Team 1: Satellite Operations System Optimizer – SatOpsMQ

by

Walid Al-Dari, Ruth Bezabeh, Rafael Dolores, Youssef Hany,
Stanley Ihesiulo, Hashir Jamil & James Le

A capstone project
presented to York University
in fulfillment of the
requirements for the course
ENG 4000
in
The Lassonde School of Engineering

Toronto, Ontario, Canada, 2024

Capstone Project Teaching Team

The following served as the Teaching Team for this capstone project.

Course Director:	Dr. Franz Newland Professor, Dept. of Earth & Space Science & Engineering Dr. Marzieh Ahmadzadeh Assistant Professor, Dept. of Electrical Engineering & Computer Science Dr. Edris Hassan Assistant Professor, Dept. of Mechanical Engineering
Supervisor:	Dr. Regina S.K. Lee Professor, Dept. of Earth & Space Science & Engineering

Authors' Declaration

We hereby declare that we are the sole authors of this report. This is a true copy of the report, including any required final revisions, as accepted by teaching team of ENG 4000.

Team Members	Student ID
Walid Al-Dari	218 375 162
Ruth Bezabeh	216 171 795
Rafael Dolores	216 142 069
Youssef Hany	216 885 766
Stanley Ihesiulo	216 985 236
Hashir Jamil	217 452 954
James Le	217 270 943

Acknowledgements

We would like to thank all of our families, friends, colleagues, instructors, mentors & supervisors who made this journey in our engineering degrees possible over these last few years. As well we thank the Canadian Space Agency (CSA) for sponsoring the project, giving us the opportunity to work on such an incredible problem. Thank you for all of your patience and help.

Dedication

This report and project is dedicated to Alexander Bath (1994-2023)

Executive Summary

Satellites are a key component of our everyday life. From weather forecasting and telecommunication networks to GPS navigation they are essential to many systems we use daily. To help keep up with the rising demand of earth observation capabilities there is a new five-satellite constellation of imaging satellites that will be launched shortly into a sun-synchronous low-earth orbit by the Canadian Space Agency (CSA). Each satellite is an identical copy of the other and will be placed into orbit equally spaced apart. These satellites will then be used to image the earth night and day based on the needs of a diverse set of users. Communication with satellites is typically done through large ground-based antennas. While the new constellation of satellites does have a globally distributed network of stations, there will only be certain times when a given satellite is available to communicate with a ground station for data to be retrieved. Imaging activities are not the only commands that will need to be sent to the satellite. To ensure a healthy constellation, certain activities may need to be performed periodically (i.e., battery calibration and/or orbital maneuvers). Each satellite will also have limits concerning storage space for images, power to operate the imaging payload, and a variety of other constraints that will have to be accounted for when planning the activities.

The product, SatOpsMQ, is a desktop-based software system capable of scheduling and downlinking images ordered by users amid a wide variety of constraints. The users of this system are operators at the CSA, who, aside from managing image orders, are also equipped to schedule various types of satellite maintenance tasks—details of which will be elaborated on later in the document. The system is configurable while also being largely automated, not requiring any user intervention to carry out the nominal process. While automation is critical, it is still important to allow users of the system to observe the state of the system and override the status of the submitted orders. To accomplish these objectives we built an event-driven system using a micro-services architecture to communicate the various activities that occur between the satellite and ground station, leveraging the Advanced Message Queuing Protocol. This system is capable to run locally on a basic computer. The product will be fully open-source with the complete source code posted on GitHub. The codebase will be regularly reviewed by the sponsor of the project. The technical risks of developing an open-source, event-driven system like this include data inaccuracies resulting in unusable stale data, challenges in selecting data transmission protocols, and scalability issues as sponsors add more assets like satellites and ground stations. SatOpsMQ's design allows for potential applications beyond its original use, such as serving as a broader communication hub within the CSA's infrastructure to enable data transfer between their various systems. The stakeholders of the project are the CSA, the Teaching Team for ENG 4000, and the Nanosatellite Research Lab which is the lab of our supervisor.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Satellites are integral to modern civilization, seamlessly integrating into numerous aspects of our daily lives. They have revolutionized meteorology, communication, and navigation, becoming silent yet indispensable allies in the orchestration of contemporary societal operations. To address the ever-growing demand for earth observation, a new constellation comprising five identical imaging satellites is scheduled for deployment into a sun-synchronous low-earth orbit. These state-of-the-art satellites will furnish around-the-clock imaging capabilities, catering to a broad spectrum of users and applications.

Despite the technological advancements embodied within this new constellation, the task of managing such an elaborate system presents unique challenges. Communication with these orbiting platforms is conventionally managed via terrestrial antennas, which impose inherent limitations on the frequency and timing of command and data retrieval operations. These satellites must undergo a suite of maintenance routines, including battery calibration and orbital adjustments, which are vital for the preservation of their operational integrity. Additionally, the practical constraints of onboard storage, power availability, and other system-specific limitations necessitate a meticulous approach to activity planning and execution

1.2 Background

The project at hand seeks to devise a desktop-based software system designed for the sophisticated task of scheduling satellite imaging and maintenance activities within a complex web of operational constraints. The envisioned system must strike a delicate balance between high-level automation and user configurability. It should minimize the need for human intervention in its nominal workflows, yet retain the capability for users to monitor and comprehend the current state of operations and the status of their imaging requests.

To further describe how satellites function as part of our day-to-day lives, we need to understand the basic operation of satellite communications. Satellites are self-contained communication systems used to transmit and receive signals from Earth or a specific target. They follow a specific altitude relative to the objectives of the satellite and are locked into Earth's orbit similarly to how the Moon can be considered a natural "satellite". Satellites can communicate with Earth through specific ground stations that can receive radio waves emitted at certain wavelength bands for specific information used in the application of a satellite such as Earth Observation (EO), communications networks, and more.

1.3 Problem Statement

The landscape of satellite imaging and ground station scheduling is currently facing a notable gap in resources—there is a distinct lack of open-source applications that can automate the intricate process of satellite imaging scheduling. This automation is critical for handling the high volume of image orders that need processing, potentially in the hundreds per minute. The complexity of operating satellite systems is often associated with a steep learning curve and significant manual effort. Furthermore, professionals in the field are in constant pursuit of more efficient and user-friendly solutions to streamline their operations.

1.4 Project Demand

The project emerges as a necessity because of the ever-growing dependence on satellite imagery for Earth observation tasks across various sectors. The existence of many varying number of satellite constellations significantly complicates the management of imaging and maintenance requests. Operators are now tasked with a complex coordination of scheduling, ensuring satellite health, and meeting strict operational guidelines.

The expanding scope of satellite constellations highlights the urgency for an advanced scheduling system. Such a system is pivotal for processing numerous imaging requests simultaneously and maximizing the use of satellite capabilities. It must integrate smoothly with current operations, yet be flexible to adapt to the dynamic nature of satellite activities, which now face more frequent maintenance needs and narrower windows of operation because of the increased number of satellites.

This project is important when considering the value of satellite data for environmental monitoring, urban planning, agriculture, and crisis management, among other applications. An open-source platform that facilitates quick processing of imaging requests and fine-tunes the scheduling to fit within the satellites' operational limits is a significant step towards making the most of the data these satellites collect. Open-source software, by its nature, offers several advantages as it encourages global collaboration, allowing experts from various fields to contribute and refine the tool set and it promotes transparency, enabling users to understand and trust the scheduling and data acquisition process. It also ensures adaptability, as the software can be customized to meet the specific needs of different satellites and operations. Therefore, this project is a direct answer to the growing demand for efficient, accessible satellite data utilization in the face of expanding global satellite infrastructure.

1.5 Project Goals

SatOpsMQ aims to address the challenges by providing an open-source solution that not only simplifies the automation of scheduling satellite imaging activities but also ensures the system is robust enough to accommodate the demands of rapid and accurate order fulfillment. It recognizes the necessity for operators at CSA, alongside open-source developers, students, and researchers, to have access to a system that is both powerful and accessible, improving the efficacy of satellite operations that are inherently complex and difficult to manage. This project is tasked with optimizing the workflow for scheduling image orders and maintenance requests in a satellite constellation and its associated ground stations. This further involves consideration of key operational constraints, while taking into account the increasing demand for satellite-based earth observations.

1.6 Competitor Solutions

There are several software solutions that offer various capabilities and tools for users ranging from casual observers to industry professionals in the realm of satellite operations and Earth imaging. This section explores existing applications and tools that provide services similar to those envisioned for SatOpsMQ along with their strengths and features. Our assessment of these competitor solutions, such as Earth Explorer, MASS, NASA Operations Software, and Systems Tool Kit (STK), will help identify areas where SatOpsMQ can differentiate itself, particularly in terms of user-friendliness, automation, and adaptability to specific mission needs. By analyzing these existing platforms, we aim to enhance SatOpsMQ with a focus on accessibility, efficiency, and comprehensive functionality, ensuring it stands out in the competitive landscape of satellite scheduling and Earth observation tools.

1.6.1 Earth Explorer

Earth Explorer is a web-based software application developed and maintained by the United States Geological Survey (USGS) [2]. It provides access to satellite imagery allowing users to search, discover, and download a wide range of data sets related to the Earth's surface, atmosphere, and more. Compared to our application it has more parameters and handles larger constellations. To enhance the user experience with our application, we will provide a more user-friendly interface. We understand that Earth science data can be intricate for newcomers, and we are devoted to ensuring that users of all levels can navigate and comprehend our features with ease. By offering intuitive tools, we aim to enable both novices and experts to harness the full potential of Earth science imagery.

1.6.2 Multi-Aspect Automated Satellite Scheduler

The paper introduces the Multi-aspect Automated Satellite Scheduler (MASS), a predictive scheduling software designed to be usable by any CubeSat mission [3]. The program aims to meet the need for accessible hybrid automation in the aerospace industry. By leveraging MASS and adapting it to our team's satellite operations, we can potentially streamline scheduling, reduce manual workload, and enhance the efficiency of our software.

1.6.3 NASA Operations Software

This space mission design tools webpage is an invaluable resource provided by the NASA Small Spacecraft Virtual Institute [4]. It is specifically curated for teams engaged in satellite projects, serving as a comprehensive repository of tools and information. These tools encompass a wide spectrum of critical aspects related to mission design, including trajectory design, satellite optimization tools, and mission operations software. With these resources, we can navigate the intricacies of our project with greater precision and efficiency.

1.6.4 Systems Tool Kit (STK)

Systems Tool Kit (STK) is a software platform developed by Analytical Graphics, Inc. (AGI) that allows users to perform complex analyses of various platforms and scenarios in the air, sea, space, and ground domains [5]. STK can model different aspects of satellite operations, such as trajectory, communication, and imaging, and provide advanced features for traffic management and bandwidth control. STK is used by a wide range of industries, including aerospace, defense, telecommunications, and more [5].

1.7 Pointers on How To Read This Report

- All GitHub repositories are available at the SOSO GitHub Organization
- All supplementary figures and materials are available at the SatOpsMQ Supplementary Materials Repository
- Throughout the document there are many hyperlinks in red to external sites and hyper-references in blue to locations within the report. These are designed to make reading easier and we have tried our best to make sure the links are consistent and correct but it is possible a link here or there does not lead to the correct location.
- Almost all diagrams are vector graphics and you can zoom in on any pdf reader without the diagrams losing resolution.
- There is a comprehensive appendix of extra materials, including figures, tables and acronyms.
- If you have any questions please feel free to reach us by email for any clarification.

Chapter 2

Technical Volume

2.1 Needs & Requirements

2.1.1 Product Vision Board

An agile product vision board is a tool that helps you define and communicate the vision and strategy of your product. It consists of five sections: vision, target group, needs, product, and business goals. The vision states the overarching goal and purpose of the product. The target group describes the market segment, customers, and users of the product. The needs are the problems or desires that the product should address. The product is the key features or capabilities that the product should offer. The business goals are the benefits that the product should create for the customers, users, and the business.

As we reach the end of the project, we can see that the features that were needed by the user have been completed, tested, and verified to meet the user needs for the web-app's functionality. In terms of the business goals for the project, we detail the economical aspects of the web-application as well as its merits for the user later in the project.

2.1.2 User Needs & Requirements

For the user needs and requirements, as per the agile project management scheme that the team has adopted the use of user stories to address the need and requirements for the user in the third chapter.

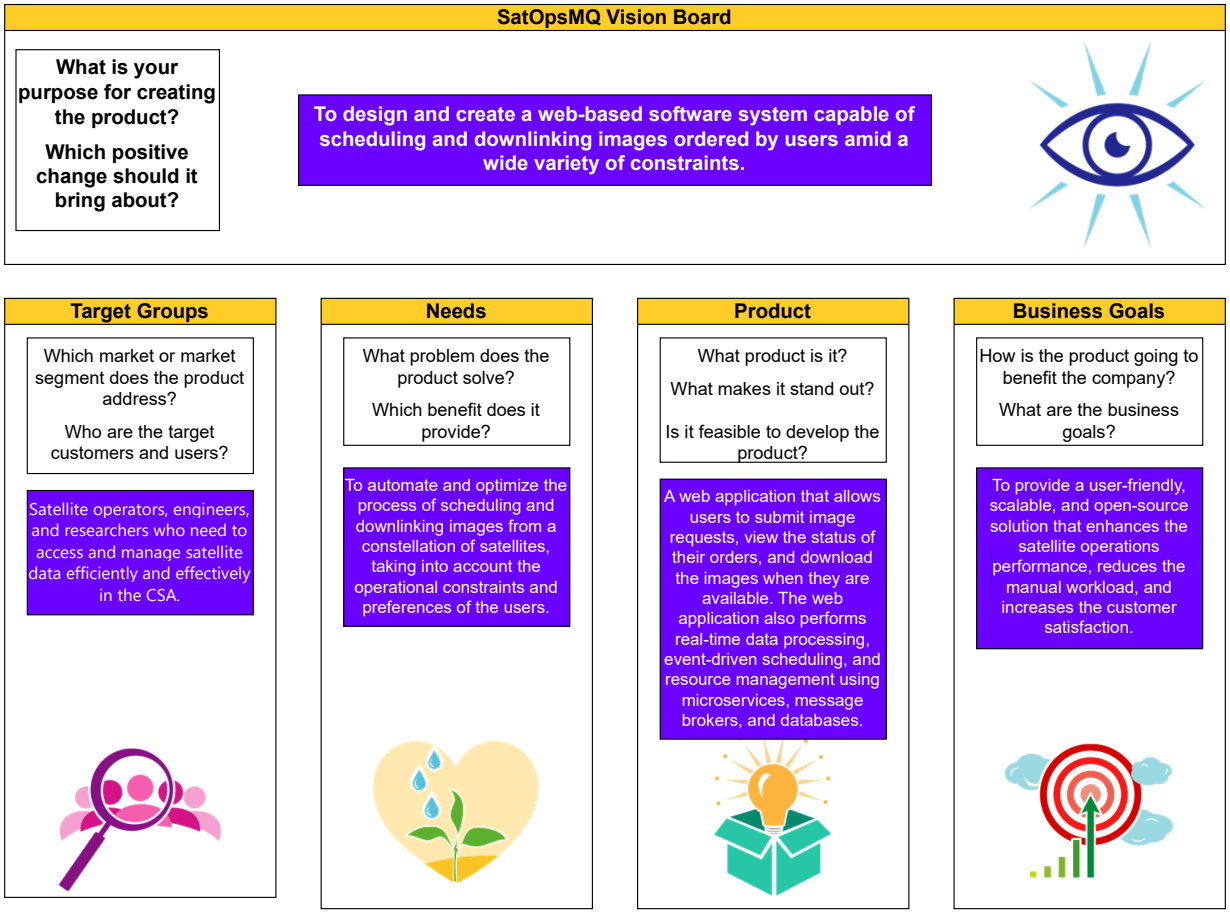


Figure 2.1: Product Vision Board for SatOpsMQ

2.1.3 System Needs & Requirements

Similarly to the user needs and requirements, the system needs and requirements for the web application can be found in the third chapter which are primarily the back-end servers, front-end client, and development operations for the web application.

2.1.4 Key Challenges & Difficulties

The main challenges that the team faced when trying to address the needs and requirements for the project was being able to continuously improve and add the necessary features as required by the users/stakeholder of the web application.

Firstly, when we would communicate with the CSA regarding the features they wanted, there was always an element of ambiguity as they wanted to see which solution we would adopt for various needs the user wanted. For example, when deciding on a architectural frame work to see which technology stack the team would use, the CSA initially said that they were open to various alternatives that used a monolithic structure, microservices structure, or cloud-based deployment. However, as we began to consolidate our solution, the CSA stated that they didn't want a cloud based solution that would lock them into a specific vendor which limited our options for the development of the web application.

