Gets the history of health tests executed in a concrete adapter. Useful to display in the adapter information tab.

Method: GET

Path: /healthtest/last/{gsid}/{adapterID}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "testid": 43,
    "testname": "moveantennas",
    "time": "2020-03-31T15:00:13+00:00",
    "testStatus": "complete"
  },
  {...}
]
```

- 400 Bad Request

badRequestExample.json

```
{
  "error":"The adapter does not exist"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/healthtest/last/1/1
```

Visualization endpoints

View Orbit

Description: Returns the positions of the satellite for the three following orbital periods in GeoJSON format. It uses multilinestrings to separate the points where the satellite's position crosses lat lon limits.

Method: GET

Path: /view/orbit/{norad}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "type": 'MultiLineString',
  "coordinates": [
        [ [100.0, 0.0], [101.0, 1.0] ],
        [ [102.0, 2.0], [103.0, 3.0] ]
  ]
}
```

- 500 Internal Server Error

InternalServerErrorExample.json

```
{
  "error":"could not retrieve satellite's orbit"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/view/orbit/23464
```

View Position

Description: Returns the position of the satellite with id = noradid in GeoJSON format(RFC7946).

Method: GET

Path: /view/position/{noradid}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
    "type": "Point",
    "coordinates": [lon, lat]
}
```

- 400 Bad Request

BadRequestExample.json

```
{
  "error":"no norad especified"
}
```

- 500 Internal Server Error

InternalServerErrorExample.json

```
{
  "error":"could not retrieve satellite's position"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/view/position/28652
```

View Next Pass Information

Description: Returns a report with the results of a test if it has finished.

Method: GET

Path: /view/next

Body: -

Answers:

- 200 OK

answerOKExample.json

```
{
  "Start": "2020-03-31T15:00:13+00:00",
  "End": "2020-03-31T15:00:13+00:00",
  "Norad": "25685",
  "SatName": "satname"
}
```

- 500 Internal Server Error

InternalServerErrorExample.json

```
{
  "error":"could not retrieve next pass"}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/view/next
```

Uplink endpoints

Get Pass Planning

Description: Returns the graph of commands planned for a pass.

Method: GET

Path: /uplink/planning/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "node_id": 1,
    "command_id": 1,
    "command_name": "hello",
    "parent_node": ""
  },
  {
    "node_id": 2,
    "command_id": 2,
    "command_name": "goodbye",
    "parent_node": "1"
  },
  {...}
]
```

- 400 Bad Request

InternalServerErrorExample.json

```
{
  "error":"could not retrieve the pass"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/uplink/planning/4
```

Get Satellite's Commands

Description: Returns all available commands for a specific satellite.

Method: GET

Path: /uplink/commands/{norad}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "command_id": 1,
    "command_name": "hello"
  },
  {
    "command_id": 2,
    "command_name": "goodbye"
  },
  {...}
]
```

- 400 Bad Request

InternalServerErrorExample.json

```
{
  "error":"the satellite does not exist"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/uplink/commands/25463
```

Get Commands History

Description: Returns a list with all the commands that have been sent during a pass, identified by its id. Includes the command id, its name and the time when it was sent.

Method: GET

Path: /uplink/history/{id}

Body: -

Answers:

- 200 OK

answerOKExample.json

```
[
  {
    "command_id": 1,
    "command_name": "hello",
    "execution_time":"2020-03-31T15:00:13+00:00"
  },
  {
    "command_id": 2,
    "command_name": "goodbye",
    "execution_time":"2020-03-31T15:00:13+00:00"
  },
  {...}
]
```

- 400 Bad Request

InternalServerErrorExample.json

```
{
  "error":"could not retrieve the pass"
}
```

Example:

example.sh

```
curl -X GET http://X.X.X.X:YYYY/uplink/history/4
```

Update Pass Planning

Description: Updates the graph of commands planned for a pass.

Method: POST

Path: /uplink/planning/{id}

Body:

CommandsGraph.json

```
[
  {
    "node_id": 1,
    "command_id": 1,
    "command_name": "hello",
    "parent_node": ""
  },
  {
    "node_id": 2,
    "command_id": 2,
    "command_name": "goodbye",
    "parent_node": "1"
  },
  {...}
]
```

Answers:

- 200 OK
- 400 Bad Request

InternalServerErrorExample.json

```
{
  "error":"could not retrieve the pass"}
```

Example:

example.sh

```
curl -d @CommandsGraph.json -X POST
http://X.X.X.X:YYYY/uplink/planning/4
```

Execute Command

Description: Executes the command identified by its id.

Method: POST

Path: /uplink/execute/{id}

Body: -

Answers:

- 200 OK
- 400 Bad Request

InternalServerErrorExample.json

```
{
  "error":"could not retrieve the pass"
}
```

- 403 Forbidden

InternalServerErrorExample.json

```
{
  "error":"the pass has not started yet"
}
```

Example:

example.sh

```
curl -X POST http://X.X.X.X:YYYY/uplink/execute/1
```

D | System Level Tests

Ground Station System Testing

Scheduling functionalities

TS1 – Schedule pass

- Should return 200 OK + the scheduled pass info + check db that the pass is inserted:
  - Scheduling a pass with all the correct parameters when there is no pass overlapping.
  - Scheduling a pass with all the correct parameters and start time is “now”.
- Should return 400 Bad Request:
  - When the pass we are trying to schedule overlaps another pass.
  - When the pass we are trying to schedule is in the past.
  - When there is some missing camp.
  - When a camp does not follow the correct format.

TS2 – Delete pass

- Should return 200 OK + check db that the pass is deleted
  - When requesting for the deletion of an existing pass
- Should return 400 Bad Request:
  - When requesting for the deletion of a non-existing pass.

Manual management functionalities

TM1 – Manually move antennas

- Should return 200 OK and the antennas should move (check camera):
  - Moving the antennas introducing a valid azimuth, elevation and adapter id.
- Should return 400 Bad Request:
  - Introducing invalid azimuth and elevation values.
  - Introducing an invalid adapter id.
  - If some camp is missing.

TM2 – Manually stop antennas and the antennas should stop moving (check camera):

- Should return 200 OK:
  - Stopping the antennas of a valid adapter id.
- Should return 400 Bad Request:
  - Introducing an invalid adapter id.

TM3 – Manually get rotor position

- Should return 200 OK + the position of the rotor:
  - When requesting a valid adapter.
- Should return 400 Bad Request:
  - When requesting for an adapter id that does not exist.

TM4 – Manually set Rx / Tx

- Should return 200 OK + correctly set the TR/TX mode:
  - When requesting on an existing adapter.
- Should return 400 Bad Request:
  - When requesting for a non-existing adapter.

TM5 – Manually park rotor

- Should return 200 OK + antennas should move to known parking position (check camera):
  - When requesting a valid adapter
- Should return 400 Bad Request:
  - When requesting an invalid adapter

TM6 – Start/stop chain

- Should return a 200 OK + turn on or off the chain (there is connection with the sockets)
  - When requesting for a valid adapter id.
- Should return 400 Bad Request:
  - When requesting an invalid adapter.

Task Executor

TE1 – Executing prepass

- All hardware turns on correctly.
- The antennas move to starting position.
- GNURadio blocks start correctly.

TE2 – Executing a pass

- The antennas move to the predicted positions.
- The gnuradio block processes data

TE3 – Executing postpass

- All hardware turns off correctly.
- The antennas move to parking position.
- GNURadio blocks close correctly.

TE4 - Receiving a satellite

- Tracking AIM satellite (TE1, TE2 and TE3).
- Downloading and demodulating data.