This is one of many examples where we had to exchange the solutions we would come up with for the user and deciding if we should push back on the requirement or accept it as needed for the project.

2.1.5 Scope Delineation

The scope of work needed to be done that would be necessary for the project is:

- Development of back-end servers using PostgreSQL that would handle all data transfer and communication for user orders.
- Development of front-end client that can handle deliver of user orders to the back-end and display necessary data based on user needs.
- Development operations used to ensure that all connections between the back-end and front-end work seamlessly for the user.

The scope of work that isn't necessary part of the project is:

- Have discrete tests that validate the web application on user systems. (Ex: the CSA can pull the code-base and run the web application at any point during the development process but we don't have specific tests to verify it works on their systems).
- User login registration is not part of the scope for the project. We only focus on developing a functional application for the users of the web-app.
- Only order management of imaging, maintenance, and outage orders are taken into account. The database only looks at the orders statically through .json formats and not actual images from satellites.
- Realistic communications between the satellites and ground stations is not part of the project's scope. All satellites and ground stations are assumed to be mock/simulated endpoints.

2.1.6 Requirements Compliance

To ensure that the web application is compliant with all release in the agile project management scheme that we adopted, the team met with the CSA on a weekly basis to discuss the progress of the web application and check if there were any bugs or non-compliances that occurred while we were demoing or reviewing the web application. This is later discussed extensively in the report where we verified and validated features as well (Test Tracking for Non-Compliances).

2.2 Engineering Concepts & Theories Applied

The relevant engineering concepts & theories applied to the product are based on realistic operations of satellites and ground stations and how they communicate. As well, software engineering principles for full-stack web-based applications are used. The back-end of the web-app takes the satellite to ground station operational principles into account by tracking the orbit of the five satellites in the constellation proposed by the CSA. This orbit is used to generate the current physical status (position, altitude, latitude, longitude, etc.), viewing the satellite's orbit in relation to the Sun, and field of view approximation to estimate if image requests made by user's can be taken or not. To propagate the satellite constellation's physical state information, we use the Skyfield API Python library. This will allow us to use Simplified General Perturbations 4 (SGP4) and to simulate ground tracking from the Earth's surface. We used a formula derived from the paper "Earth Coverage by Satellites in Circular Orbit" by Alan R. Washburn [1]. We also derived the magnitude and simulation of the Earth and Sun using an Ephemeris file derived from NASA's Jet Propulsion Lab (JPL) Planetary and Lunar Ephemerides [6].

2.2.1 FOV Calculation

For calculating the field of view (FOV) ground tracking for the image capture, we applied the following formula to our codebase:

$$FOV = 2 \arctan\left(\frac{12742}{2(Altitude + 6371)}\right) \tag{2.1}$$

This is based on the concept mentioned in the Earth Coverage Paper where it illustrates the cap angle α, the masking angle β, and field of view of 2γ. Assuming an idealized system,

we can arrive to the formula above where the value of 12742 is the diameter of the earth, and 6371 is the radius of the Earth [1]. A visualization of this is shown in the FOV reference sketch.

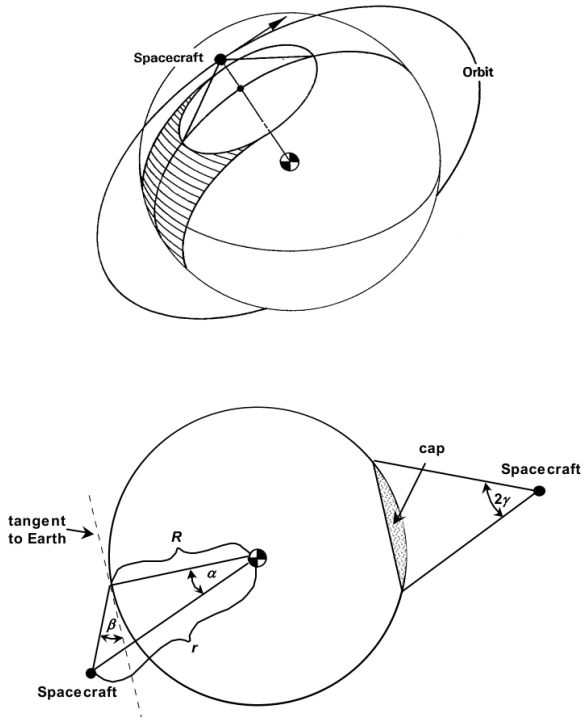
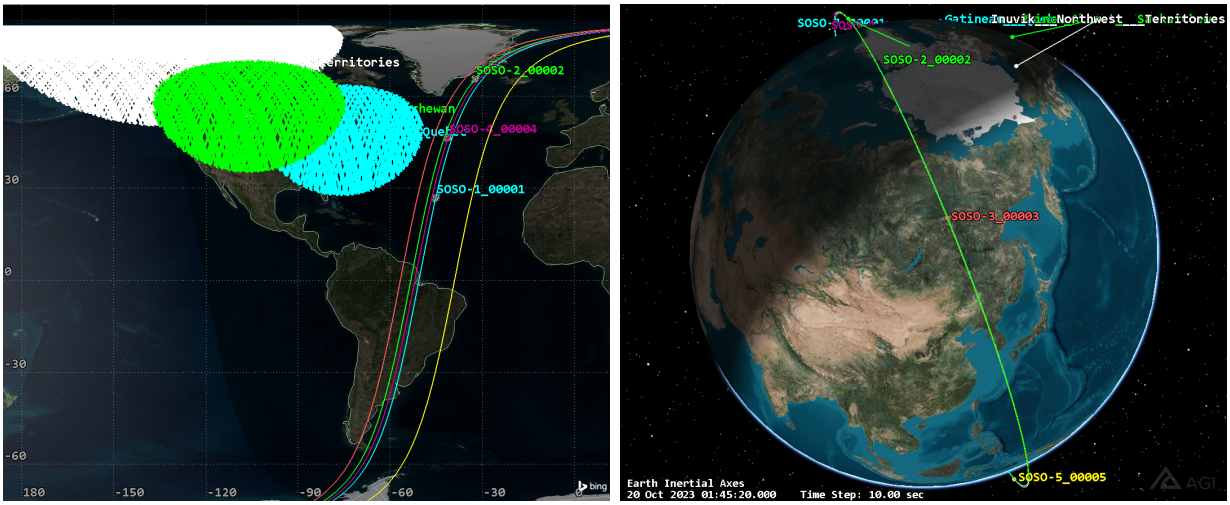


Figure 2.2: FOV Reference image for calculation [1].

The STK was used to simulate the satellite and ground station communications and for testing procedures to better understand if the codebase is in line with realistic satellite to ground station communications as shown in the STK orbit visualization simulations.

2.2.2 General Software & Space Engineering Principles

The software architecture is modeled extensively using universal modeling language (UML) which allows for implementation and programming language independent description of the design. This is a very popular method used in software engineering to communicate design decisions and document what has been built. We contend that we have performed this process at a very rigorous level and this is evident in the vast breadth of UML diagrams



(a) 2D Visualization (b) 3D Visualization

Figure 2.3: STK Orbit Visualization Simulations

presented further ahead in the chapter. These models are present in the form of use-case diagrams, activity diagrams, sequence diagrams, an overall system component diagram, and a database diagram. In addition, we show data-flow pipelines for satellite mapping and scheduling processes, ensuring that the design is replicable.

Moving on, several concepts of web-based software architecture are incorporated to make this design possible. These will be introduced in this section and then described in detail throughout the rest of the chapter. To start, this system uses an event-driven programming approach. In this approach, the server software (also known as the backend) is coordinated by logical events to manage concurrency. This helps decouple the components thereby reducing bugs in the software. It also provides a high threshold of scalability for immense workloads [7]. Using this approach will ensure a reduction of system complexity.

The system components in SatOpsMQ are decoupled microservices (as opposed to monolithic software), a popular approach when building modern software. This is a methodology of developing several small individual modules that run on their own instances. Each microservice does very little on its own but does it very well. Furthermore, microservices offer high developer productivity and high scalability for large workloads [8].

Further advancements in modern software engineering tooling allow for the virtualization of microservices into containers using a tool called Docker (this is described in detail further ahead). This workflow allows for universal compatibility among developer machines

and all major cloud computing providers [9, 10, 11]. Individual microservices will run on separate containers leading to a robust and reliable web application [12].

Moreover, many modern tools allow for event-based message passing between decoupled system components (the microservices in the system). SatOpsMQ is built on top of the Advanced Message Queuing Protocol by incorporating RabbitMQ [13]. RabbitMQ is a commonly used event-messaging bus that serves as a broker for transmitting events between designated endpoints. These endpoints are producer-consumer pairs that produce and consume event messages for real-time system updates [14].

The requirements for this system necessitate this multidisciplinary approach to solving the problem. SatOpsMQ sits at the nexus of advanced methods of space engineering and software engineering techniques. Given this introduction to the concepts used in the system, we now move into describing the details of the system.

2.2.3 Event-Driven Infrastructure

Event-driven architecture is a software design paradigm that emphasizes the production, detection, consumption, and reaction to events. An event is any significant state change or occurrence within a system that is relevant to the software or business process. This architecture is built around the concept of events being the primary drivers of software behavior, rather than traditional request/response flows.

To implement this event-driven architecture, the system uses RabbitMQ. The microservices operate without direct knowledge of each other and instead, RabbitMQ acts as an event broker that manages the connections between them. Each microservice is equipped with a pair of queues – one for consuming events from others (consumer queue) and another for publishing events (publisher queue). RabbitMQ's implementation of topic exchanges plays a critical role here. Topic exchanges are a type of event routing mechanism within RabbitMQ where events are sent to queues based on a topic—a specific routing pattern known as routing key. This mechanism is shown in detail in the Topic Exchange Schematic. This allows microservices to publish events to a topic and only the microservices subscribed to that topic receive those events. For example, a microservice handling image orders might use a routing key like "image.order". This means it will only receive events published with this routing key, effectively filtering the communication to only relevant topics. Subscribing microservices only process events that match their subscribed routing keys, thereby enhancing communication efficiency. Further details on these microservices and their specific functionalities will be covered in the subsequent sections.

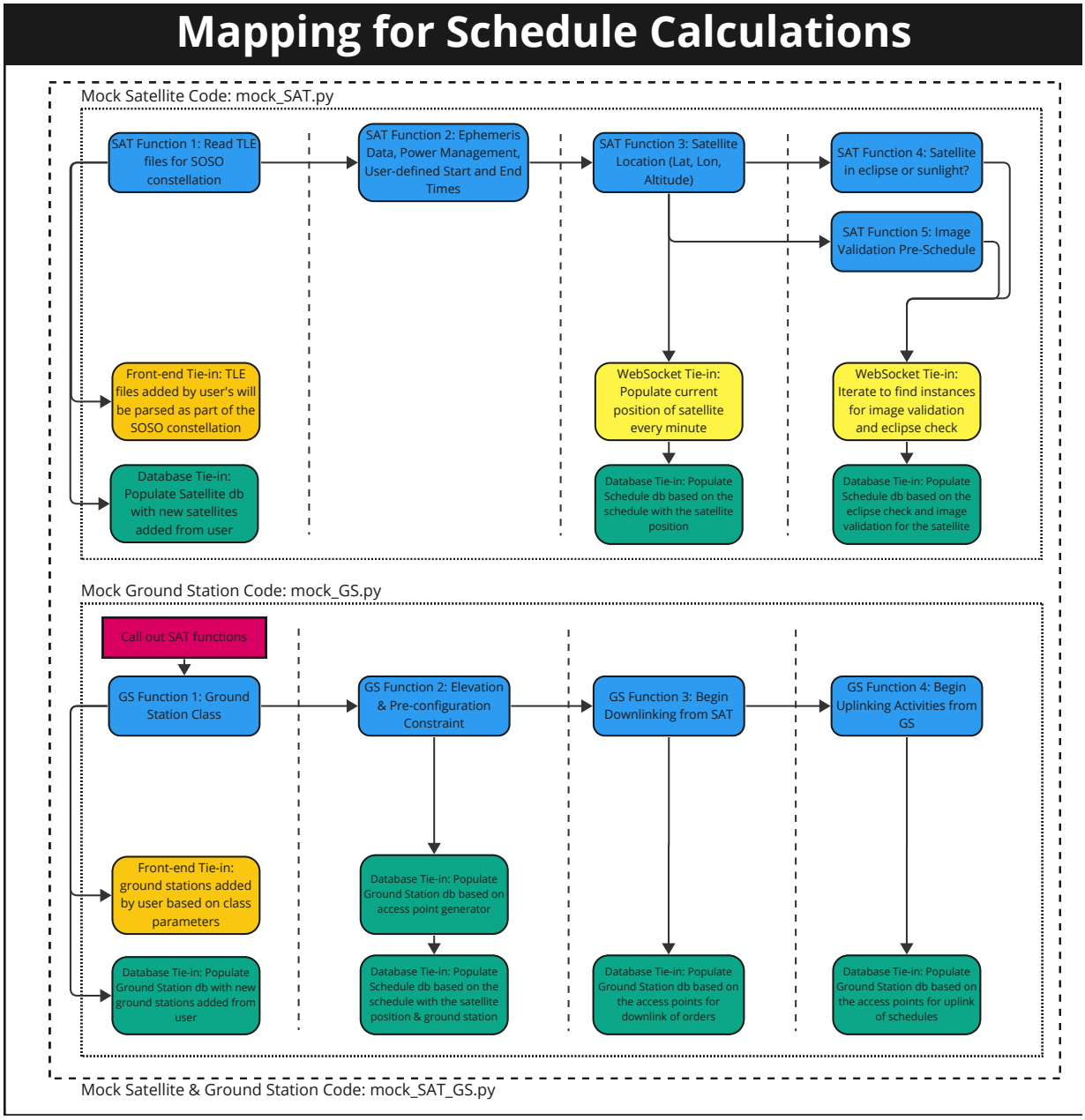


Figure 2.4: Mapping for Schedule Calculations for SatOpsMQ

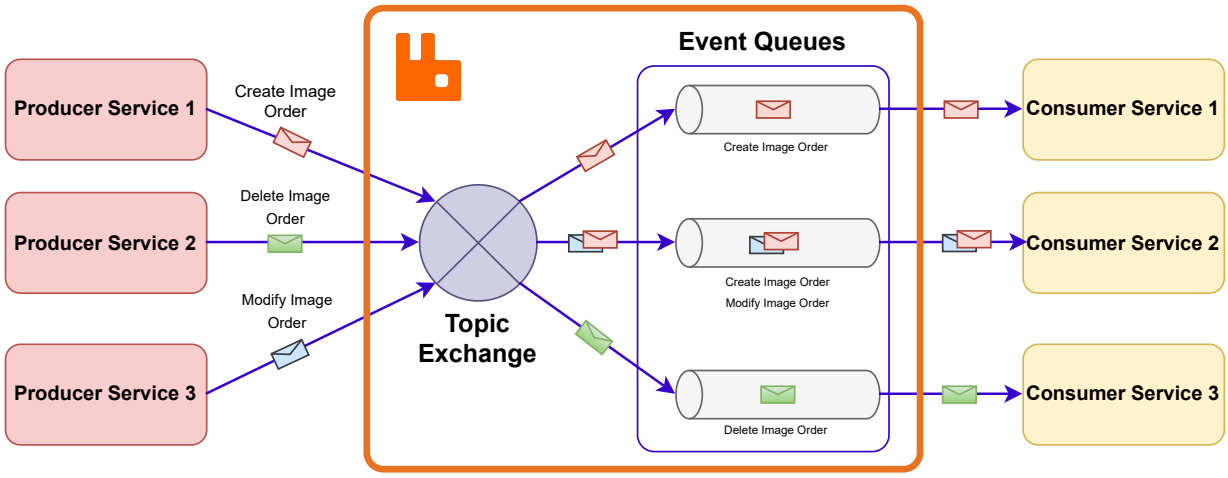


Figure 2.5: Producer-Consumer connections mediated by RabbitMQ topic exchange & event queues.

2.2.4 Client-Server Architecture with REST (Representational State Transfer) Protocol

REST is a list of guidelines and conventions used for the design and development of web-based applications. It outlines a set of constraints that, when followed, yield systems that are scalable, simple, lightweight, and performant. REST is particularly suited for building web services (commonly known as RESTful APIs) that can be easily consumed by diverse clients across the internet. In SatOpsMQ, REST was used to interact with the set of exposed server API endpoints from the client side, forming the basis for the client-server architecture within the system. SatOpsMQ architecture facilitates CRUD (Create, Read, Update, Delete) operations which are essential for managing resources on the backend. Each operation corresponds to and is accessible through the standard HTTP method: POST for creating new resources, GET for reading or retrieving them, PUT for updating existing resources, and DELETE for removing them. These operations ensure that the API can handle all necessary interactions with the system's data in a standardized way. This can be seen in the SatOpsMQ REST figure.