- Checking that the demodulated data is correct.

OpCen Backend System Testing

Satellites endpoints tests

TAS1 – Get all satellites
- Should return 200 OK + all the satellites in the database (see documentation to see format)

TAS2 – Toggle Autoupdate
- Should return 200 OK + the new autoupdate value which should be the same one as stored in the database property autoupdate → new value = !old value :
  - When a valid NORAD ID is specified.
- Should return 400 Bad Request + not change anything:
  - When an invalid NORAD ID is specified.

TAS3 – Toggle Favourite
- Should return 200 OK + the new favourite value which should be the same one as stored in the database property favourite → new value = !old value :
  - When a valid NORAD ID is specified.
- Should return 400 Bad Request + not change anything:
  - When an invalid NORAD ID is specified.

TAS4 – Add new Satellite
- Should return 200 OK:
  - When adding a satellite with a non-repeating NORADID and all the necessary camps are present.
- Should return 400 Bad Request:
  - When adding an already existing satellite.
  - When there is some camp missing.
  - When some camp does not follow the required format.

TAS5 – Edit existing Satellite
- Should return 200 OK:
  - When editing a satellite with an existing NORADID and all the necessary camps are present.
- Should return 400 Bad Request:
  - When editing a non-existing satellite.
  - When there is some camp missing.
  - When some camp does not follow the required format.

TAS6 – Delete a Satellite
- Should return 200 OK:
  - When deleting a satellite with a valid NORAD ID.
- Should return 400 Bad Request:
  - When deleting a satellite with an non-valid NORAD ID.

Ground station endpoints tests

TGA1 – Get Ground Stations
- Should return 200 OK + all the ground stations in the database (see documentation to see the used format).

TGA2 – Get Ground Station Detailed

- Should return 200 OK + all the detailed information of a concrete ground station and its adapters:
  - When requesting for a valid ground station id.
- Should return 400 Bad Request:
  - When requesting for a non-existent ground station id.

TGA3 – Move Antennas

- Should return 200 OK and move the antennas (check cameras):
  - When requesting it to a correct adapter and the Azimuth and Elevation parameters are well-formed (within the acceptance range and in the correct format).
- Should return 400 Bad Request:
  - When there is some parameter missing.
  - When the Azimuth or Elevation value is out of range.
  - When requesting to an invalid adapter id.

TGA4 – Get Antennas Position

- Should return 200 OK + the position of the antennas:
  - When requesting the position of a valid adapter.
- Should return 400 Bad Request:
  - When requesting the position of an invalid adapter.

TGA5 – Get Antennas Camera URL

- Should return 200 OK + the url of the camera:
  - When requesting the position of a valid adapter.
- Should return 400 Bad Request:
  - When requesting the position of an invalid adapter.

TGA6 – Stop Antennas and stop the antennas movement (check cameras):

- Should return 200 OK:
  - When requesting the position of a valid adapter.
- Should return 400 Bad Request:
  - When requesting the position of an invalid adapter.

Passes endpoints tests

TPA1 – Get All Passes

- Should return 200 OK + a list of the scheduled passes or empty if any.

TPA2 – Get Next Pass Time

- Should return 200 OK + the start time of the next pass if any.

TPA3 – Schedule Passes

- Should return 200 OK + a request identifier:
  - When scheduling a single pass with all the valid parameters.
  - When scheduling multiple passes with all the valid parameters.
- Should return 400 Bad Request
  - When some basic camp is missing.
  - When some specific scheduling-mode-camp is missing.

TPA4 – Check Scheduling Request Status

- Should return 204 Partial Content:
  - When the passes have not finished scheduling yet.
- Should return 200 OK + the information of the scheduled passes:
  - When the passes have been scheduled.

Viewer endpoints tests

TVA1 – View Orbit

- Should return 200 OK + a valid orbit (check it is the same one as the output of other programs such as Gpredict):
  - When requesting the orbit of a valid satellite.
- Should return 400 Bad Request:
  - When requesting the orbit of a non-existent satellite.

TVA2 – View Position

- Should return 200 OK + a valid position (check it is the same one as the output of other programs such as Gpredict):
  - When requesting the position of a valid satellite.
- Should return 400 Bad Request:
  - When requesting the position of a non-existent satellite.

TVA3 – View Next Pass Information

- Should return 200 OK + the information the next pass:
  - If there is a scheduled pass in the future and none at the moment.
- Should return 200 OK + the information of the current pass:
  - If there is a pass happening at the moment.
- Should return 200 OK:
  - If there is no scheduled pass in the future.

Adding/deleting Ground Stations

TAG1 – Add Ground Station

- Add a row in the Ground Stations table with the connection parameters.
- Add a row at the adapters table for each available adapter.
- Add the Ground Station to Wireguard network.
- Check connection.
- Check it appears in the Management GUI (GS tab)

TAG2 – Delete Ground Station

- Delete the row in the Ground Stations table .
- Delete all rows in the Adapters table that reference the Ground Station.
- Delete the Ground Station from Wireguard network.
- Check connection.
- Check does not appear in the Management GUI (GS tab)

OpCen GUI System Testing

Satellites management tests

TS1 - Getting all satellites

- Opening the management tab should display all satellites cards.
- Cards should display satellite's name, its NORAD ID, the frequencies and favourite and autoupdate icons.

TS2 - Viewing a concrete satellite

- Clicking on a card should display satellite's full information, including the norad of the satellite, satellite's name, the frequencies, the gain, the bandwidth, the TLE, favourite and autoupdate.