2.2.5 FTP Protocol

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. SatOpsMQ uses

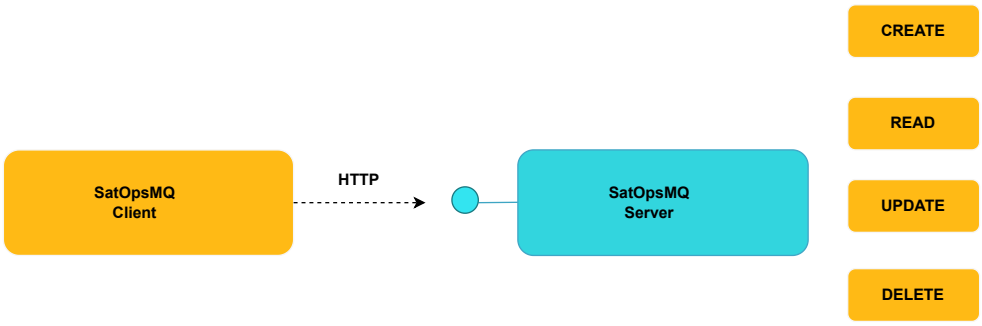


Figure 2.6: SatOpsMQ client and server interaction with REST protocol using the different HTTP verb conventions.

this protocol to fetch the image orders that have been requested by third-party users (e.g. scientists, business personnel, engineers). The backend is scheduled to pull periodically every hour from a third-party file server (every hour). It does this by operating in the traditional active mode of FTP. In this mode, the backend, acting as the client, opens a port and listens, while the third-party file server, functioning as the server, actively connects to it to transfer the requested image files. This is shown visually in the SatOpsMQ FTP diagram.

2.2.6 Dynamic Programming - The Knapsack Problem

Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable where the problem can be divided into overlapping subproblems and can be solved using solutions to those subproblems. A classic example of dynamic programming in action is the Knapsack problem. The idea of the Knapsack problem is to determine the optimal subset of items to pack into a knapsack, such that the total weight does not exceed the knapsack's capacity while maximizing the total value of the packed items. The underlying algorithm that SatOpsMQ uses to schedule satellite imaging, maintenance, and outage requests can be likened to solving a Knapsack problem. In SatOpsMQ, the "knapsack" is the available time slots and bandwidth on the satellite or ground stations. These resources have a finite capacity similar to the weight limit of a knapsack. Each satellite operation (imaging request, maintenance task, outage response) represents an "item" to be placed in the knapsack. These tasks have "weights" (resource requirements like time and bandwidth) and "values" (priority or importance of the task). The visualization of this can be seen in this figure.

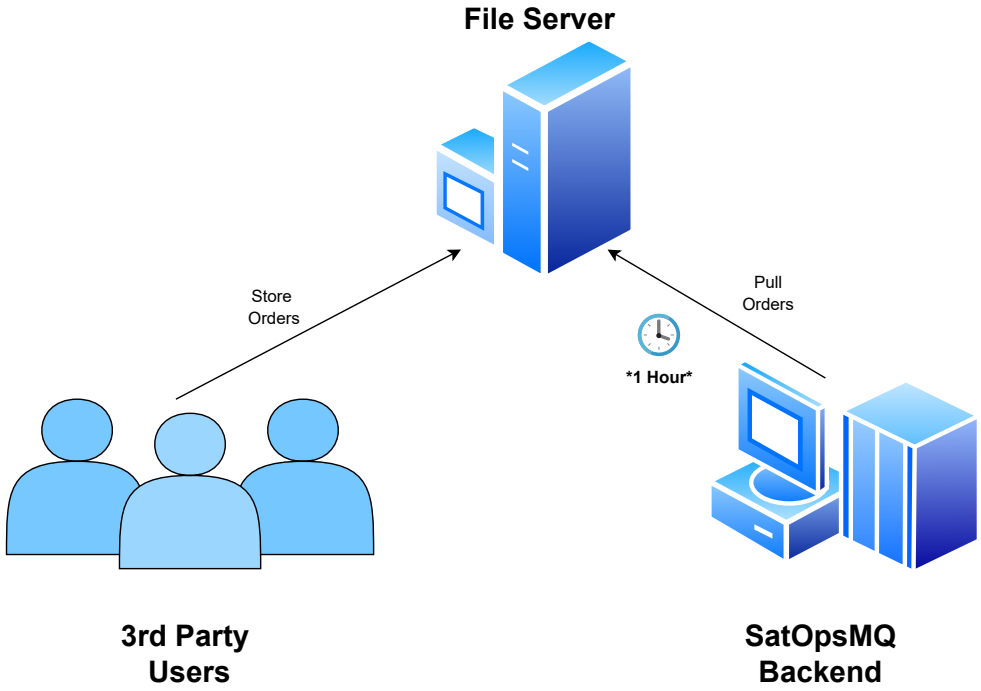


Figure 2.7: The periodic interaction of SatOpsMQ to fetch orders from file server via FTP protocol

2.2.7 Heuristic Solutions

Heuristics are strategies that are designed to find approximate or "good-enough" solutions within a reasonable time frame, often at the cost of not guaranteeing the optimal solution. The SatOpsMQ algorithm offers heuristic solutions to avoid enumerating the vast number of possible scheduling options due to timing constraints by user requests. This allows SatOpsMQ to quickly generate viable schedules that, while not guaranteed to be optimal, meet essential criteria and maximize overall operational effectiveness within resource constraints. This helps solve the core challenge of the Knapsack problem which is to maximize the total value of items packed into a limited capacity. Translating this to SatOpsMQ, the heuristic solutions prioritize tasks (items) that offer the highest value relative to their resource demands (weight) which provides a method to arrive at good solutions in a fraction of the time it would take to compute the optimal solution.

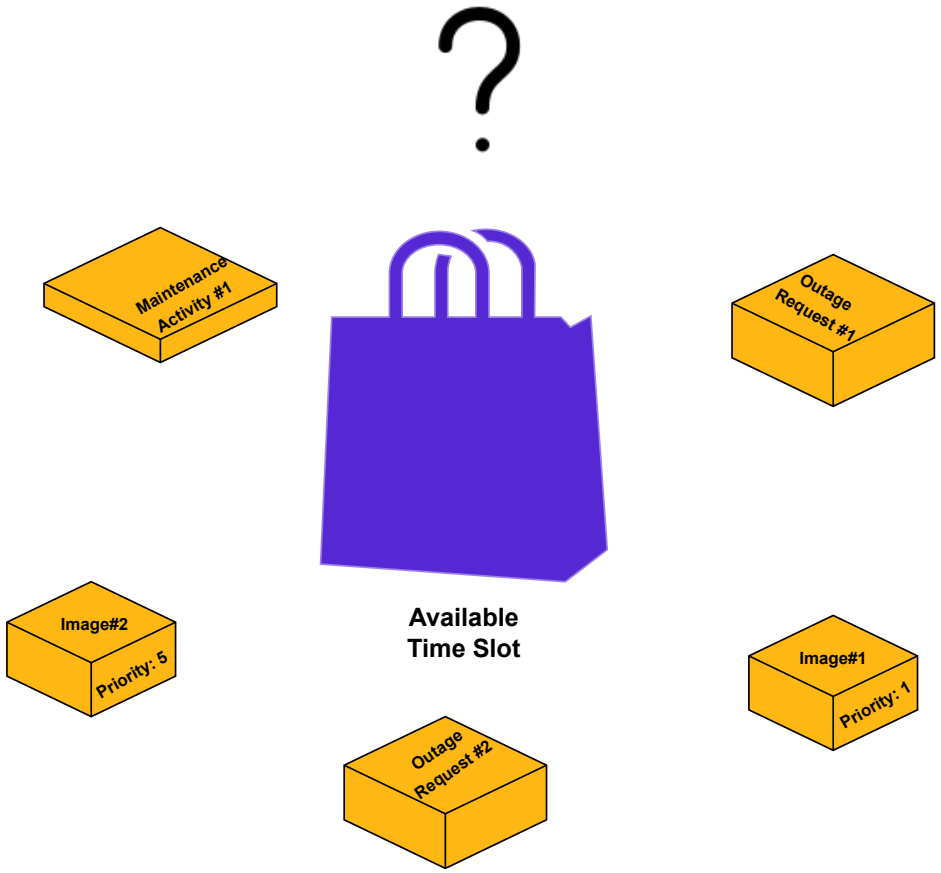


Figure 2.8: Visualization of the Knapsack Problem from SatOpsMQ perspective

2.2.8 Greedy Algorithm

A greedy algorithm is a heuristic method that makes the locally optimal choice at each stage with the hope of finding a global optimum. SatOpsMQ's algorithm uses a greedy approach by actively seeking and adopting better-performing schedules as they become available. This strategy is based on continuous evaluation against specific performance metrics, such as resource utilization efficiency, cost reduction, and improved service delivery timelines. In a very high-level view, this could be understood by the following pseudocode.

Algorithm 1 Dynamic Schedule Optimization

```
Initialize:
current_schedule ← initial_feasible_schedule
performance_metrics ← evaluate(current_schedule)

Loop:
while changes in task priorities or resource availability do
    possible_schedules ← generate_potential_schedules(current_schedule)
    for each schedule in possible_schedules do
        metrics ← evaluate(schedule)
        if is_better(metrics, performance_metrics) then
            current_schedule ← schedule
            performance_metrics ← metrics
            break
        end if
    end for
end while

function GENERATE_POTENTIAL_SCHEDULES(current_schedule)
    new_schedules ← empty list
    for each task in tasks do
        if task's priority has changed or new task added then
            temp_schedule ← update_schedule(current_schedule, task)
            new_schedules.append(temp_schedule)
        end if
    end for
    return new_schedules
end function

function IS_BETTER(new_metrics, current_metrics)
    return (new_metrics.efficiency > current_metrics.efficiency) or
    (new_metrics.cost < current_metrics.cost) or
    (new_metrics.timeline < current_metrics.timeline)
end function
```

2.3 As-built System Configuration

2.3.1 Technology Stack

The architecture of the system is comprised of a versatile technology stack that were carefully chosen to meet our performance and scalability requirements. Below is a table that summarizes our selections and the subsequent sections will delve into the rationale as to why each were chosen.

Category	**Technologies and Utilization**
Backend Technologies	Python for TLE calculations and backend services, Node.js for FTP server
Frontend Technologies	React.js for dynamic UIs, Next.js for server-side rendering, Cesium.js for geospatial visualizations
Deployment and Containerization	Docker for containerization, AWS EC2 for hosting, Tauri for desktop deployment
Database Management	PostgreSQL for database management

Table 2.1: Overview of Technologies Used in the Satellite Operations Management System

2.3.1.1 Backend Technologies

The backend infrastructure employs a dual-language combination:

- **Python** - Chosen primarily for its powerful Skyfield library, which provides us with the precision and capabilities required for TLE (Two-Line Element Set) propagation calculations. Python's straightforward syntax and readability make it an excellent choice for creating clean, maintainable code that drives several of our backend services.
- **Node.js** - Our FTP server is implemented using Node.js, capitalizing on its non-blocking, event-driven architecture.

2.3.1.2 Frontend Technologies

The frontend stack comprises several advanced technologies tailored for efficient and dynamic user interface development:

- **React.js** - Serving as the backbone of our frontend applications, React.js enables us to build a dynamic and highly interactive user experience with its component-based architecture.
- **Next.js** - We use the Next.js framework for its server-side rendering and static site generation capabilities, which paved the way for easy application deployment.
- **Cesium.js** - For geospatial visualizations, Cesium.js provides powerful 3D globe and map views which are integral to our satellite tracking and asset monitoring features.

2.3.1.3 Deployment and Containerization

Our deployment strategy involves containerization and cloud computing:

- **Docker** - Docker containers encapsulate our application's runtime environment, ensuring consistency across development, testing, and production environments.
- **AWS EC2** - We utilize AWS EC2 for hosting our containerized services, taking advantage of its scalable compute capacity. EC2's flexibility allows us to respond efficiently to varying workloads and traffic.
- **Tauri** - For desktop application deployment, we use Tauri, which utilizes Rust's performance and security strengths to build lightweight and secure cross-platform applications.

2.3.1.4 Database Management

Database management is critical for maintaining data integrity and performance:

- **PostgreSQL** - Our database of choice is PostgreSQL, which offers advanced features and strong community support essential for complex query operations and data integrity.

2.3.2 Docker Configuration

SatOpsMQ uses Docker to deploy its backend microservices. The deployment process relies on two critical configuration files

- **DockerFile:** This file contains all the commands a user could call on the command line to assemble an image. It defines the Docker image and specifies the operating environment, dependencies, and the setup steps necessary for the microservices.

- **docker-compose.yaml:** This file is used to configure the services that make up the application so they can be run together in an isolated environment. It simplifies the configuration of multiple containers, defining how these containers interact, network settings, volumes, and other dependencies.

It is crucial to follow these configurations as-is to deploy the backend correctly.

2.3.2.1 Docker-Compose: Services Configuration

Each microservice container has a dedicated port. The event-relay-API service serves as the central communication hub and is configured to expose port 80 and route internal traffic to port 5000. It depends on RabbitMQ and PostgreSQL services for message queuing and data persistence respectively. The scheduler service, being responsible for the order scheduling, is exposed on port 81 and linked to the same dependencies as the Relay API. It is built from a Dockerfile located within the scheduler service directory. The Satellite Activities Service handles the orchestration of satellite-related operations and is available on port 82. Lastly, the scheduler worker executes the scheduled task, with Celery for tasks queuing, and operates in conjunction with the Scheduler service where they use the same Docker image. These services configuration are declared within the docker-compose.yaml.

Service	**Port**	**Dependencies and Notes**
Relay API Service	80, 5000	Depends on RabbitMQ and PostgreSQL.
Scheduler Service	81	Shares dependencies with Relay API.
Satellite Activities Service	82	Uses common data handling with other services.
Scheduler Worker	Uses Scheduler's port	Works with Scheduler Service, uses Celery.

Table 2.2: Summary of Docker Configurations for System Services

2.3.2.2 Docker-Compose: Supporting Services

The SatOpsMQ uses three supporting services in conjunction with its microservices. RabbitMQ was employed as the message broker, facilitating communication between services with management exposed via port 15672 and service communication through port 5672.

PostgreSQL serves as the primary database, set to restart consistently with environment variables defining the root user, password, and database specifics. Port 5400 is designated for database connections, with a named volume for data persistence. The pgAdmin provides a web-based database management interface, utilizing port 85, and is dependent on the PostgreSQL service. It also has its own named volume similar to PostgreSQL. At the time of the final release, all supporting services were configured to use the latest images provided by Docker Hub. These configurations are declared within the docker-compose.yaml file along with the configuration of the main services.

Service	**Port**	**Details and Dependencies**
RabbitMQ	15672, 5672	Acts as the message broker, managing service communication.
PostgreSQL	5400	Main database, automatically restarts; uses environment variables for configuration. Volume used for data persistence.
pgAdmin	85	Web interface for PostgreSQL management, requires PostgreSQL service. Configured with default credentials.

Table 2.3: Summary of Supporting Services Configurations

2.3.2.3 DockerFile Configuration

Each microservice uses a consistent Dockerfile structure (this is an example from the scheduler service). This approach ensures uniformity in the build process and simplifies maintenance. The Dockerfile for each service performs the following steps.

1. **Base Image:** We initiate our Dockerfile by pulling the lightweight `python:3.11-alpine` image, chosen for its minimal footprint while providing the necessary Python environment.
2. **System Dependencies:** The container is updated and Git is installed, enabling us to manage code versions and dependencies directly within the container.
3. **Python Environment Setup:**
   (a) Pip, the Python package installer, is upgraded to ensure we can install the latest versions of required libraries.

4. **Requirements Installation:**
   (a) The `requirements.txt` file, containing base requirements common to all services, is copied into the container and used to install essential Python packages.
   (b) A service-specific `requirements.txt` is conditionally copied and installed if it exists, allowing for service-specific dependencies to be managed separately.
5. **Application Configuration:** Application configuration scripts and files are copied to a designated directory within the container. These scripts are executed to integrate the configuration into the Python environment, ensuring the application runs with the correct settings.
6. **Service Code:** The source code of the satellite activities service is then copied into its respective directory within the container.
7. **Final Setup:**
   (a) The working directory is set to the service's code directory.
   (b) The default command to run the microservice (`main.py`) is defined.