- Clicking close button should return to the main screen.

TS3 – Editing a satellite

- Clicking on the edit button of a satellite card should open the edit form.
- Clicking on the edit button of a concrete satellite view, should open the edit form, containing input camps for the norad of the satellite, satellite's name, the frequencies, the gain, the bandwidth, the TLE, favourite and autoupdate.
- The edit form should have all camps autocompleted by default.
- NORAD ID camp should not be editable.
- Clicking the cancel button should not effectuate the edit.
- Clicking on the Save button should propagate the changes.

TS4 - Adding a Satellite

- The main tab should display an add satellite card.
- Clicking the add satellite card should open an empty form, containing input camps for the norad of the satellite, satellite's name, the frequencies, the gain, the bandwidth, the TLE, favourite and autoupdate.
- Saving the form should create a new satellite.
- Exiting the form should not add a new satellite.

TS5 – Adding a satellite as Favourite

- If a satellite is set as favourite, the favourite icon should be filled, whether it was clicked on the card, the add satellite form or the edit satellite form.
- If the satellite is not set as favourite, the icon should not be filled.

TS6 – Enabling autoupdate TLE

- If a satellite is set as autoupdate, the autoupdate icon should be filled, whether it was clicked on the card, the add satellite form or the edit satellite form.
- If the satellite is not set as autoupdate, the icon should not be filled.
- If the satellite is not set as autoupdate, the TLE file should remain the same, even if a newer version is available at the Celestrak page.

TS7 – Deleting a satellite

- When deleting a satellite, a warning message should be displayed.
- When accepting the warning message, the satellite information should be deleted.
- Deleted satellites should stop appearing at the main page.

Ground Station management tests

TGS1 – Viewing all ground stations
- Opening the Ground Stations tab should display all available Ground Stations
- Ground Stations cards should display a photo of the ground station, its location, name and available chains.

TGS2 – Viewing a concrete ground station
- Clicking on a ground station card, should redirect the the concrete ground station page.
- The ground station information should be displayed there too.
- There should be tabs for each chain the ground station has.
- The first chain tab should be loaded by default.

TGS3 – Chains information
- The chains tabs should display the camera of the chains, as well as the position of the antennas.
- There should be buttons to manually control the movement of the antennas.

TGS4 – Moving the antennas
- Setting a value for the azimuth and elevation and clicking move should start antenna's movement.
- Clicking stop should make the antenna stop moving progressively.

TGS5 – Changing chains
- Clicking on another chain tab should display the information about the clicked chain.
- All the previous operations should be done without problem in the new chain.

Passes scheduling management tests

TPS1 – Viewing all passes
- Opening the passes tab should display a table with all scheduled passes.
- The table should contain the name of the satellite, its NORAD ID, the start and end of the pass (including pre and post pass operations), the LOS and AOS and the max elevation.

TPS2 – Scheduling passes
- There should be a button on the main page to schedule a pass.
- Clicking on the schedule button should open a form.
- The form should let the user choose different scheduling modes.
- The form should contain camps for the satellite's parameters needed to schedule a pass.
- When selecting a satellite, its paramters should be autocompleted if a value is known.
- Clicking the schedule button should schedule the passes.
- The user should see a progress of how passes are being scheduled.
- On cancel the user should be redirected to the main page.

TPS3 – Scheduling modes
- The user should be able to schedule a single pass.
- The user should be able of scheduling multiple passes indicating the total number of passes.
- The user should be able to schedule a number of passes starting on a concrete date, indicating the aforementioned date and the total number of passes.
- The user should be able to schedule all the passes in a time interval indicating the starting and ending date.
- The user should be able to schedule all passes until a concrete date indicating that date.

Viewer client tests

TV1 – Display
- The viewer should display a satic page containing four components
- Each component should be displayed using a card

TV2 – Pass Information
- The first component should be text with information about the next or current pass,
- including the start time, the satellite name and NORAD ID, and the progress if the pass has
- already started.
- The information should update automatically onces a pass ends.

TV3 – Map display component
- The second component should be a world map with the satellite’s next three orbital periods.
- The satellite showed should be the one which has a current or next pass.
- The map should also display the current position of the satellite.
- There should be highlighted day/night zones.
- The position of the satellite and its orbit should be updated automatically.

TV4 – Antennas camera
- There should be a live video of the antennas that will cover the next/current pass.

TV5 – Data flow
- There should be a waterfall graph showing the received data from the satellite if possible.

E | Requirements verification

E.1 GS Requirements Verification

Requirement	Priority	Design	Unit Tests	System Test
API-HC-001	Medium	Schedule Health Test Endpoint		
API-HC-002	Medium	Run Health Test Endpoint		
API-HC-003	Medium	Schedule Health Test Endpoint (through opcen planning)		