2.3.2.4 Desktop Deployment

The deployment of our desktop application is streamlined through the Rust package manager, Cargo, enabling us to execute builds with the cargo tauri build command. To ensure a successful build and deployment, the following steps must be followed:

1. **Next.js Configuration:** Modify the next.config.js file to specify the output setting for static export, crucial for generating the static files Tauri requires, which are placed in the "out" folder. The configuration file must look like below:

```
1   // next.config.js setup for static export
2   const path = require("path");
3   const nextConfig = {
4     reactStrictMode: true,
5     output: 'export',
6     sassOptions: {
7       includePaths: [path.join(__dirname, "styles")],
8     },
9     transpilePackages: ["vis-timeline", "vis-data", "vis-util", "
    @mui/x-charts"],
10  };
11  module.exports = nextConfig;
12
```

2. **Deployment Script:** Construct a deployment script that establishes custom managers, menu items, and system tray functionalities for the desktop user interface. The `tauri::Builder` is invoked to assemble the application with all the predefined components, ensuring a cohesive user experience.
3. **Installation of Prerequisites:**
   (a) Ensure Rust is installed on the system. Follow the official Rust installation guide.
   (b) Verify the presence of C++ build tools, which are typically included in the Rust setup. If absent, refer to the build tools guide.
4. **Tauri CLI and Build Execution:**
   (a) Install the Tauri CLI using Cargo with the command `cargo install tauri-cli` and ensure its availability in NPM.
   (b) Generate the executable file by running `cargo tauri build`. This command implicitly performs `npm run build`, preparing all necessary frontend assets before packaging them with Tauri.

2.3.2.5 AWS Backend Cloud Configuration

Our backend server is provisioned using Amazon Web Services (AWS) with an EC2 instance finely tuned to support our operational requirements. This table is a detailed configuration that outlines the essential parameters set up for our backend infrastructure.

The EC2 instance identified by Instance ID i-0fc2bc8d8e4498e0c, designated as 'Backend-Server,' is configured with a t2.micro instance type, which is selected for its cost-effectiveness and adequate performance capabilities suitable for backend services. The instance is currently at an active and operational status. Network connectivity is established via a public IPv4 address (44.204.157.210) for external access alongside a private IP address (172.31.89.105) that ensures secure internal communications within the AWS environment. The public IPv4 DNS, ec2-44-204-157-210.compute-1.amazonaws.com resolves the public IP address and aids in network identification and accessibility.

The instance is located in the us-east-1a availability zone to minimize latency and maximize availability for the primary user base. Currently, security measures can be seen with absence of a specified IAM role. This is because delegating specific IAM roles, which are designed to manage permissions and control access to AWS resources, extends beyond the scope of the free tier services provided by EC2. The system's monitoring utilizes AWS's default options with the capability for enhanced monitoring should detailed insights be required.

Category	Details
Instance Configuration	Instance ID: i-0fc2bc8d8e4498e0c (Backend-Server) Instance Type: t2.micro Instance State: Running
Networking Setup	Public IPv4 Address: 44.204.157.210 Private IP Addresses: 172.31.89.105 Public IPv4 DNS: ec2-44-204-157-210.compute-1.amazonaws.com
Location and Accessibility	Availability Zone: us-east-1a
Security and Monitoring	IAM Role: Not specified Monitoring: Default monitoring
Storage and Performance	AMI ID: ami-00ebbc213c0308284 Virtualization Type: HVM
Resource Management	VPC ID: vpc-035f00acb9b0099d1 Subnet ID: subnet-0d6a7faeb3e72f772
System Resilience	Capacity Reservation: Open Auto-recovery: Default setting
Amazon Machine Image (AMI)	AMI ID: ami-00ebbc213c0308284 ('SOSO-image') AMI Details: Last launched on March 30, 2024. Linux/UNIX, x86_64 architecture Boot Mode: UEFI preferred
Elastic Block Store (EBS) Volumes	Volume Configuration: 8 GiB gp2 EBS volume (vol-0bd09ba5863a17df0), baseline of 100 IOPS Snapshot Management: Snap-0d8e7433f1669bbc7

Table 2.4: Details of the AWS setup

Storage and performance are managed through AMI ID ami-00ebbc213c0308284, which serves as a pre-configured starting image for the instance where it employs HVM virtualization to optimize performance. The instance operates within VPC ID vpc-035f00acb9b0099d1 and is part of subnet-0d6a7faeb3e72f772 to enhance network isolation and management. System resilience is supported by an open capacity reservation policy and default auto-recovery settings.

The AMI, termed 'SOSO-image' for the instance, supports the deployment and scaling of our EC2 instances. It encapsulates a Linux/UNIX operating system with x86_64 architecture to adapt to various applications.

Storage solutions are implemented using an 8 GiB gp2 EBS volume (vol-0bd09ba5863a17df0) which is observed with an active baseline performance of 100 IOPS. Data integrity and recovery strategies are supported by snapshot management, with the current snapshot (snap-0d8e7433f1669bbc7) safeguarding the data which provides crucial recoverable points for enhanced data durability.

2.3.3 Security Settings

The backend utilized CORS (Cross-Origin Resource Sharing) middleware. CORS is a security feature that controls how the backend responds to requests from different origins, typically used to restrict which websites can interact with the backend. For our local development, CORS settings were adjusted to specifically allow secure communication such that only the frontend's localhost is capable of using the server's APIs. This setup was key to mimicking a real-world scenario where the frontend and backend, hosted on different domains, need to interact safely and efficiently. An example of the CORS configuration is shown in the CORSMiddleware listing below.

In the development setup, environment variables stored in a .env file plays a crucial role in managing the system's configuration securely. These environment variables are essentially external parameters set outside the application code. They are used to store sensitive information such as queue and database URIs, and other configuration details that should not be hard-coded into the source code for security reasons. By keeping this information in a .env file, developers can quickly change configuration settings (like switching database connections) without needing to modify and redeploy the application code itself. This approach significantly enhances security, as sensitive details are not exposed in the code repository, and also provides flexibility, allowing for swift adjustments to the application's configuration as needed, without impacting the core codebase. This method is a standard practice in software development, ensuring that the application can be easily

and safely configured across different environments, from local development to production. An example configuration of the can be seen in the sample .env listing below.

```
    app.add_middleware(
        CORSMiddleware,
        allow_origins=["http://localhost:3000"],  #Replace with the correct
    frontend origin
        allow_credentials=True,
        allow_methods=["*"],  #Or restrict to ["GET", "POST"], etc.
        allow_headers=["*"],
    )

```

Listing 2.1: CORS Middleware configuration example

```
        #Rabbit Settings
        RABBIT_HOST = localhost
        RABBIT_PORT = 5672
        RABBIT_USER = guest
        RABBIT_PASS = guest
        RABBIT_VHOST = /

        LOG_LEVEL = debug
        LOG_FORMAT = '%(asctime)s:%(name)s: %(message)s'

        #Database Settings
        DB_HOST = localhost
        DB_USER = postgres
        DB_PASS = postgres
        DB_NAME = soso_db
        DB_SCHEMA = soso_schema
        DB_DRIVER = postgresql+psycopg2

        #Message Queues for direct communication
        SCHEDULER_QUEUE = scheduler
        SAT_ACTIVITIES_QUEUE = satellite_activities
        RELAY_API_QUEUE = relay_api

```

Listing 2.2: .env file configuration example

2.4 As-Built Design

This project includes two main components: a client application, and a backend server. Their codebases are stored on GitHub publicly to establish the transparency and accessibility of the open-source requirement. The server is divided into different Python-based microservices and a Postgres database that are designed to run locally using Docker which allows it to connect to different ports within the same local environment. Dockerization facilitates the isolation of these components and makes it easy for each to operate in a self-contained environment with its dependencies and configurations. Docker containers are notably lightweight because they run on the same operating system as the host machine, and they share the kernel. This shared kernel architecture means that Docker containers do not require the overhead of a full operating system, leading to more efficient use of system resources like CPU and memory. Additionally, this contributes to faster start-up times for containers compared to traditional virtual machines. In other words, by running in Docker containers, these components can be easily scaled, updated, and deployed, which is vital for different operations that require robustness and reliability [9, 11, 10].

The client application was developed using the Next.js framework and is executable locally without docker containers. This approach was adopted for two main reasons. Firstly, the intent for the final release was to create a desktop application out of the client code. This requires server-side rendering and static site generation, all of which are facilitated very easily with the Next.js framework compared to other alternatives because it allows for the generation of the required static files by just running one command and one JSON configuration file [15]. Secondly, front-end development often benefits from more direct interaction with the local environment for faster feedback and easier debugging. Running the client application directly on localhost allows for quicker iterations and real-time updates, which are essential in the development of immediate visual and functional feedback. Furthermore, since front-end applications typically have fewer complex dependencies compared to back-end services, the need for the controlled environment provided by Docker is less critical and can be considered unnecessary. This setup enables efficient and isolated testing of the system's functionalities in its early development stage and simplifies the further evolution of the project.

For the final release, the selection of deployment tools was carefully made to minimize the number of setup steps and potential errors as these factors could be detrimental to the overall project's success. The backend server components were deployed on AWS EC2. This decision was driven by the need for stability and reliability, particularly during live demonstrations and production deployment [16]. Hosting on AWS EC2 minimizes the risk of unexpected hiccups by offloading the responsibility of server management to the

cloud. The server was made accessible through an AWS-generated IP address. Despite being hosted in the cloud, it should be noted that the SatOpsMQ backend system is still capable of running locally, albeit with more steps and additional configurations. The client application was packaged using Tauri [17], a lightweight Rust-based framework, to create the executable desktop application. Tauri stands out for its minimal footprint and ability to produce smaller and more optimized applications compared to traditional frameworks like ElectronJS by not relying on heavy number of dependencies or extensive configurations [18]. By using Tauri, the application benefits from Rust's memory safety guarantees and concurrency management which significantly reduces the risks associated with common software vulnerabilities. The decision to use Tauri also reflects a commitment to user privacy and data protection. Unlike some other frameworks that may require more extensive permissions or the inclusion of telemetry, Tauri applications can operate with minimal permissions and do not send user data without explicit consent. This approach not only respects user privacy but also aligns with modern standards for software development and distribution. The architecture of this fully deployed project is displayed by this figure.

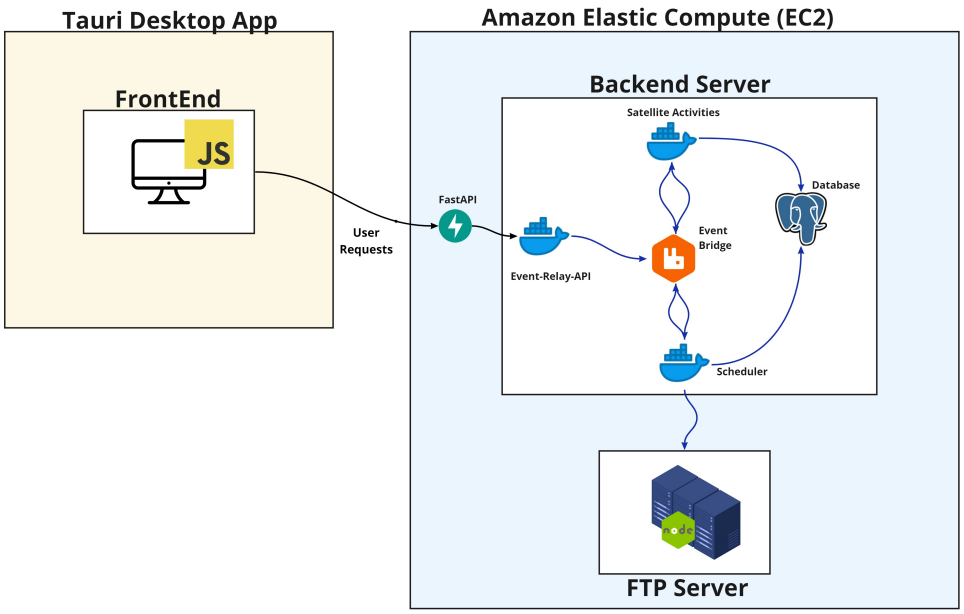


Figure 2.9: Final Architecture of SatOpsMQ for Deployment

2.4.1 Project Design Revisions & Rationale

2.4.1.1 SatOpsMQ 1.1 - Cloud Based

The initial architecture of SatOpsMQ V1.1 was designed as an integrated suite where the backend server functioned as the system's core which interfaced directly with a JavaScript-based frontend and a mocked connection to satellite ground station communications. This backend server housed several key microservices: a Scheduler to manage task timings, an Event-Relay-API to handle event-driven operations, Ground-Station-Outbound which is responsible for sending the schedules to the mocked assets, and separate modules for Satellite-Activities and Image-Management. Each of these modules interacted with its respective database table for storing and retrieving relevant data. An Event Bridge was employed to facilitate communication between services. However, this version was ultimately deemed too dependent on proprietary cloud services by our stakeholders which prompted a shift towards a more open-source solution that could operate in on-premises servers.

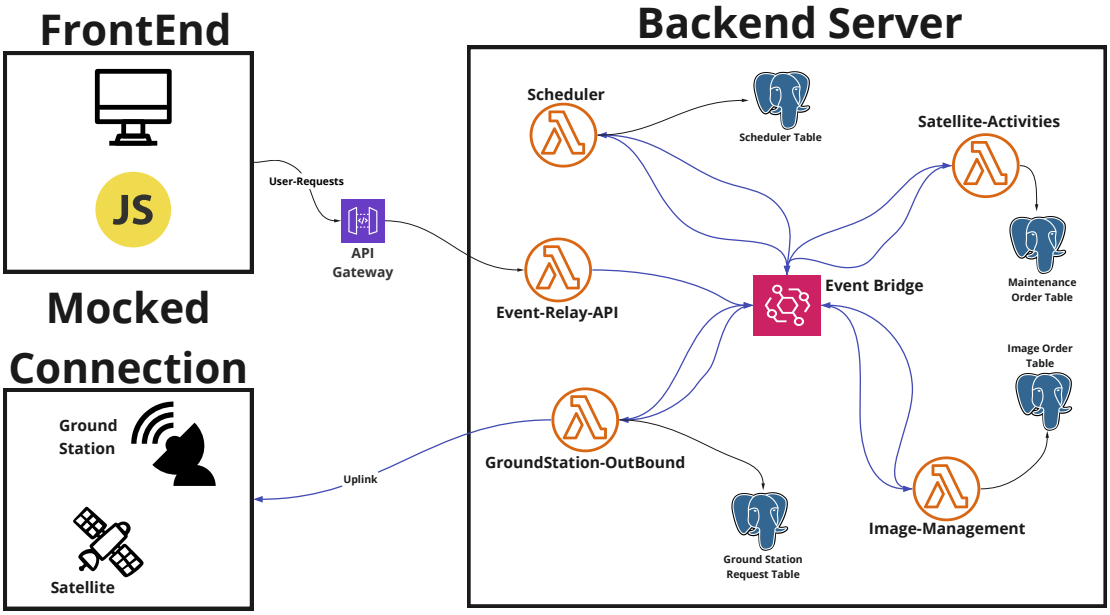


Figure 2.10: SatOpsMQ V1.1 - Amazon Web Services Lambda Serverless Architecture

2.4.1.2 SatOpsMQ 1.2 – Django

The SatOpsMQ 1.2 architecture iteration marked a transition to containerization with Docker and a switch to RabbitMQ for event handling, stepping away from AWS-specific services to embrace more open-source technologies. This version also introduced the Django REST framework to structure the backend's API services. Despite these improvements, this architecture version was also ultimately rejected. The primary concern was that Django's full-stack nature added unnecessary weight to the system. It was found quickly by the team through early testing that its heavyweight infrastructure led to bloated memory usage and slower response times which is problematic in an environment that requires swift processing of satellite data. For example, the overhead from Django's Object-Relational Mapping (ORM) system, though beneficial for complex queries, proved to be excessive for the simpler database interactions typically needed. The framework's lengthy startup process also conflicted with the need for rapid testing in response to the variable demands of satellite operations. The preference for a monolithic structure within Django also meant that inter-service dependencies could become complex which conflicted with our intent of maintaining individual microservices. Django was also deemed too resource-intensive for the lightweight needs of a microservices architecture, which required nimble and rapid processing. Furthermore, while Django offers rich functionality, it was considered overkill for SatOpsMQ's specific requirements, where a simpler and more focused framework could achieve the desired outcomes with greater efficiency. This feedback led to a reconsideration of the technology stack in pursuit of a solution that would better align with the project's goals for an optimized, open-source, and lightweight system.