API-HC-004	Medium	Run/Schedule Health Tests response is an identifier		
API-HC-005	Medium	Get Test Results Endpoint		
API-HC-006	Medium	Get Health Tests Endpoint		
API-SC-001	High	Schedule Pass Endpoint	Complete	TS1
API-SC-002	High	Schedule Pass Endpoint response contains an identifier	Complete	TS1
API-SC-003	Medium	Edit Pass Endpoint		
API-SC-004	High	Delete Pass Endpoint	Complete	TS2
API-SC-005	High	Schedules are stored to the database and replicated to the OpCen	Does not apply	Does not apply
GS-AST-001	High	Calculated on schedule	Complete	TS1
GS-AST-002	High	Calculated on schedule	Complete	TS1

GS-AST-003	High	Calculated on schedule	Complete	TS1
GS-AST-004	High	Execute Pass function	Complete	TE2
GS-AST-005	Low	Send telecommand Endpoint		
GS-AST-006	High	Execute PrePass function	Complete	TE1
GS-AST-007	High	Execute PrePass event on the task executor	Complete	TE1
GS-AST-008	High	Execute PostPass event on the task executor	Complete	TE3
GS-HW-002	Medium	Send telecommand Endpoint		
GS-HW-003	High	ConfigureAndStartSDR function	Complete	TE2
GS-HW-004	Medium	SetTx/SetRx functions	Complete	TM4
GS-MD-001	High	The entrypoint to the software is an API	Does not apply	Does not apply
GS-MD-002	High	API has access to the controllers and task executor	Does not apply	Does not apply
GS-MD-003	High	The only thing that changes are the mux and the adapters	Does not apply	Does not apply
GS-MD-005	High	Adapters interface implemented for each specific adapter	Does not apply	Does not apply
GS-RFC-001	High	External GNURadio program	Does not apply	TE4
GS-RFC-002	High	External GNURadio program	Does not apply	TE4
GS-RFC-003	Low	External GNURadio program	Does not apply	TE4
GS-RFC-004	High	External GNURadio program	Does not apply	TE4
GS-RFC-005	Medium	External GNURadio program	Does not apply	TE4
GS-RFC-006	High	External GNURadio program	Does not apply	TE4

E.2 OpCen Requirements Verification

Requirement	Priority	Design	Unit Tests	System Test
OPC-DA-001	High	File synchronization	Does not apply	Syncthing working
OPC-DA-002	High	File synchronization	Does not apply	Syncthing working
OPC-DA-003	Medium	File synchronization	Does not apply	Syncthing working
OPC-DA-004	Low	Get Health Test Report Endpoint	Does not apply	
OPC-DA-005	High	File synchronization	Does not apply	Syncthing working
OPC-DA-006	Low	Does not apply	Does not apply	Proxmox displays the statistics and it is accessible because it is in the same VPN
OPC-GSC-001	High	OpCen makes calls to the GS API	Does not apply	All OpCen Backend System tests
OPC-GSC-002	High	Get Antennas Camera URL Endpoint	Does not apply	TGA5
OPC-GSN-001	High	Adding rows to the database	Does not apply	TAG1
OPC-GSN-002	High	Deleting rows in the database	Does not apply	TAG2
OPC-SC-001	High	Schedule Passes	Does not apply	TPA3
OPC-SC-002	Low	Schedule Passes algorithm should distribute it		
OPC-SC-003	High	Schedule Passes		TPA3
OPC-DA-001	High	Does not apply	Does not apply	Created on deploy