2.4.1.3 SatOpsMQ 1.3 – FastAPI

In version 1.3 of the SatOpsMQ architecture, the backend was restructured to utilize FastAPI. This change was implemented to address performance bottlenecks identified and it aimed at enhancing the system's responsiveness and efficiency. While concurrent processing was not utilized, the move to FastAPI brought several other advantages. For instance, FastAPI's compatibility with modern Python asynchronous features meant that the system could maintain high throughput during operations which was an important addition as the CSA stakeholders mentioned that it is expected for our system to handle 500 simultaneous orders. An illustrative advantage here is the improved handling of a high volume of concurrent data retrieval requests from the satellite, even if the processing of these requests is not concurrent. The system could queue up requests efficiently and serve them without blocking thanks to FastAPI's event loop. Another significant benefit was

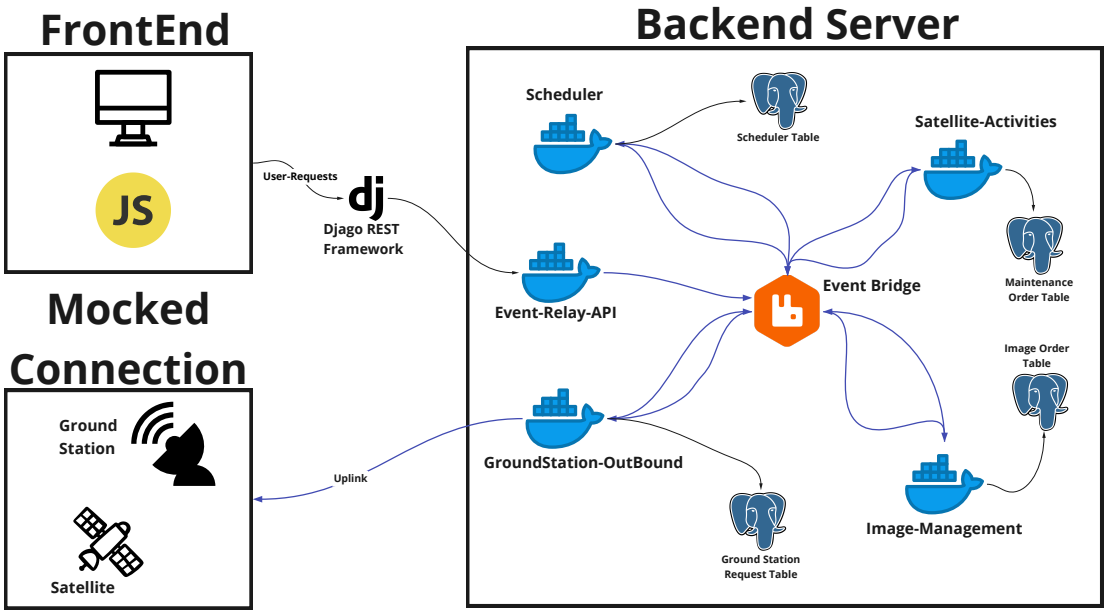


Figure 2.11: SatOpsMQ V1.2 - Microservices Architecture Using Django Rest Framework

FastAPI's compatibility with Celery, an asynchronous task queue/job queue system based on distributed message passing. By integrating Celery, SatOpsMQ was able to handle long-running and scheduled tasks more effectively, especially for batch orders that spanned longer timeframes. This architecture was the one that supported the as-built MVP design of the system.

2.4.1.4 SatOpsMQ 1.4 – Deprecation of Microservices

Near the project's MVP date, the stakeholders mentioned that mocking the ground station connection was not needed. This prompted the team for a slight architectural rework for version 1.4 where the mocked ground station and ground station outbound microservices were deleted. A new requirement was brought up where image orders were not to be sent directly via the user interface but rather through the backend API or a third-party FTP server. Thus, the architecture now incorporates a Node.js programmed FTP server to manage the third-party user's image orders. This scheduler service interacts with this server, which is configured to check for new image orders hourly. This modification deleted the Image Management microservice as its preprocessing role was delegated and transferred

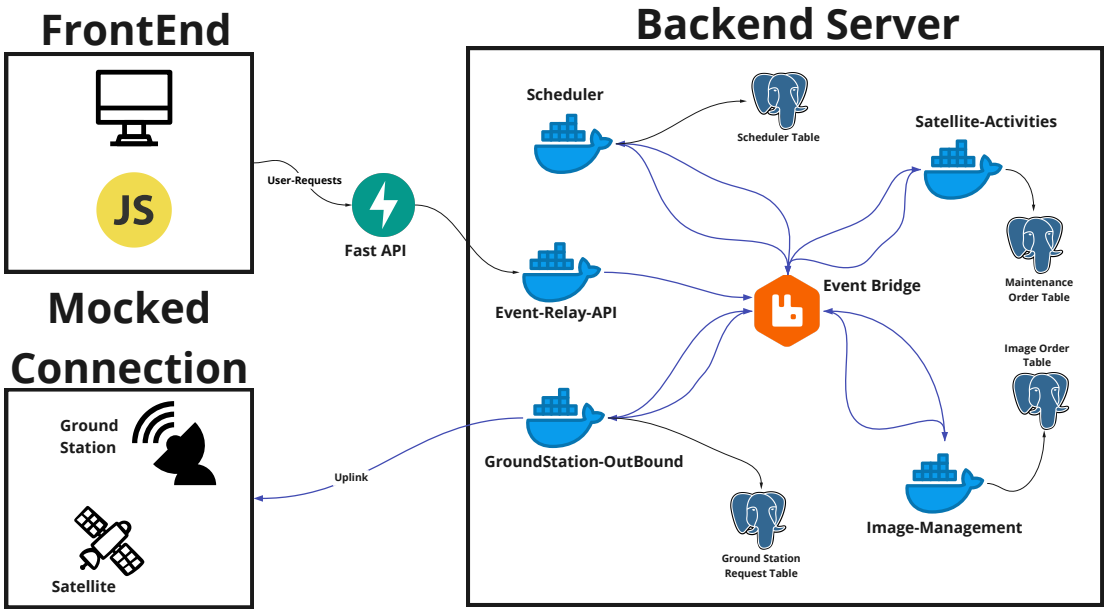


Figure 2.12: SatOpsMQ V1.3 - High Level Architecture

to the Scheduler.

The trade study between the four iterations of the architectural diagrams can be found below. The scales represent ascending rank. A score of 1 indicates the least amount of deployability, scalability, and compatibility. Equations (2.2)-(2.4) show the methodology used to determine the best solution.

Architecture	**Weight (%)**	**Scale**	**V1.1**	**V1.2**	**V1.3**	**V1.4**
Deployability	30	1-5	5	3	4	3
Scalability	30	1-5	5	5	5	5
Compatibility	40	1-5	2	4	4	5
Weighted Totals in %	100%		76%	80%	86%	85%

Table 2.5: Trade Study for Architectural Solutions for SatOpsMQ

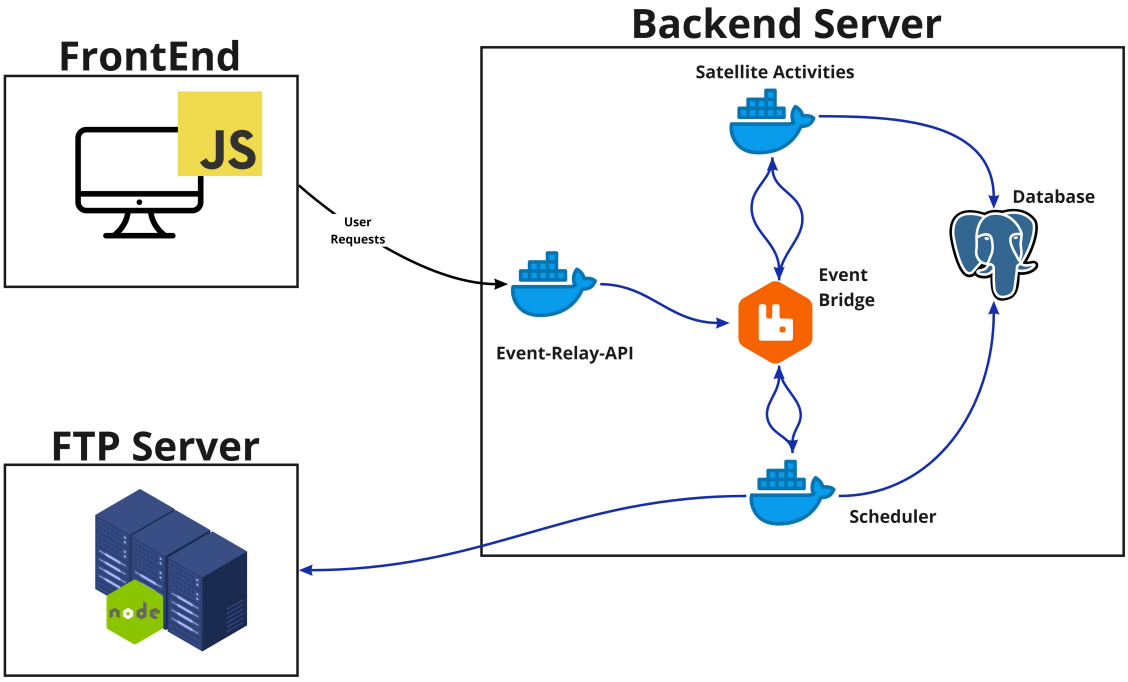


Figure 2.13: SatOpsMQ V1.4 - High Level Architecture

$$\text{Weighted Score} = \text{Score of Option} \times \text{Weight of Criterion} \tag{2.2}$$

$$\text{Maximum Possible Score} = \text{Maximum Score on Scale} \times \text{Weight of Criterion} \tag{2.3}$$

$$\text{Percentage} = \left(\frac{\text{Weighted Score}}{\text{Maximum Possible Score}} \right) \times 100\% \tag{2.4}$$

The weights used for each of these factors in the trade study is based on the user needs for the web application. For example, compatibility is the most weighted factor in the trade study because the users should be able to pull the code base and modify the code to suit their needs. As for scalability and deploy-ability, these are factors that are used to prove the usability of the web application to the user.

Finally, through our analysis of the four solutions we developed for the users, we found that the third solution architecture using a micro-services architecture with dockerization

was the most suitable for allowing the user to change any parameters they needed for the web application once it was complete. Although, the weighted total of version 1.4 was slightly lower than version 1.3, this discrepancy is negligible and thus version 1.4 was adopted for the final release to fulfill the stakeholder's adjusting requirements.

2.4.2 Inputs

Detailed tables showing the descriptions of these inputs are shown in Appendix C.1 Input Parameter Data Descriptions

Image Requests: These specify the desired location, timing, and type of satellite images. An example is shown below:


```
{
   "Latitude": -17.83012427765297,
   "Longitude": 29.92473979644751,
   "Priority": 3,
   "ImageType": "Medium",
   "ImageStartTime": "2023-11-18T05:09:04",
   "ImageEndTime": "2023-11-18T17:52:01",
   "DeliveryTime": "2023-11-19T01:52:01",
   "Recurrence": [{"Revist": "False"}]
}
```


The image request parameters define the criteria for satellite imaging tasks. Each parameter specifies key details that inform the scheduling and execution of image capture, ensuring that user requests are met with precision and efficiency. The following table elucidates each parameter and its role in the image request process.

Satellite Activities: These include various maintenance tasks required for satellite operation. A sample input is as follows:


```
    {
      "Target": "SOSO-3",
      "Activity": "OrbitParameterUpdate",
      "Window": {
        "Start": "2023-10-08T00:00:00",
        "End": "2023-11-15T23:59:59"
```

```
    },
    "Duration": "30",
    "RepeatCycle": {
      "Frequency": {
        "MinimumGap": "518400",
        "MaximumGap": "691200"
      },
      "Repetition": "4"
    },
    "PayloadOutage": "FALSE"
  }
```


The satellite activities parameters encompass the various tasks necessary for maintaining satellite health and optimal operation. These parameters guide the scheduling of routine checks and adjustments vital to the satellite's performance. Below is a detailed description of each activity type and its significance.

Maintenance tasks are categorized into different activities such as Payload Diagnostic Activity Maintenance, Orbit Parameter Update, Orbit Maintenance, and Memory Scrub.

Outage Requests: These requests specify periods when a satellite or ground station is not available. An example is:


```
{
   "Target": "SOSO-1",
   "Activity": "outage",
   "Window": {
     "Start": "2023-10-08T00:00:00",
     "End": "2023-11-15T23:59:59"
   }
}
```


The outage request parameters are critical for managing periods when satellites or ground stations are unavailable due to maintenance or other operational considerations. The parameters outlined in the table ensure that these outages are accurately reflected in the scheduling system, minimizing disruptions to imaging and communication activities.

2.4.3 Outputs

Upon submission of any type of request, whether for imaging, maintenance, or outages, the system generates a unique identifier (ID). This ID is used for tracking and will be referenced for viewing in the corresponding tables. For instance, a submitted image order will receive an ID such as "ID 1", which can then be used to track the status and details of that particular order. This feature ensures that users can easily monitor and manage their requests throughout the processing lifecycle.

2.4.4 Database Schema

The structure of our database was designed in such a way to allow both for efficient querying without the need for unnecessary joining of tables, and for a clean interface to interacting with the database, minimizing the number of columns on each database table, and avoiding repeated storage of the same data. With these constraints in mind, we utilized database inheritance - a feature which is supported in postgresql. We have omitted some of the unused parent tables from our schema diagram in the figure above, displaying only the critical tables, in order to facilitate a clearer explanation of our system. The condensed database schema is shown in the V1.4 Database Schema

In general, our database tables can be categorized into 5 groups. The order of these groups listed below give a good idea of how data flows across our system, from when a user requests an order to when that order is scheduled. The groups, as well as detailed description of the tables belonging to each group, and their function, is listed below:

1. **Asset tables**: This group simply consists of the 'satellite' and 'groundstation' database tables - the assets in our system to which events are scheduled.
2. **System order tables**: This group consists of the tables 'maintenance_order', 'imaging_order', and 'outage_order'. These tables are used to store orders that users make to the system. A user could create a system order for a repeated event. For example, a user could put in an order for an image to be taken of a particular latitude/longitude within a particular time window, repeating every day; this would be considered one single system order.
3. **Schedule request table**: This group consists of only one table - the 'schedule_request' table. When orders have been created, the system creates schedule requests for each individual item. For example, if there is an image order, to take an image of a particular latitude/longitude around 10pm-12pm everyday, for the next five days, a schedule request is created for each of the five distinct repeated instances

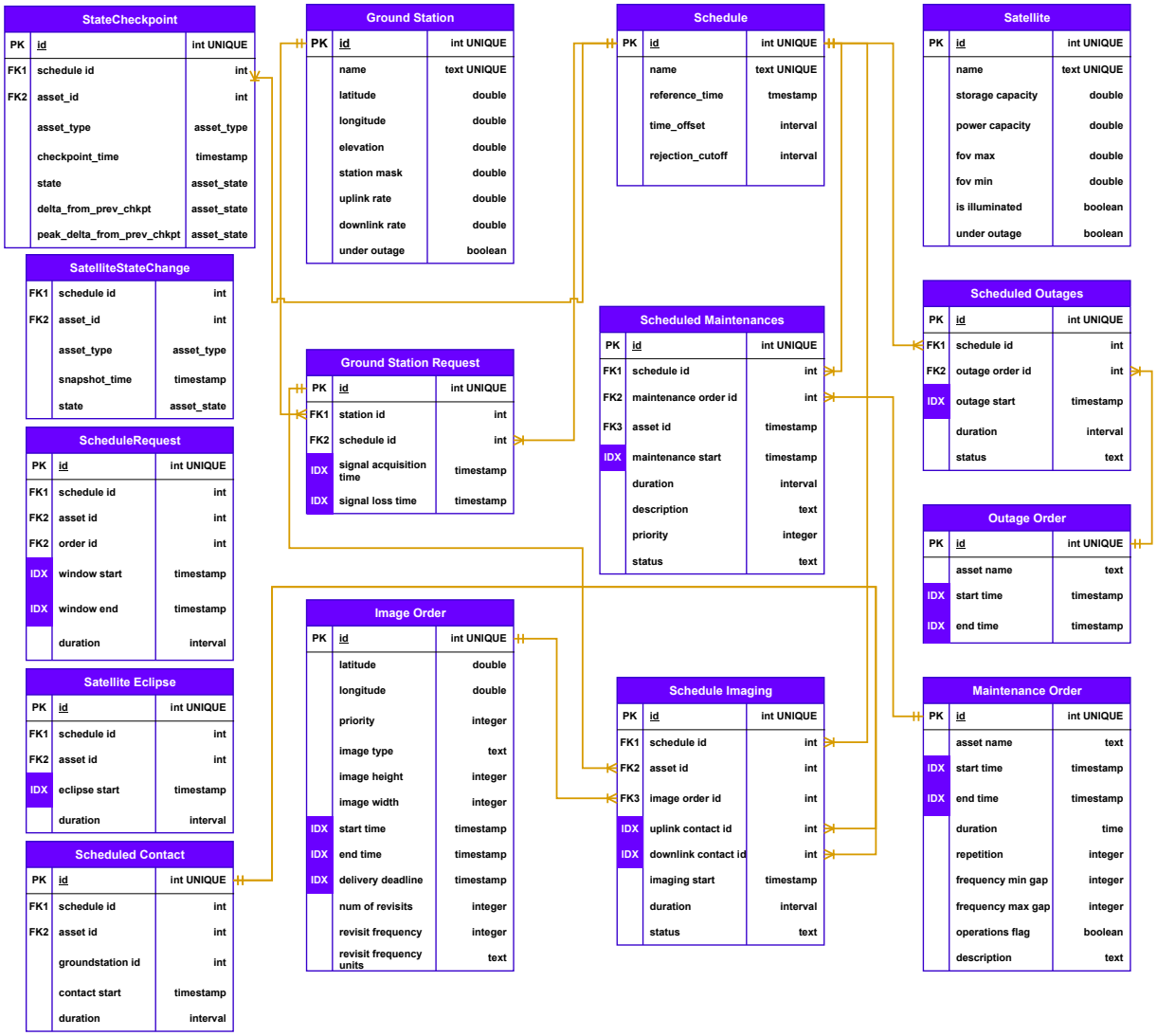


Figure 2.14: SatOpsMQ V1.4 - Database Schema

of the desired event, using the 'schedule_request' table. Each of these five schedule_requests that were created is passed on to the scheduler, and for each of these five schedule requests, the scheduler finds the ideal time between 10pm to 12pm to schedule the event, as well as the ideal satellite to perform this event on.