OPC-OPCA-001	High	The API endpoints are open inside the NanoSat Lab network	Does not apply	All API tests passing confirm that
OPC-GUI-001	High	Management client Ground Stations tab	Does not apply	TGA1
OPC-GUI-002	High	Management client Passes tab	Does not apply	TPA 1-4
OPC-GUI-003	Medium	Management client Health tests tab		
OPC-GUI-004	High	Viewer Client	Does not apply	TVA 1-3
OPC-GUI-005	Low	Uplink Client		

OPC-GUI-006	High	Downlink Client		
OPC-GUI-007	High	Management client detailed Ground Station view	Does not apply	TGA 2-6
OPC-UP-001	Low	Get Satellite Commands endpoint		
OPC-UP-002	Low	Get Commands History endpoint		
OPC-UP-003	Low	Execute Command endpoint		
OPC-UP-004	Medium	Get/Update Pass Planning endpoint		
OPC-UP-005	Medium	Get/Update Pass Planning endpoint		
OPC-UP-006	Medium	Get/Update Pass Planning endpoint		

E.3 System Communication Verification

Requirement	Priority	Design	Unit Tests	System Test
DF-DC-001	High	Database synchronization		PgLogical working on deployment
DF-DC-002	High	Database synchronization		PgLogical working on deployment
DF-DC-003	High	Files synchronization		Syncthing working on deployment
DF-DC-004	Low	Could add a background task to erase rows periodically as design is concurrent		
DF-DD-001	High	GS GetPassSockets (downlink endpoint)		
DF-DD-002	Medium	GS GetPassSockets (downlink endpoint)		
DF-DD-003	Medium	GS GetPassSockets (downlink endpoint)		
DF-DD-004	Low	GS GetPassSockets (downlink endpoint)		
DF-DD-005	Low	GS GetPassSockets (downlink endpoint)		
DF-UT-001	Medium	Send telecommand Endpoints (OpCen and GS)		
DF-UT-002	Medium	Uplink client		
DF-UT-003	Medium	Uplink client		
DF-UT-004	Medium	Send telecommand endpoint (OpCen)		
DF-UT-005	High	Does not apply		Logs inside the code
N-AIN-002	Low	Does not apply		

N-AIN-004	Low	Does not apply		Close network to the exterior
N-AIN-005	High	Wireguard is the used technology		Wireguard deployed
N-AIN-006	Low	GS GetPassSockets (downlink endpoint) and send command endpoint		

F | GUI Results

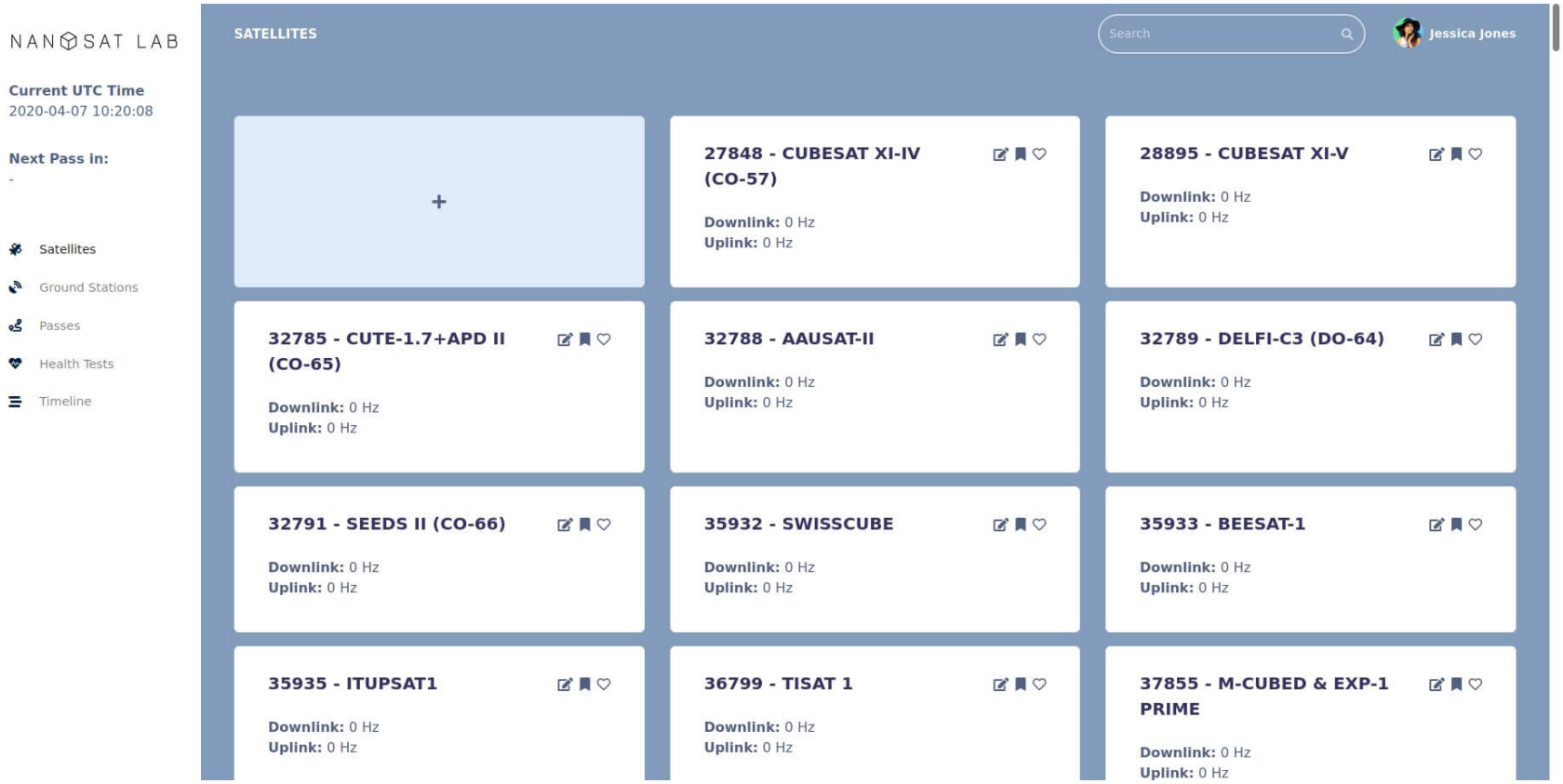


Figure F.1: Main view of the satellite's tab

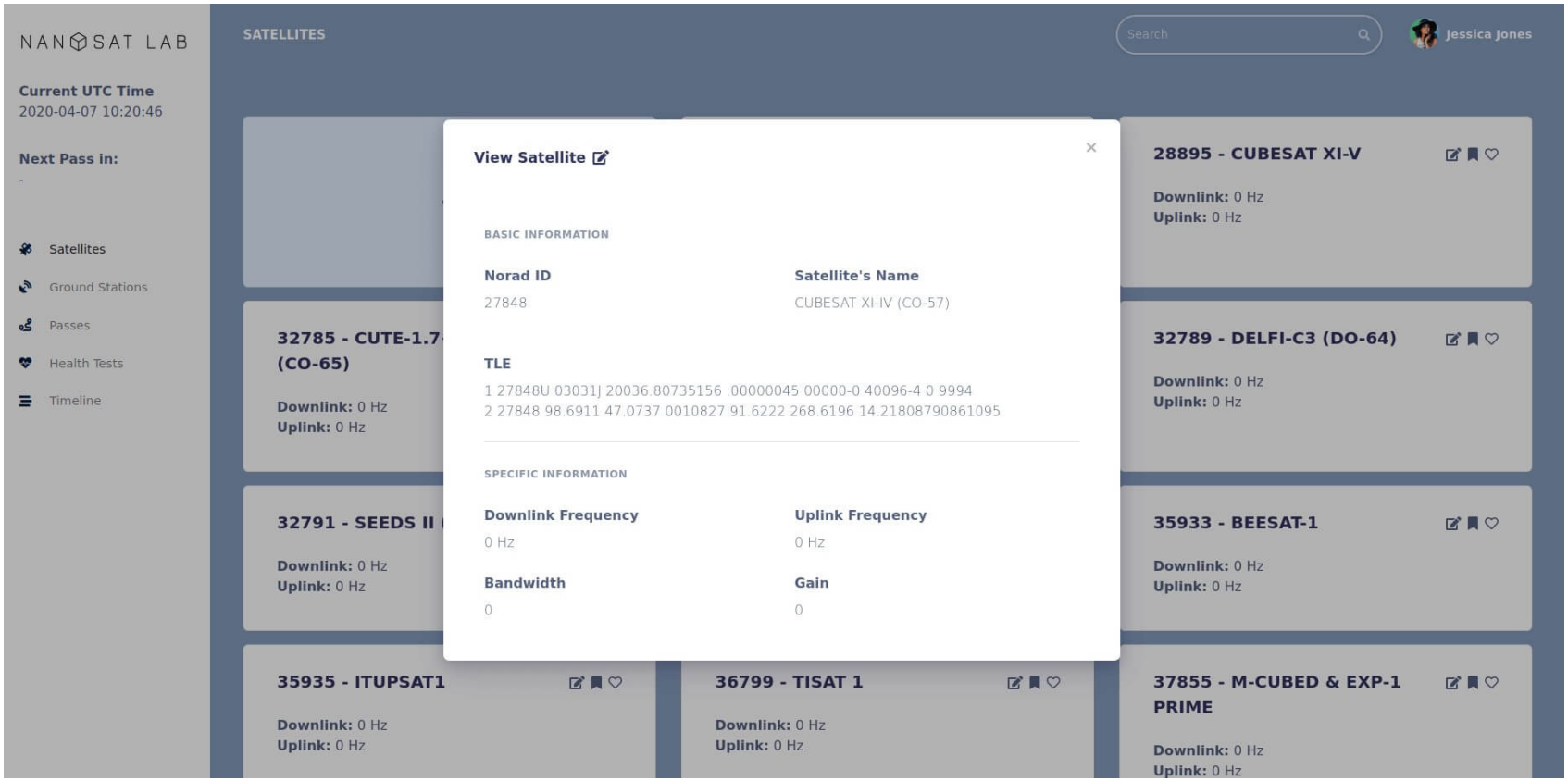


Figure F.2: Detailed Satellite view

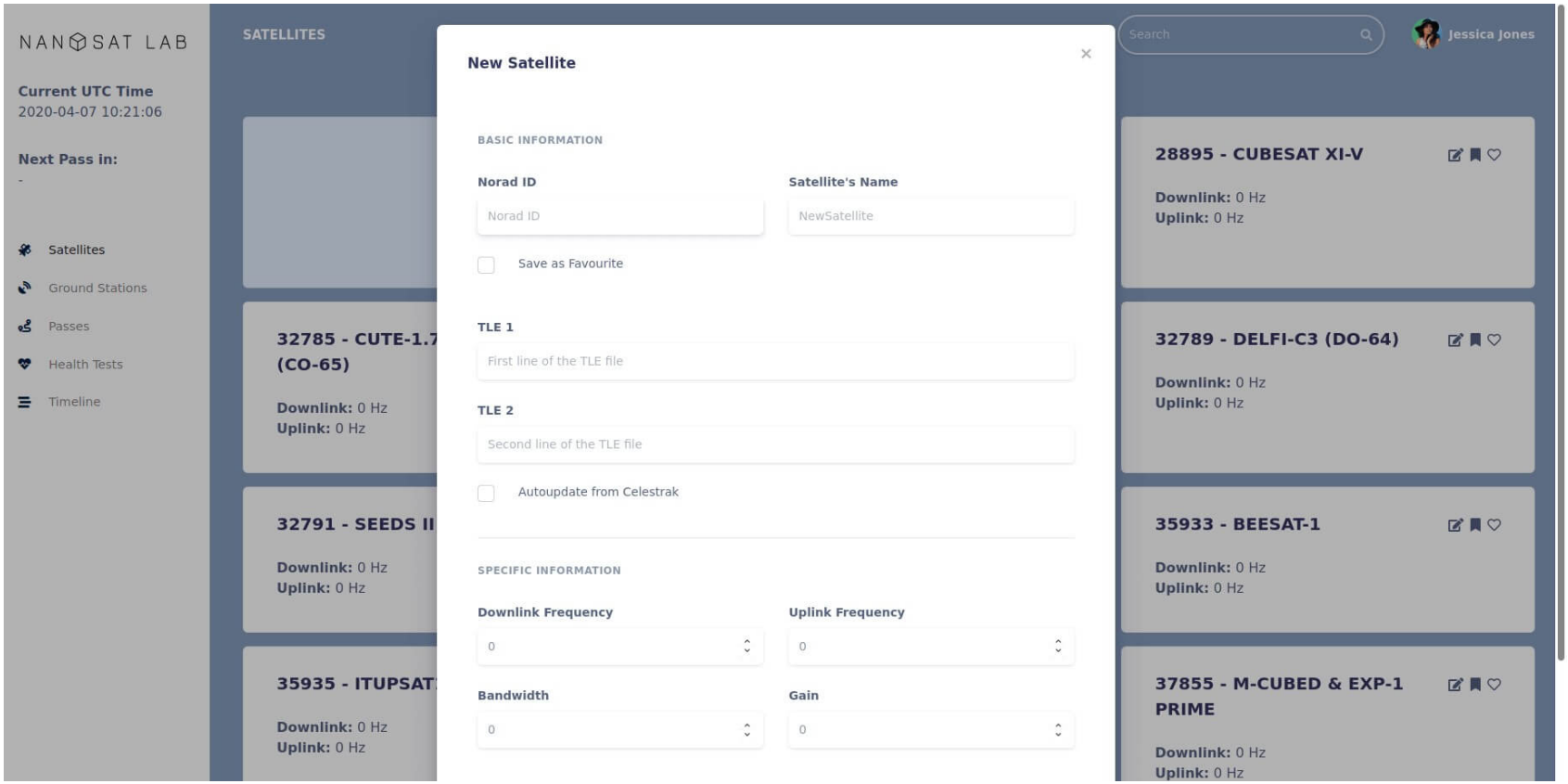


Figure F.3: New satellite form

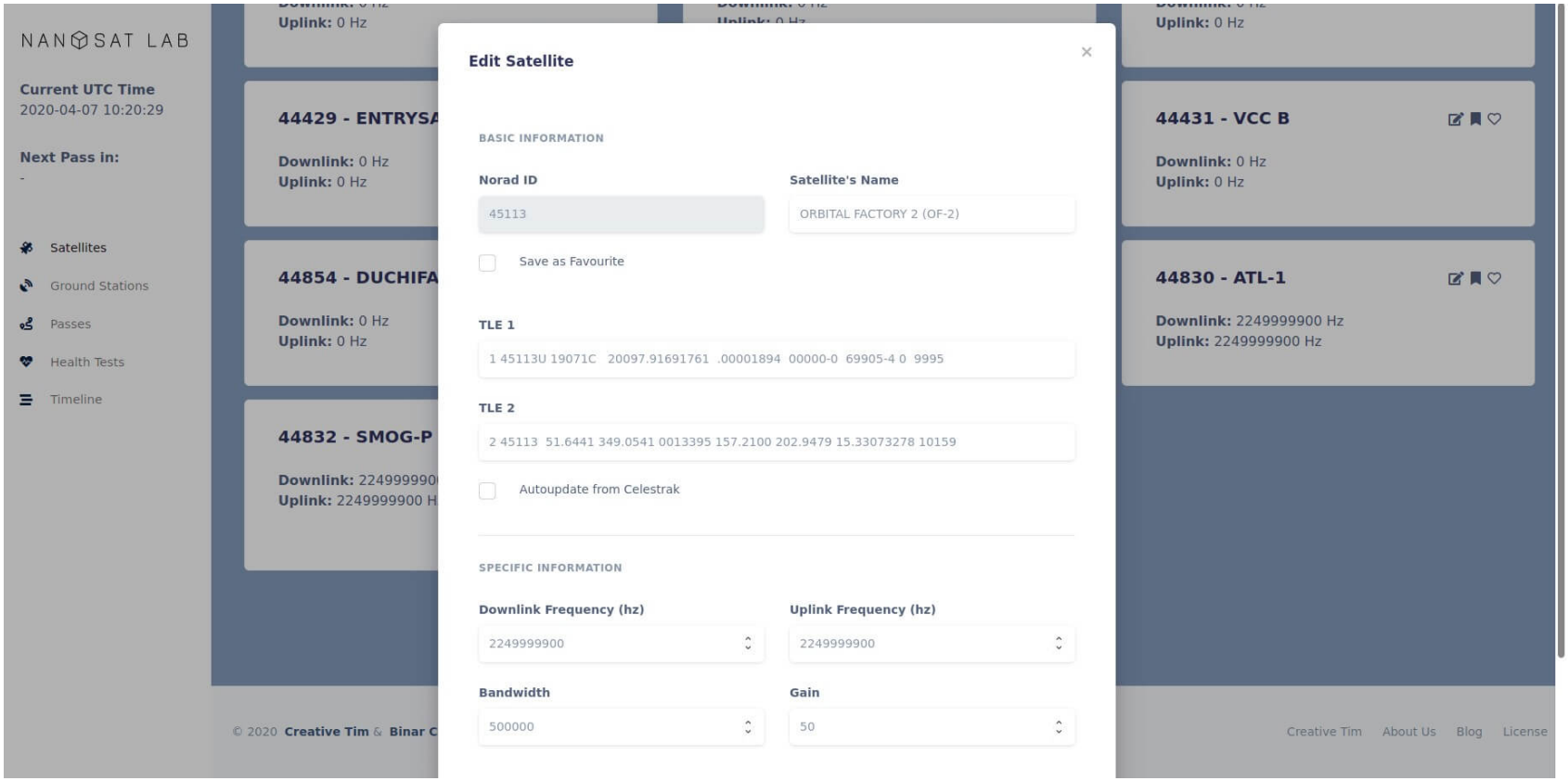


Figure F.4: Edit Satellite I

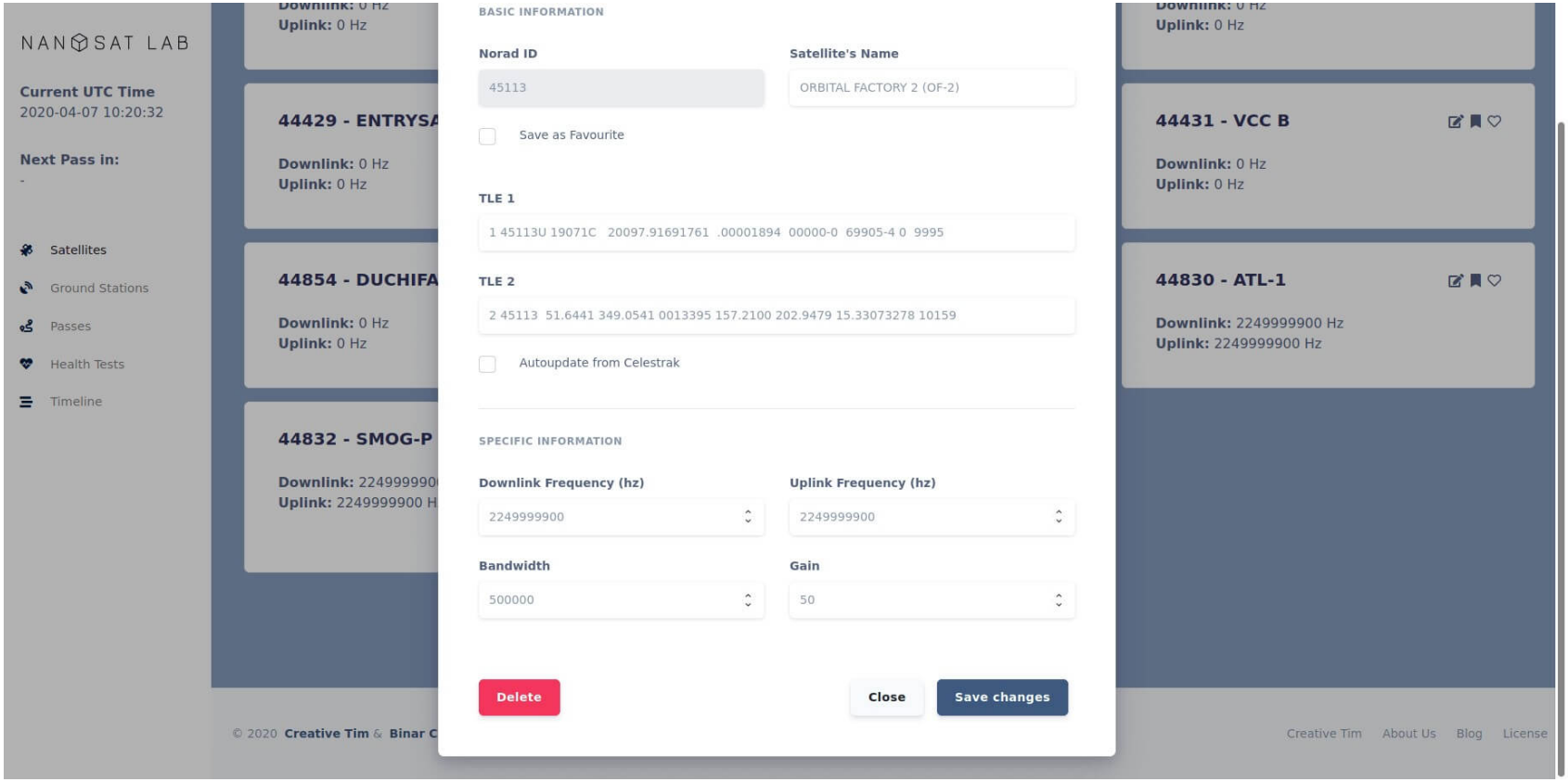


Figure F.5: Edit Satellite II (with delete button)

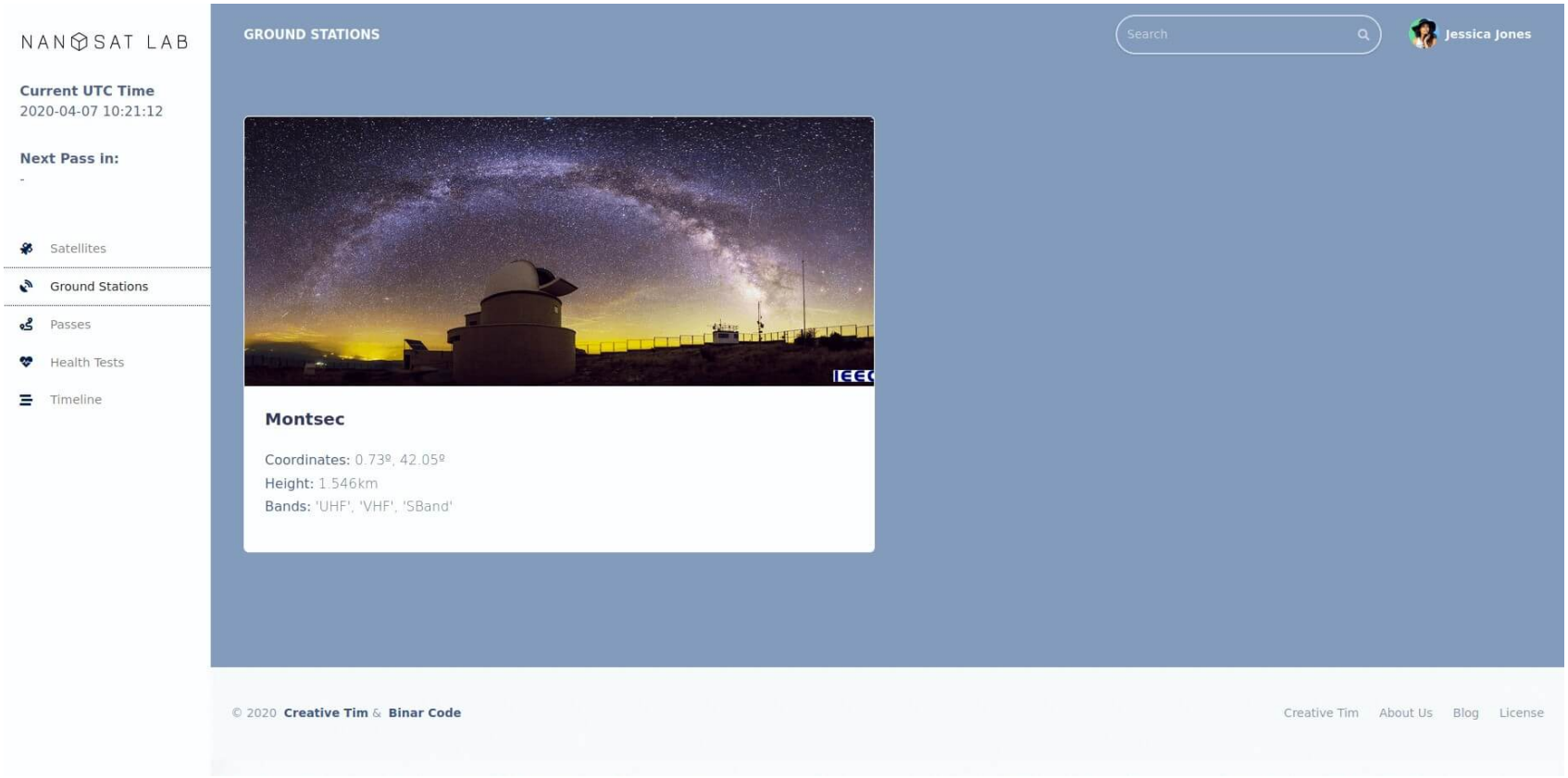


Figure F.6: Ground Stations tab main view

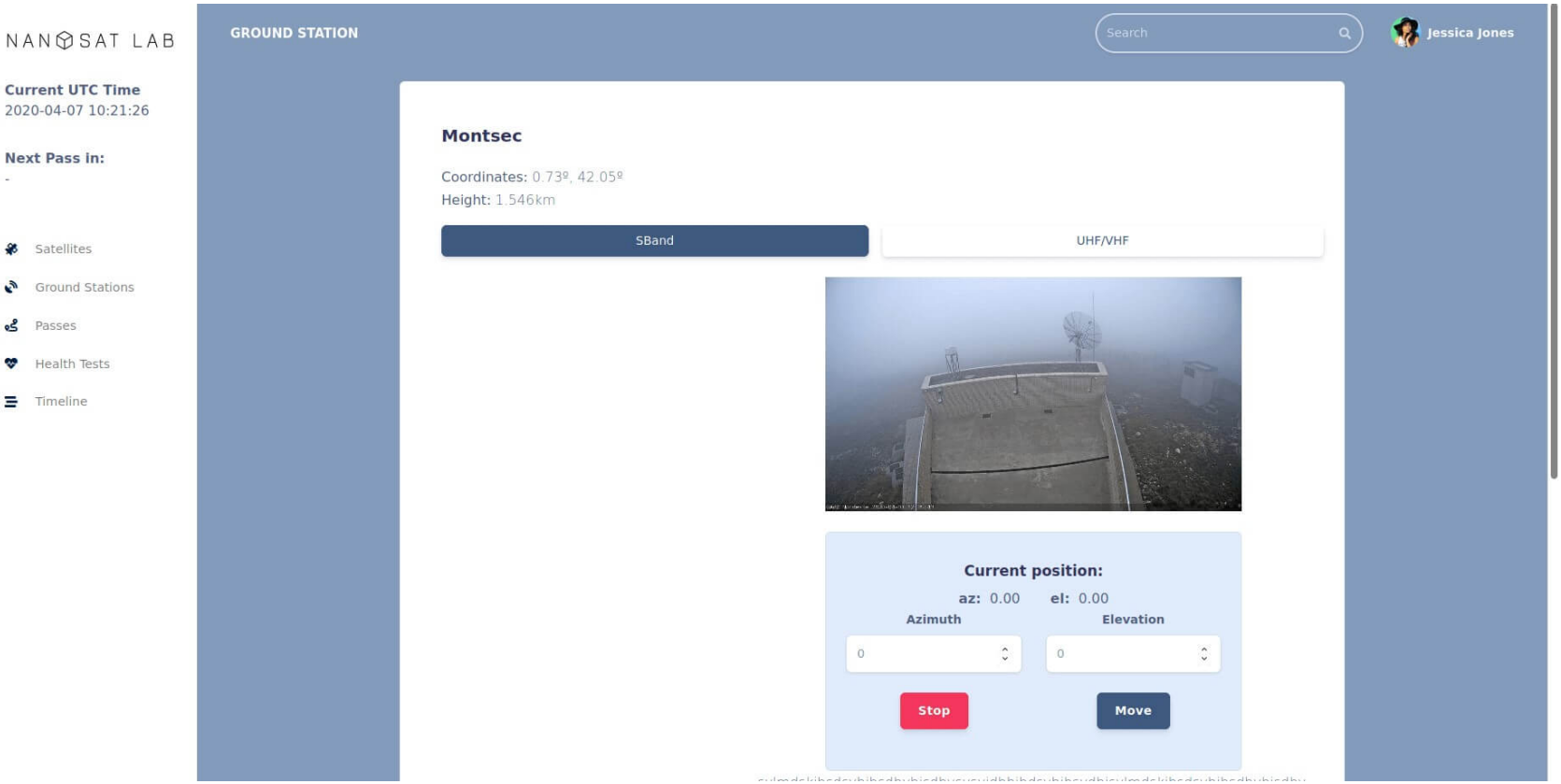


Figure F.7: Montsec Ground Station detailed view

Figure F.8: Passes main view

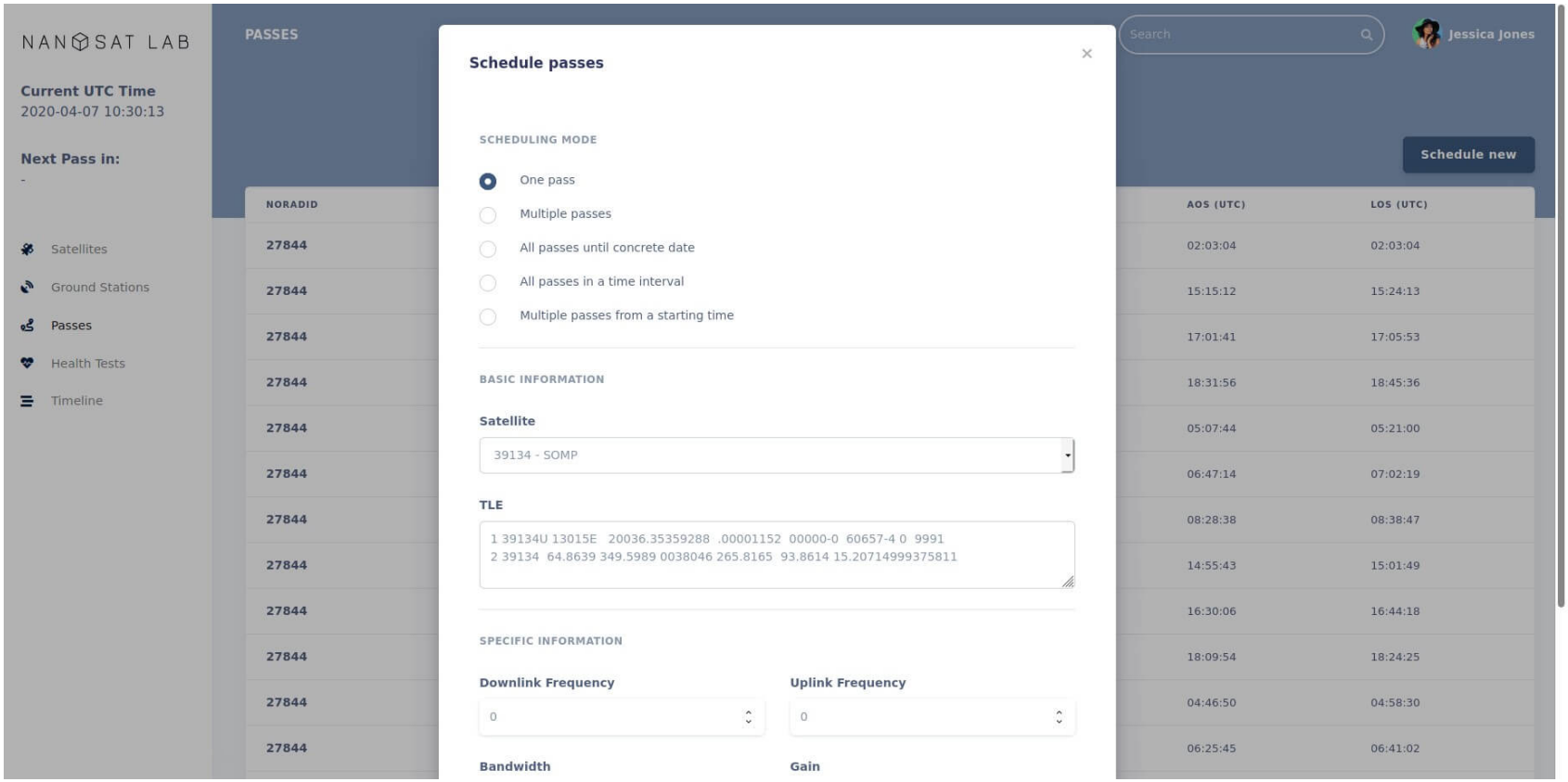


Figure F.9: Schedule pass view example I

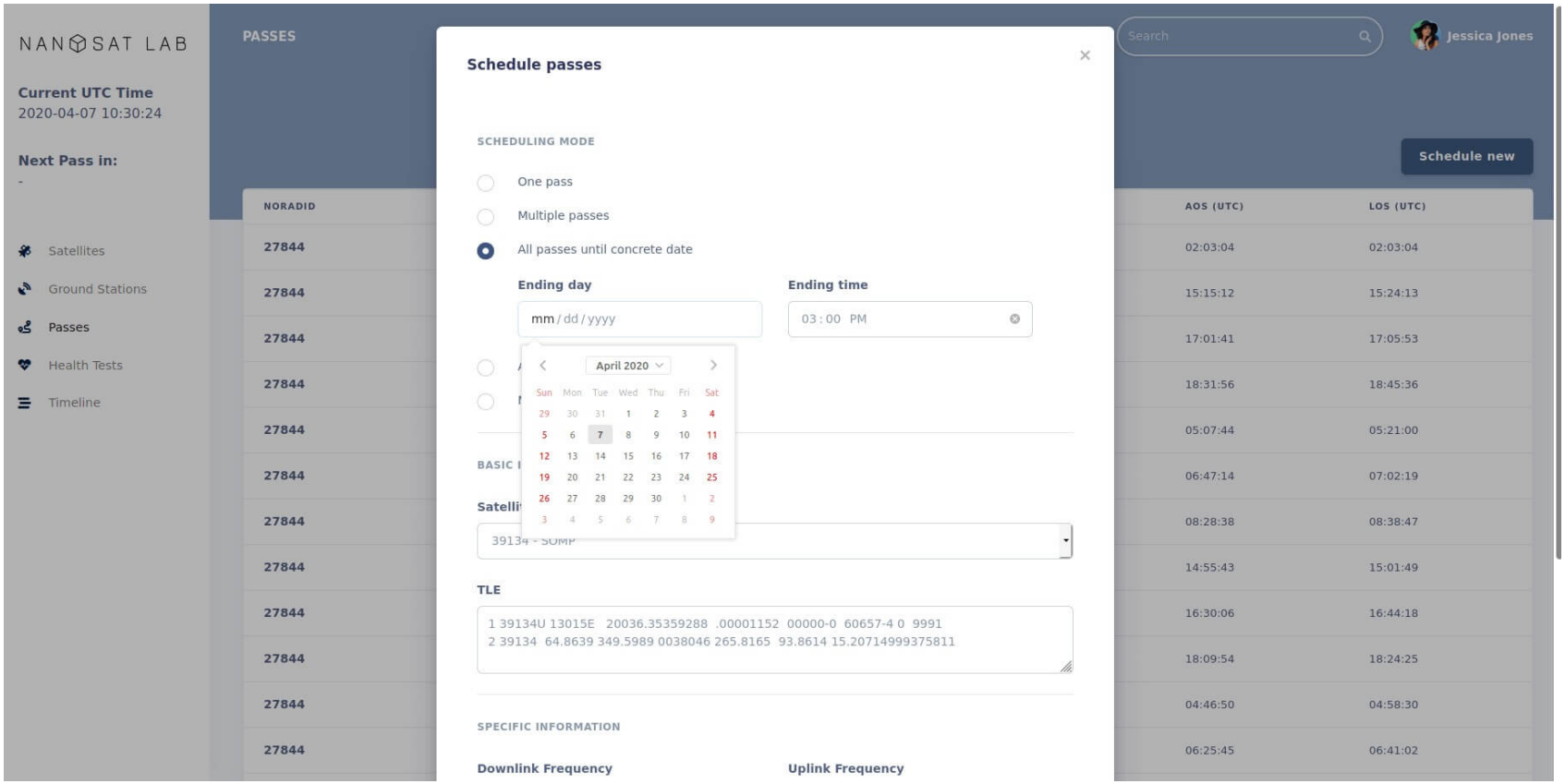


Figure F.10: Schedule pass view example II

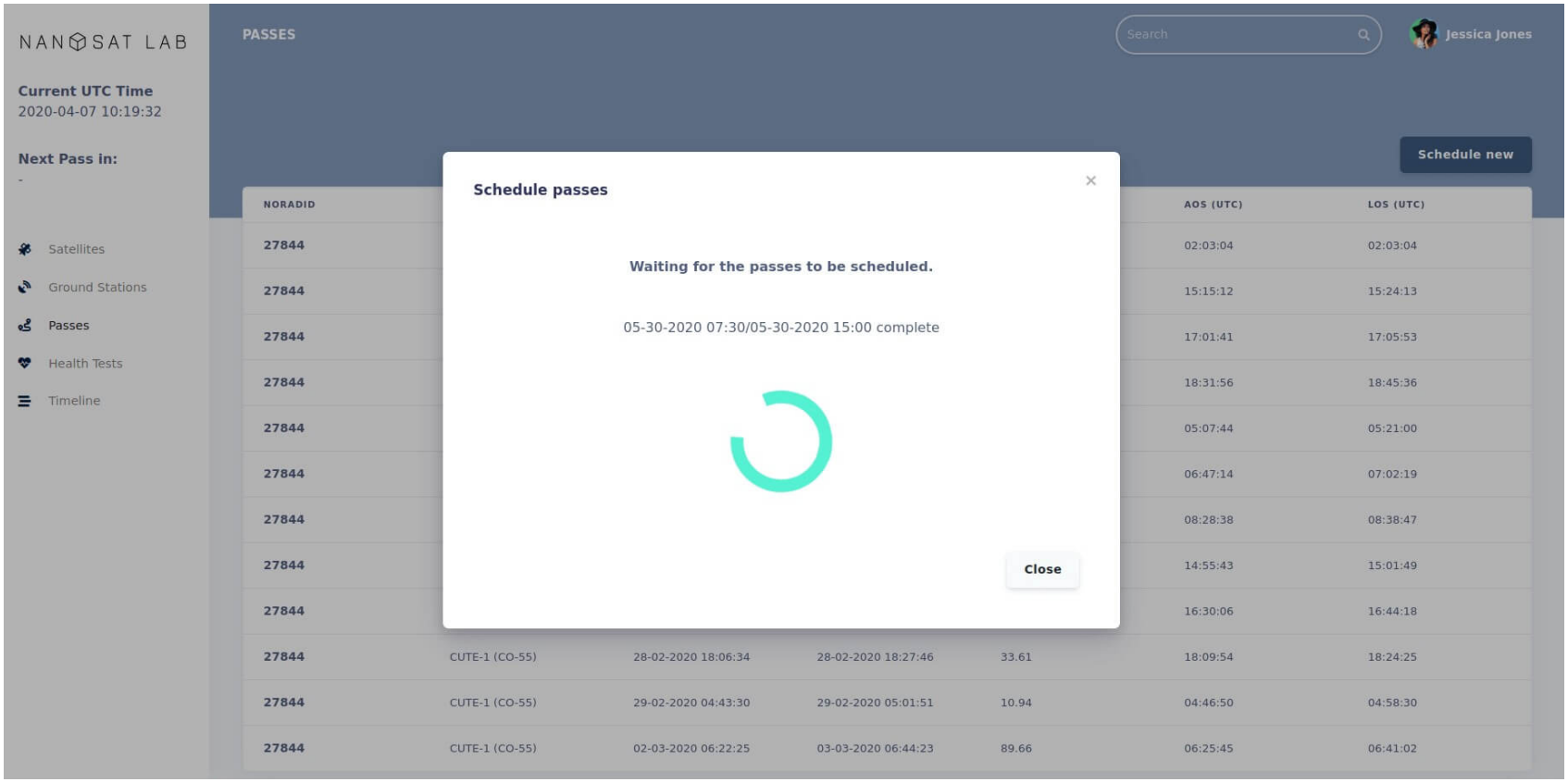


Figure F.11: Scheduling progress

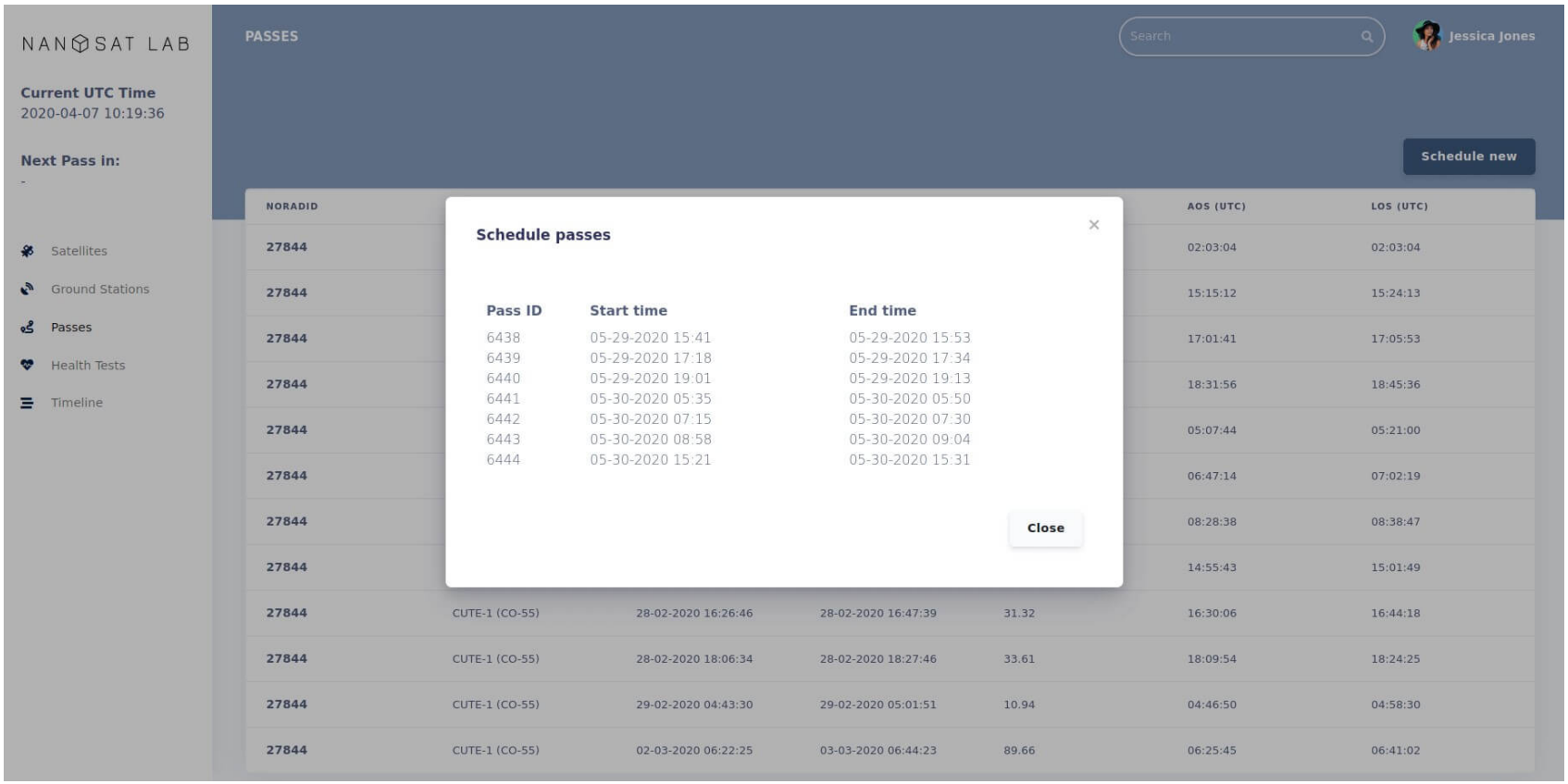


Figure F.12: Scheduled passes overview

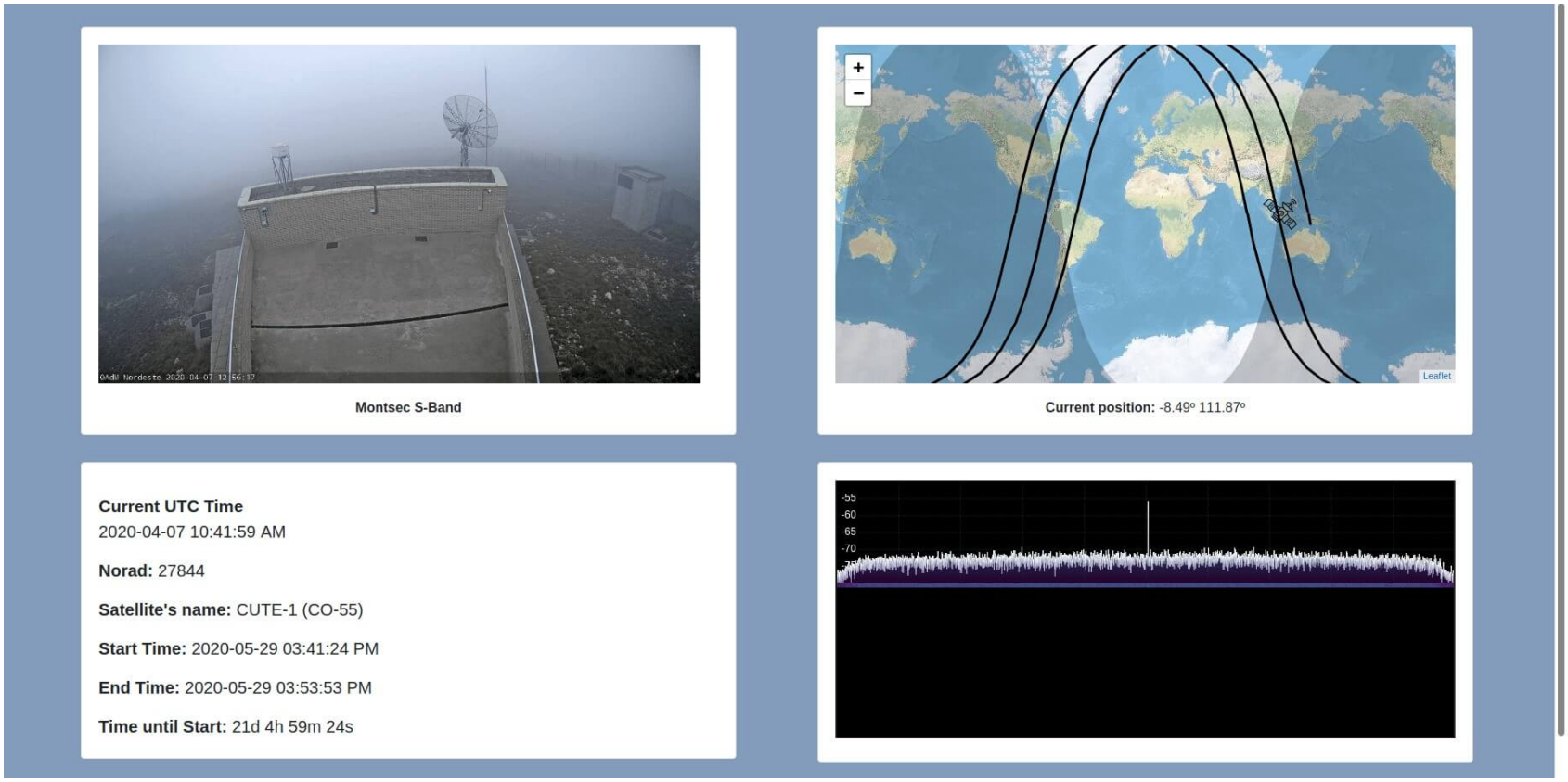


Figure F.13: Viewer client