4. **Scheduled event tables**: There is a corresponding 'scheduled_event' database table for each of the order types, 'imaging', 'maintenance', and 'outage', as well as

for other static events such as eclipse events and contact events (i.e. where a satellite is in contact with a ground station, creating an opportunity for transmission of data). These concrete event tables are the end product of our scheduler system - they contain the information about what exact event should be performed, on which asset, and exactly when.

5. **State tracking tables**: In order for the scheduler to know when to schedule events, and to what satellite/ground-station to schedule the events, it needs to keep track of the satellite's state, in order to avoid creating invalid states: for example, the scheduler should know to avoid scheduling an image to be taken at a time where the satellite's storage is almost used up, and won't have enough capacity to hold the image once captured. The table 'state_checkpoint' are used to keep track of the satellites' state at certain checkpoints, while the 'satellite_state_change' view computes the change in the satellites' state at each point where it's state changes.

2.4.5 Backend Microservices

2.4.5.1 Microservices Structure

In this section, the details of the individual microservices will be described. There are five services in the architecture and they are all linked via the RabbitMQ event messaging framework. The services are loosely coupled and highly cohesive in this format. This ensures the services are scalable and relatively easy to develop in most teams that wish to fork this project. The general file structure for all the microservices is outlined in the microservice folder tree layout below.

Each service has four subdivisions referred to as packages: config, helpers, models, and router. The config package serves to create configurations for connection to external components, such as the database management system RabbitMQ with the microservice. Each microservice needs to establish this integration with these two external systems to realize the design. Moreover, the config package contains a logging system to generate detailed events and error logs to ensure a robust system. The config package is uniform across all services.

The helper package serves as a repository of reusable functions that are needed by the rest of the service. This ensures that the files and functions in each service stay lightweight and simple. Anytime complexity increases in a given function or program file, then the complex operation(s) can be broken up into smaller operations and placed in an appropriate file in the helper package. In the file structure shown below the helpers package

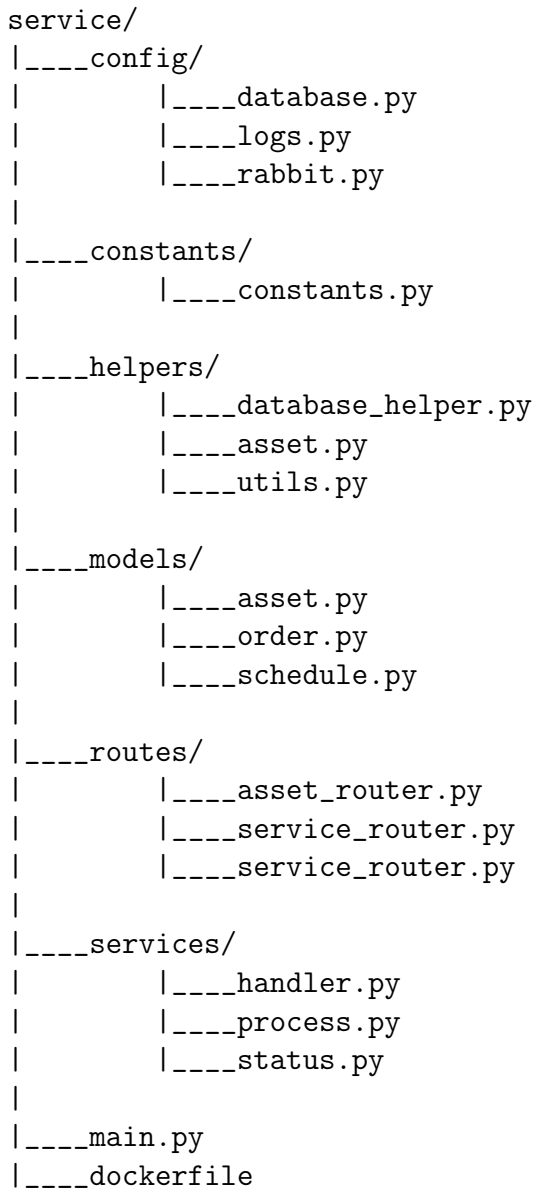


Figure 2.15: General Structure of a SatOpsMQ Microservice

shows samples of files present in this folder. This package is not uniform across the service and may have different modules present depending on the service's nature/needs.

The models package contains schema classes that serve as templates to validate inputs from operators and generate response objects. This package will also have fluidity like the helper package and the exact nature of the Python classes here will vary based on the needs of the microservice.

The routes package is only in the Event Relay API (which will be described in detail in the next section). This package contains API endpoints, which are the URLs used to send and receive information between the frontend and the backend services using the HTTP communication standard. This package divides its modules based on the purpose of the requests. Each file serves a family of requests grouped by this purpose. Moreover, this allows for rigorous URL names that logically relate to one another based on their concern/purpose. The routes package creates a RESTful framework in the event relay API, which is the architectural style built on top of the http protocol. Lastly, this package serves as the outbound routes to other services by pushing events to the RabbitMQ event queue.

The services package is found in the four remaining services as these do not communicate with the frontend directly, but rather are alerted by the RabbitMQ event messaging broker when system updates occur. This package is uniform across the four services and has a handler module to interact with event queues. It also contains a process module to transform requests/orders into scheduled activities.

2.4.5.2 Event-Relay-API

The Event Relay API serves as the primary interface for direct interaction with the frontend This service, handling the highest overhead and traffic, manages numerous requests and is designed to be scalable for hundreds of requests per minute. The major interaction points for this service are form input from the front end and subsequent relay to the other services as well as the database. Additionally, the service facilitates the retrieval of data, sending it back to the frontend in response to user queries. The Event Relay API exposes several important API endpoints that facilitate direct interactions with the backend. We used the FastAPI framework for building our APIs. One of the reasons for this choice is that FastAPI allows us to auto generate documentation for our APIs. This is an especially useful feature, especially as our web application scales. The reason for this is that often, documentation does not represent the current state of the system due to the disconnect between the work of building the system, and the work of maintaining the documentation. Using FastAPI removes the problem of having inaccurate, or out-of-date documentation, as these are automatically generated.

FastAPI uses the “Swagger UI" system for visualizing the API endpoints. This documentation can then be seen through a webpage automatically generated by FastAPI. We have printed out the contents of this webpage containing documentation for our current API endpoints, and this printout can be found here. For each endpoint, it contains a detailed breakdown of the api endpoint, sample inputs to the endpoint, and sample outputs from the endpoint. A brief breakdown of our API endpoints is provided in the section below.

Endpoint (/assets)	**Method**	**Description**
/groundstations	GET	Fetch list of ground stations
/groundstations/{id}	GET	Fetch information for a single ground station via its ID (location, uplink, downlink rate etc.)
/groundstations/create	POST	Create new ground station with specified parameters
/satellites	GET	Fetch list of satellites
/satellites/{id}	GET	Fetch information for a single satellite
/satellites/create	POST	Create a new satellite entry in the system via TLE file

Table 2.6: API Endpoints for assets route

Endpoint (/schedules)	**Method**	**Description**
/	GET	Retrieve list of schedules
/requests	GET	Retrieve the list of requests for scheduling
/requests/{request_id}/decline	POST	Override existing schedule by declining it
/{id}/events	GET	Fetch scheduled events through their ID
/name={name}	GET	Fetch scheduled events by their name

Table 2.7: API Endpoints for schedules route

Endpoint (/images)	Method	Description
/orders	GET	Fetch all made image orders
/orders/{id}	GET	Fetch a single image order through its ID
/orders/create	POST	Create an image order with specified parameters

Table 2.8: API Endpoints for images route

Endpoint (/maintenance)	Method	Description
/orders	GET	Fetch all maintenance orders
/orders/{id}	GET	Get a single image order via its ID
/orders/create	POST	Create a maintenance request based on specified parameters

Table 2.9: API Endpoints for maintenance route

2.4.5.3 Satellite Activity Service

2.4.5.4 Scheduler Service

The Scheduler Service is tasked with taking the information from the satellite activity service that is sent via RabbitMQ and running a scheduled job to fetch the image orders from the FTP server. It uses these orders to create a workable maintenance and imaging schedules to be uplinked to satellites for execution and downlink for later usage. The algorithm is detailed step-wise as follows:

1. **Determine Simulation Limits**
   - Identify the earliest image start time and latest delivery time from orders to set the simulation's temporal boundaries.
2. **Simulate Satellite Operations**
   - Commence at the earliest determined time, progressing in 60-second intervals until the simulation period concludes.
   - For each satellite:
     (a) Update parameters like position, latitude, longitude, altitude, and field of

view.
   (b) Calculate the positions of the Earth and Sun.
   (c) Determine illumination status (sunlit or eclipsed).
   (d) In finer steps (10 seconds), enhance the operation simulation to:
      – Check image order fulfillment possibilities.
      – Initialize structures for new observations.
      – Manage observability slots for imaging.
      – Evaluate and adjust ground station communication slots.

3. **Initialize Optimization**
   - Prepare satellites and ground stations by:
     (a) Adding maintenance events and their repetitions.
     (b) Assessing image order area visibility.
     (c) Resetting ground station communication slots.

4. **Execute Optimization**
   - Iterate from start to end time in 60-second steps to schedule:
     (a) Assess each satellite's capability to fulfill orders using a greedy approach that prioritizes orders based on a heuristic such as the proximity of the satellite to the target or the urgency of the order.
     (b) Apply the heuristic algorithm to validate events against maintenance schedules, preferring times where the satellite is least likely to be needed for high-priority image captures.
     (c) Confirm and allocate downlink slots, adjusting for overlaps by using a greedy method that selects the most efficient sequence of downlink opportunities that maximize the ground station usage balance and satellite data offloading.
     (d) Remove processed image events to prevent duplication, ensuring that once an order is scheduled, it's no longer considered by the greedy-heuristic routine to optimize subsequent scheduling decisions.

5. **Compile Satellite Activity Schedule**
   - Consolidate schedules for each satellite, including:
     (a) Image captures with timing and priority.
     (b) Downlink sessions with corresponding ground stations.
     (c) Maintenance tasks, both planned and repetitive.
   - Present a unified view of each satellite's scheduled activities for the entire period.

For a more comprehensive outline of the algorithm, Algorithm blocks 2-45 serve as a

comprehensive abstract representation of the steps in very high level psuedocode. Furthermore, we have made available the full Jupyter Notebook for readers to try out and verify.

Algorithm 2 Initialize Ground Station

```
1: function INITIALIZE(name, latitude, longitude, height, mask_receive, mask_transmit, uplink_rate, downlink_rate)
2:     Name ← name
3:     Latitude ← latitude
4:     Longitude ← longitude
5:     Height ← height
6:     StationMaskRecv ← mask_receive
7:     StationMaskTrans ← mask_transmit
8:     UplinkRate ← uplink_rate
9:     DownlinkRate ← downlink_rate
10:    RecofigTime ← 300 seconds
11:    Initialize topos with latitude and longitude
12:    Initialize availableSlots as an empty list
13:    Initialize allocatedSlots as an empty list
14: end function
```

Algorithm 3 Check Visibility for Receiving

```
1: function ISVISIBLERECV(satellite, time)
2:     Calculate relative position of satellite with respect to ground station at time
3:     if elevation angle > StationMaskRecv then
4:         return True
5:     else
6:         return False
7:     end if
8: end function
```

Algorithm 4 Check Visibility for Transmitting

```
function isVisibleTrans(satellite, time)
    Calculate relative position of satellite with respect to ground station at time
    if elevation angle > StationMaskTrans then
        return True
    else
        return False
    end if
end function
```

Algorithm 5 Reset Slots

```
function Reset(start, end)
    Create a slot with start and end times
    Reset availableSlots to contain just this slot
    Reset allocatedSlots to an empty list
end function
```

Algorithm 6 Convert String to Time

```
function StrToTm(time_scale, string)
    Convert string to datetime object
    return time_scale's UTC time for the datetime
end function
```

Algorithm 7 Determine Uplink Capability

```
function CanUplink(schedule, start, time_scale, visibility)
    Calculate transfer duration based on schedule size and uplink_rate
    Determine delivery start time from schedule's activity window start
    return result of attempting to get a communication slot for uplink
end function
```

Algorithm 8 Determine Downlink Capability

```
function CanDownlink(event, time, time_scale, visibility)
    Set start to time plus event's transfer time
    Calculate transfer duration based on event's storage size and downlink_rate
    Determine delivery time from event's delivery time
    return result of attempting to get a communication slot for downlink
end function
```

Algorithm 9 Get Communication Slot

```
1: function GetSlot(requested_start, requested_end, duration, visibility)
2:     for each slot in availableSlots do
3:         for each window in visibility do
4:             Determine the actual start and end times based on requested times, slot, and visibility window
5:             if a suitable duration is found within the constraints then
6:                 return True, start, slot
7:             end if
8:         end for
9:     end for
10:    return False, indicating no suitable slot was found
11: end function
```

Algorithm 10 Book Communication Slot

```
1: function BookSlot(event, start, slot, time_scale)
2:     Calculate event duration based on event's storage size and downlink rate
3:     Adjust availableSlots and allocatedSlots based on the booking
4:     Update event with slot and station information
5:     return the newly allocated slot
6: end function
```

Algorithm 11 Initialize Satellite

function InitSatellite(tle)
    LoadTLE(tle)
    InitPower()
    $StorageCapacityMax \leftarrow 40 \times 1024 \times 1024 \times 1024$
    $StorageCapacity \leftarrow 40 \times 1024 \times 1024 \times 1024$
    $viewAngle \leftarrow 30$
    $viewRatio \leftarrow \sin(viewAngle \times \frac{\pi}{180})$
    $lat \leftarrow -100000$
    $long \leftarrow -100000$
    $maint \leftarrow []$
    $Events \leftarrow []$
    $Scheduled \leftarrow []$
    $outages \leftarrow []$
    $stationVisibility \leftarrow \{\}$
end function

Algorithm 12 Load Two-Line Element Set

function LoadTLE(tle)
    $ts \leftarrow$ load timescale()
    Open tle file and read data
    $self.name, line1, line2 \leftarrow$ extracted data from file
    $self.satObj \leftarrow$ EarthSatellite($line1, line2, self.name, ts$)
end function

Algorithm 13 Initialize Power Settings

function InitPower
    $P\_sunlit \leftarrow 500$
    $P\_eclipse \leftarrow P\_sunlit \times 0.4$
end function

Algorithm 14 Process Day and Night Transitions

```
1: function ProcessDayNight(tm, is_sunlit)
2:     if is_sunlit then
3:         EndNightSession(tm)                    ▷ Conclude any ongoing night session
4:     else
5:         StartNightSession(tm)                  ▷ Begin a new night session
6:     end if
7: end function
8: function StartNightSession(tm)
9:     if self.nightSlot["Start"] is None then
10:        self.nightSlot["Start"] ← tm           ▷ Record the start of night
11:    end if
12: end function
13: function EndNightSession(tm)
14:    if self.nightSlot["Start"] is not None then
15:        Create and log the completed night slot from self.nightSlot["Start"] to tm
16:        Reset self.nightSlot["Start"] to None  ▷ Prepare for next night session
17:    end if
18: end function
```

Algorithm 15 Update Satellite State

```
1: function Update(tm, eph, images, ts, GroundStations)
2:     Calculate current position
3:     is_sunlit ← Determine if satellite is sunlit
4:     ProcessDayNight(tm, is_sunlit)             ▷ Handle day/night transition
5:     Update latitude and longitude based on current position
6:     Update field of view based on altitude
7:     for each image in images do
8:         Check and log image capture start/end
9:     end for
10:    Process ground station visibility and slots
11: end function
```

Algorithm 16 Process Ground Stations - High-Level Overview

```
1: function ProcessGroundStations(tm, stations)
2:     for each station in stations do
3:         Ensure station has an entry in stationVisibility
4:         if station is visible for transmission at currentTime then
5:             Record start of visibility period if not already started
6:         else
7:             if visibility period was ongoing then
8:                 Close the visibility period and log it
9:             end if
10:        end if
11:    end for
12:    Update visibility data structure for each station based on the observation
13: end function
```

Algorithm 17 Determine Image Intersection with Satellite FoV

```
1: function Intersect(image, Latitude, Longitude)
2:     Determine the distance between the image location and the current satellite position.
3:     Compare the calculated distances with the satellite's field of view dimensions.
4:     if the image is within the satellite's field of view then
5:         return True                                  ▷ There is an intersection.
6:     else
7:         return False                                 ▷ There is no intersection.
8:     end if
9: end function
```

Algorithm 18 Log Start of Image Capture

```
1: function LogImageStart(im, tm)
2:     Initialize image source entry if not present.
3:     Record the start time of image capture session.
4: end function
```

Algorithm 19 Log End of Image Capture

```
1: function LogImageEnd(im, tm)
2:     Exit if image source entry is missing.
3:     Calculate the duration of the image capture session.
4:     Log the session if its duration exceeds the transfer time.
5:     Reset the session start indicator.
6: end function
```

Algorithm 20 Process Captured Images

```
1: function ProcessImages(images, Latitude, Longitude, tm)
2:     for each image in images do
3:         Check if the image intersects with the satellite's field of view.
4:         if there is an intersection then
5:             Log the start of the image capture.
6:         else
7:             Log the end of the image capture.
8:         end if
9:     end for
10: end function
```

Algorithm 21 Finalize Image Processing

```
1: function ProcessImagesFinal(images, tm)
2:     for each image in images do
3:         Log the end of the capture session for each image.
4:     end for
5: end function
```

Algorithm 22 Compare Time Strings

```
function CompTimeStr(t1_str, t2_str)
    Convert t1_str and t2_str to datetime objects t1 and t2.
    if t1 > t2 then
        return 1
    else if t1 = t2 then
        return 0
    else
        return -1
    end if
end function
```

Algorithm 23 Add Child Events

```
function AddChildren(ev, tm)
    Add child events based on the type of the event ev.
    if ev type is not recognized then
        Print an error message indicating an unknown event type.
    end if
end function
```

Algorithm 24 Register an Event

```
function RegisterEvent(tm, ev, ts)
    Record the event ev as happening at time tm.
    Update any relevant data structures or counters.
end function
```

Algorithm 25 Convert String to Time Object

```
function StrToTm(ts, Str)
    Convert the string Str to a datetime object dt.
    return The UTC time corresponding to dt in the timescale ts.
end function
```

Algorithm 26 Schedule Maintenance Events

```
function AddMaintEvent(ts, ev0)
    if repetition is not required for ev0 then
        Exit the function                                  ▷ No further action needed.
    end if
    Initialize parent event as ev0.
    Schedule new maintenance events based on the repetition details in ev0.
    for each repetition level beyond the current one do
        Create and link new maintenance event with appropriate timing.
        Update events list and parent-child relationships.
    end for
end function
```

Algorithm 27 Bifurcate Maintenance Event

```
function BifurcateMaint(ev, ev_begin, ev_end, ts)
    Assess if the maintenance event ev overlaps with other scheduled activities.
    if event ev can be split before ev_begin then
        Create a new event evA before ev_begin.
        Schedule evA and link it to the parent event if applicable.
    end if
    if event ev can be split after ev_end then
        Create a new event evB after ev_end.
        Schedule evB and link it to the parent event if applicable.
    end if
    Remove the original event ev if it has been successfully bifurcated.
end function
```

Algorithm 28 Remove Maintenance Event

```
function RemoveMaint(ev)
    Recursively remove all child events linked to ev.
    if ev has a parent then
        Remove ev from its parent's child list.
    end if
    Remove ev from the global list of events.
end function
```

Algorithm 29 Check if Satellite is Busy

```
1: function IsBusy(ev1)
2:     Check if the satellite has enough storage for ev1.
3:     if not enough storage then
4:         return True
5:     end if
6:     Check if ev1 overlaps with any scheduled events or outages.
7:     if any overlap then
8:         return True                                  ▷ Satellite is busy
9:     else
10:        return False                                 ▷ Satellite is not busy
11:    end if
12: end function
```

Algorithm 30 Load Ground Stations

```
1: function LoadGS
2:     Initialize an empty list for ground stations.
3:     Add predefined ground stations with their specific parameters.
4: end function
```

Algorithm 31 Check for Downlink Capability

```
1: function CanDownlink(ev, tm, sat)
2:     Iterate over ground stations to check downlink capability for event ev at time tm.
3:     if downlink is possible with any station then
4:         return details including the station.
5:     else
6:         Log an error message indicating inability to uplink at tm.
7:         return False with default parameters.
8:     end if
9: end function
```

Algorithm 32 Load Outage Requests

```
1: function LoadOutages
2:     Retrieve outage request files from a specified directory.
3:     for each file in the list do
4:         Load and parse outage data.
5:         Convert start and end times to the appropriate format.
6:         Assign outages to corresponding satellites.
7:     end for
8: end function
```

Algorithm 33 Load Satellites

```
1: function LoadSat
2:     Initialize an empty list for satellites.
3:     Retrieve satellite TLE files from a specified directory.
4:     for each TLE file do
5:         Create a satellite instance and add it to the satellites list.
6:     end for
7: end function
```

Algorithm 34 Load Ephemeris Data

```
1: function LoadEPH
2:     Load the JPL ephemeris file to determine satellite illumination status.
3: end function
```

Algorithm 35 Load Image Order Requests

```
1: function LoadOrders
2:     Initialize an empty list for image orders.
3:     Retrieve image order request files.
4:     for each order file do
5:         Parse the order file to extract image request details.
6:         Mark the order as not completed and initialize necessary attributes.
7:         Adjust image specifications based on the image type.
8:         Add the order to the list with its index.
9:     end for
10:    Sort the orders list by priority.
11: end function
```

Algorithm 36 Simulate Satellite Operations

```
function SIMULATE(start, end)
    Convert start and end into skyfield Time objects.
    Determine the minimum and maximum times across all orders for simulation bounds.
    Adjust the simulation start and end times based on image orders.
    Initialize the simulation time tm to the start time.
    while tm is less than the end time do
        for each satellite in Satelites do
            Update satellite status and process images for the current time tm.
        end for
        Increment tm by 60 seconds for the next simulation step.
    end while
    for each satellite in Satelites do
        Finalize image processing after the simulation period.
    end for
end function
```

Algorithm 37 Check Event Overlap

```
function EVOVERLAP(ev1, ev_begin, ev_end)
    Check if the event ev1 overlaps with the time range between ev_begin and ev_end.
    if any part of ev1 falls within the range then
        return True                                  ▷ Indicates an overlap.
    else
        return False                                 ▷ No overlap.
    end if
end function
```

Algorithm 38 Validate Maintenance Bifurcation

```
function BifurcateMaintValid(ev, ev_begin, ev_end)
    Determine if the event ev can be split into parts before ev_begin and after ev_end.
    if either part has a duration exceeding the threshold then
        return True                                          ▷ Splitting is valid.
    else
        Log the time comparison details for debugging.
        return False                                         ▷ Splitting is not valid.
    end if
end function
```

Algorithm 39 Process Image Event

```
function ProcImageEvent(ev, tm, sat)
    Initialize the event time frame based on ev's transfer time.
    Assess if the image event is valid considering maintenance overlaps.
    for each maintenance event maint in satellite do
        Check for overlap with the image event.
        if overlap exists then
            Determine if the event can still proceed based on maintenance bifurcation rules.
            if cannot proceed then
                Mark the image event as invalid.
            end if
        end if
    end for
    return the validity and overlap status of the image event.
end function
```

Algorithm 40 Greedy Slot Selection

```
1: function SELECT(slots)
2:     Initialize selection with the first slot as the best
3:     Determine initial best congestion levels for stations and satellites
4:     Compute average congestion for stations and satellites
5:     for each slot in slots do
6:         Compute congestion score for the station and satellite in the slot
7:         if both averages are non-zero then
8:             Calculate relative congestion scores
9:             Combine scores into a total score for the slot
10:            if this score is better than the current best then
11:                Update best slot and best score
12:            end if
13:        else
14:            Update best slot based on simple congestion comparisons
15:        end if
16:    end for
17:    return bestSlot
18: end function
```

Algorithm 41 Get Possible Order Slots for Image Events

```
1: function GetPossibleOrderSlots(order)
2:     Initialize an empty list for slots and a flag for consumed slots
3:     Set the time margin before and after the order imaging window
4:     while current time within the extended order window do
5:         for each satellite in the system do
6:             if satellite has the order source then
7:                 for each event related to the order in the satellite's queue do
8:                     if event is valid and within the time window then
9:                         if satellite is not busy then
10:                            Check for downlink possibility and overlaps
11:                            if downlink is possible then
12:                                for each potential downlink slot do
13:                                    Create a slot order entry
14:                                    Add to the list of potential slots
15:                                    Set the consumed flag to True
16:                                end for
17:                            end if
18:                        end if
19:                    end if
20:                end for
21:            end if
22:        end for
23:        Advance the current time by a step increment
24:    end while
25:    return the consumed flag and the list of slots
26: end function
```

Algorithm 42 Initialization and Optimization of Satellite Orders

```
1: function InitOptimizationOrder(order)
2:     Convert image start and end times from strings to time objects
3:     for each sat in self.Satellites do
4:         if sat.name in order['Sources'] then
5:             for each session in order['Sources'][sat.name]['sessions'] do
6:                 if session[0].tt > start.tt and session[1].tt < end.tt then
7:                     Create and append an event to sat.Events
8:                 end if
9:             end for
10:        end if
11:    end for
12: end function
13: function OptimizeOrder(order)
14:    if order['Completed'] or order['Processed'] then
15:        return
16:    end if
17:    Simulate order conditions
18:    InitOptimizationOrder(order)
19:    consumed, slots ← GetPossibleOrderSlots(order)
20:    if consumed then
21:        slotOrder ← select(slots)          ▷ slot selection using the greedy algorithm
22:        Book the selected slot for the satellite and ground station
23:        Register the event with the satellite
24:        if there is an overlap with other events then
25:            Manage event overlaps, including bifurcation of maintenance if necessary
26:        end if
27:        Remove the completed order events from sat.Events
28:    end if
29:    order['Processed'] ← True
30: end function
```

Algorithm 43 Optimize Satellite Image Capture and Downlink Scheduling

```
function OPTIMIZE
    Start timing the optimization process.
    Set the current time tm to the simulation's start time.
    while tm is before the simulation's end time do
        for each order in orders do
            Initialize consumed as False.
            for each sat in Satellites do
                if sat does not have order in its source list then
                    Continue to the next satellite.
                end if
                for each event in sat's events do
                    if event does not match the current time and order criteria then
                        Continue to the next event.
                    end if
                    Check if event is valid and does not overlap with maintenance.
                    if not valid or overlaps then
                        Continue to the next event.
                    end if
                    Attempt to schedule a downlink for event.
                    if downlink scheduling is successful then
                        Book the slot, register the event, and mark consumed as True.
                        Break the loop to process the next order.
                    end if
                end for
                if consumed is True then
                    Break the loop to skip to the next tm increment.
                end if
            end for
            Remove processed events from sat's event list.
        end for
        Increment tm by 60 seconds.
    end while
    Calculate the duration of the optimization process and print it.
end function
```

Algorithm 44 Run Satellite Operations Simulation and Optimization

```
1: function RUN(start, end)
2:     Record the start time of the entire process.
3:     Invoke the simulation process over the specified start and end times.
4:     Record the time after the simulation completes.
5:     Print the duration of the simulation process.
6:     Initialize the optimization process to prepare for scheduling.
7:     Execute the optimization routine to schedule satellite activities efficiently.
8:     Generate and finalize the satellite activity schedule post-optimization.
9: end function
```

Algorithm 45 Generate Satellite Activity Schedule

```
1: function SATELLITEACTIVITYSCHEDULE
2:     Initialize an empty list for activity schedules.
3:     for each sat in Satellites do
4:         Create a new schedule sched with basic satellite information and activity windows.
5:         for each image in sat's scheduled images do
6:             Adjust the activity window based on image's timing.
7:             Add image capture and downlink activities to sched.
8:         end for
9:         for each maintItem in sat's maintenance list do
10:            Add maintenance activities to sched, including child activities if any.
11:        end for
12:        Finalize the activity window timings in sched.
13:        Add sched to the list of activity schedules.
14:        Increment schedule and maintenance IDs for tracking.
15:    end for
16:    Print or log the generated schedules for review.
17: end function
```

2.4.5.5 FTP Service

2.4.6 System Functionality & Design

With the high-level details and design decisions justified and described, it is now important to move deeper into the system and dissect it across various levels of abstraction to understand the design in detail. The general order of understanding that we outline here is use cases, then interfaces/components, followed by functional/logical flow of operations, and concluding with the concrete data model being used to accomplish all of this.

2.4.6.1 Use Cases

We'll be tackling the various use cases of the SatOpsMQ program. These use cases correspond to common scenarios the operator may undertake and their connections with various components of the system.

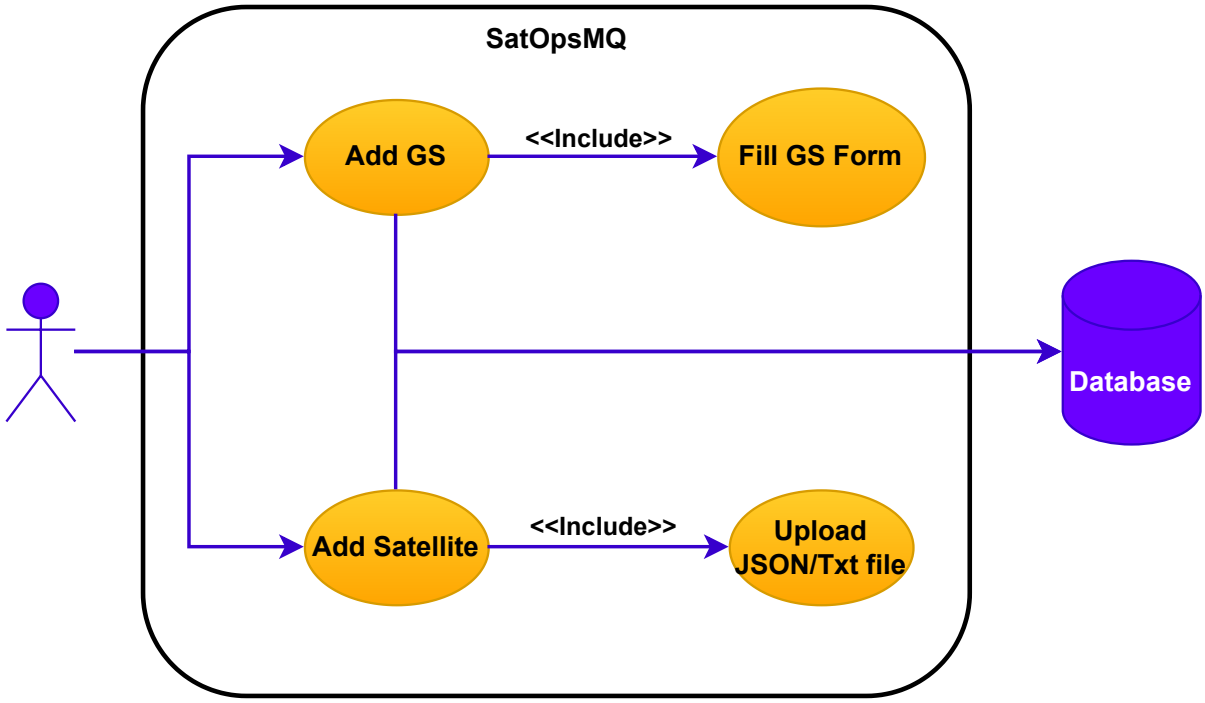


Figure 2.16: Use case diagram for adding a new satellite or ground station asset.

Adding Ground Station or Satellite Starting with the first use case which is to add a new satellite or ground station asset. The use case is visualized in add satellite or ground station diagram, and it covers the various actors involved when the operator would want to add either a new satellite or a ground station asset.

This use case exists in order to allow the program to be more dynamic, and to fulfill the needs of the operator who may need to add more ground stations or satellites.

Adding a ground station involves the operator navigating to the Ground Station form on the application and filling in the form's parameters that correspond to the ground station parameters and clicking submit. This process requires the involvement of the database in order to store the form details and have the system translate the input data into a functioning ground station entity.

If the operator wanted to add a satellite, they would need to navigate to the Add Satellite page of the application and upload either a JSON or a text file containing satellite data. The system in turn would store the input data into the database and translate it into a satellite entity accordingly.

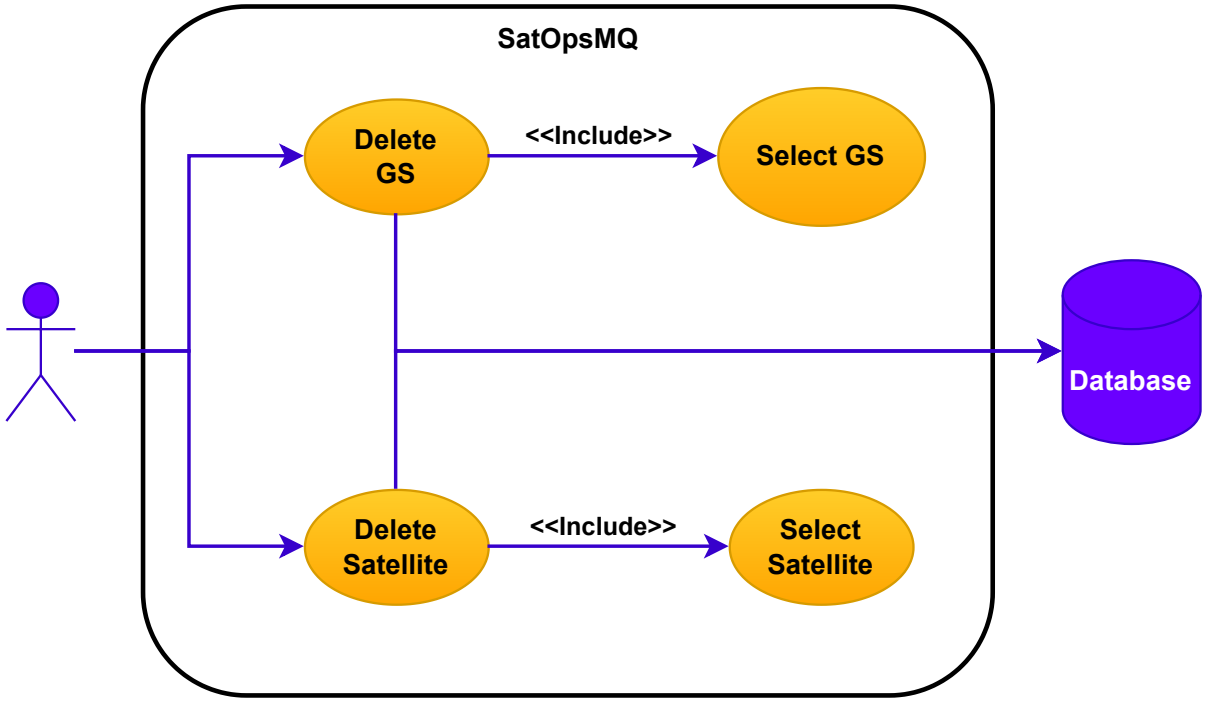


Figure 2.17: Use case diagram for deleting a saved satellite or ground station asset.

Ground Station or Satellite Deletion Moving onto the second use case is the deletion of the satellite or the ground station. This use case is visualized in Deleting a saved satellite or ground station asset diagram, and covers the various actors involved in this use case being the same actors from the first use case.

This use case exists to complement the previous use case. In case the operator decides to backtrack on their decision to add a ground station or satellite. They have the option to remove it.

The process to delete a ground station involves the operator navigating to the Ground Station page listing all the functional ground stations, selecting the appropriate one, and clicking delete. This process requires the involvement of the Database actor that stores the ground station parameters and thus can delete them.

The process to delete a satellite matches the process mentioned before, which involves navigating to the Satellite page listing all the satellites, selecting a specific satellite, and clicking delete. This process involves the database actor as it stores the satellite information and thus can delete it.

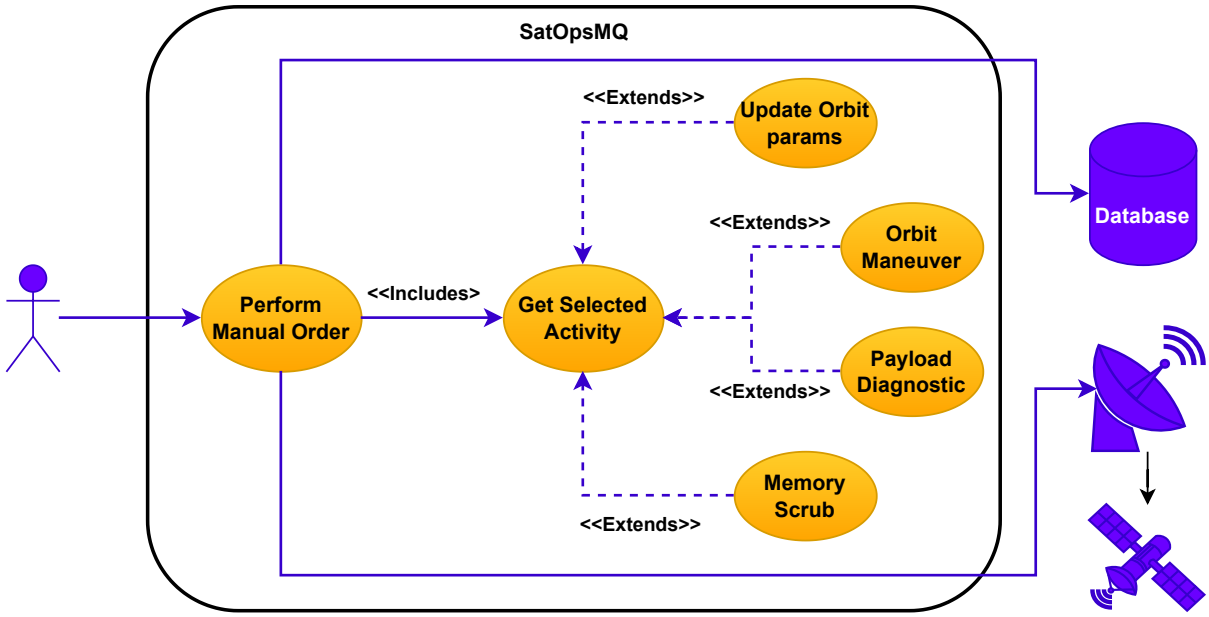


Figure 2.18: Use case diagram for manually entering a maintenance order.

Maintenance Order Submission The third use case tackles the action of submitting a maintenance order. This use case is visualized in manually entering a maintenance order

diagram, and covers the various actors involved being the database, the ground station, and a specific satellite.

This use case is necessary for operators who may need to manually alter a satellite's behaviour through a maintenance order. This allows them to either update orbital parameters for special cases, modify the orbit maneuver, perform a payload diagnostic check, or perform a memory scrub.

If the operator would like to submit a maintenance order, they would need to determine which category of maintenance they would need to perform. There are various categories of maintenance to select from: Updating orbital parameters of a satellite, modifying the orbit maneuver, performing a payload diagnostic, or performing a memory scrub.

This process requires communication with 3 actors: the database, the ground station, and the satellite. The database handles the storage of maintenance orders. The ground station receives commands from the system in regards to which manual orders are to be sent, and their respective satellites. Therefore the ground station is connected to the system, and the satellite is connected only to the ground station.

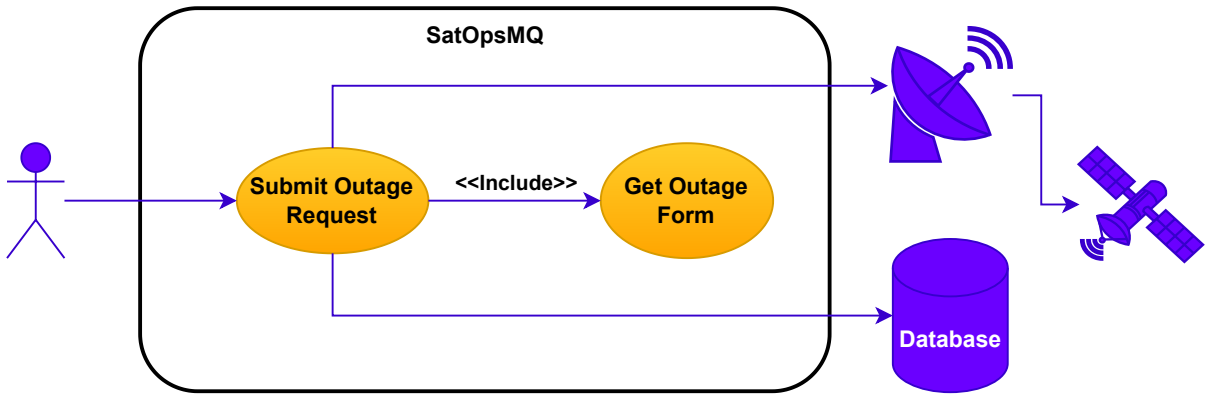


Figure 2.19: Use case diagram for an outage request

Outage Request Submission The fourth use case would be the submission of outage requests for the satellites. This use case is visualized in outage request diagram, and covers the various actors involved being the database, the ground station, and a specific satellite.

This use case is essential for cases when the operator's attempts at submitting maintenance requests aren't enough to properly lessen the load of a satellite. In that circumstance, the operator would have the option to submit an outage request to temporarily suspend a satellite thus relieving it of heavy loads if unbalanced.

If the operator would like to momentarily shut down a satellite for various reasons, they would submit an outage request. This process involves navigating to the Outage form page on the application, filling out the required parameters, and submitting.

The process involves the following actors: database, ground station, and satellite. The database stores the outage requests. The ground station receives the outage requests from the system, and in turn, sends them to the corresponding satellite.

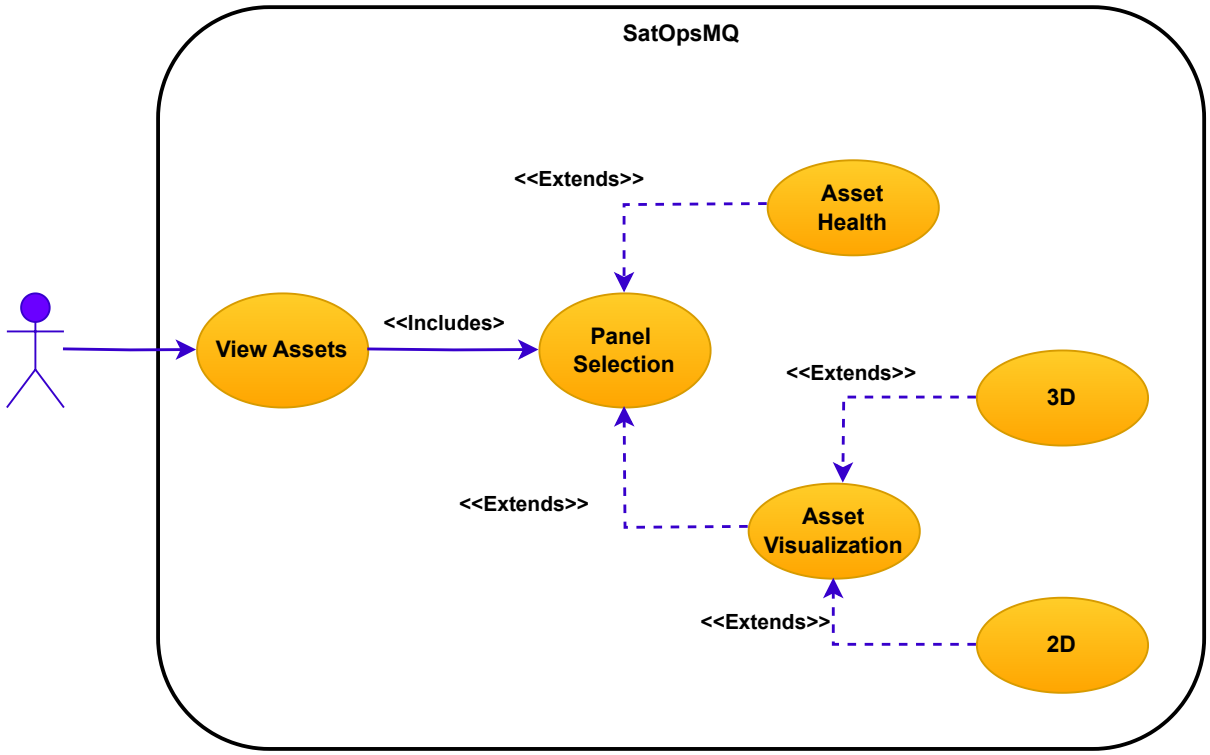


Figure 2.20: Use case diagram for checking the status of a satellite or ground station asset.

Ground Station or Satellite Status Check The fifth use case at hand is checking the status of a satellite or a ground station. This use case is visualized in checking the status of a satellite or ground station asset diagram, and covers the various actors involved being the operator and the ground station. This use case is vital as it is one of the most common checks an operator would be doing during their workflow.

If an operator would like to check the status of the ground station or the satellite, they can navigate to the health dashboard that can outline the various ground station states or

satellite states. Alternatively, there is a panel dedicated to the visualization of the current positions of the satellite that is available for 2D and 3D view.

The process above utilizes the ground station to check its status and the satellite status if required.

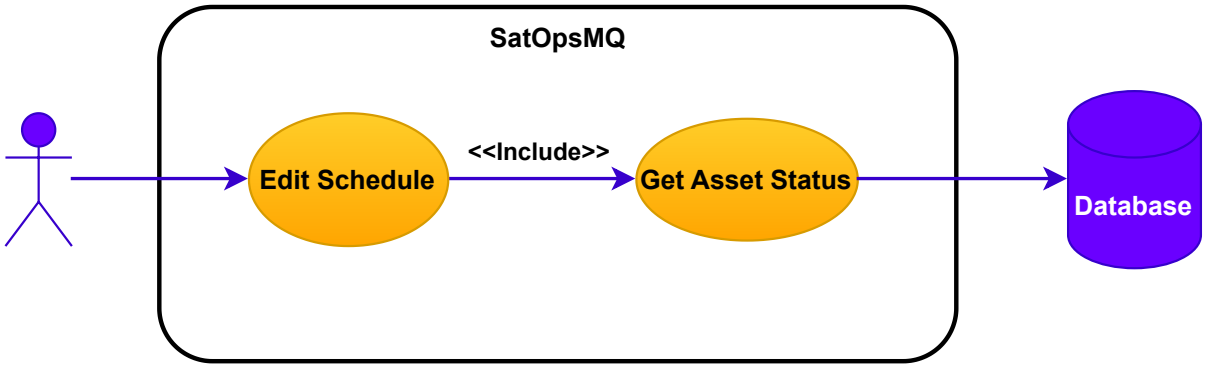


Figure 2.21: Use case diagram for viewing and editing a schedule.

Viewing/Editing Satellite Schedule The sixth use case at hand is being able to view or edit a satellite schedule. This use case is visualized in viewing and editing a schedule diagram, and covers the various actors involved being only the database.

This use case comes in handy when there exists a miscellaneous reason as to why a schedule item needs to be altered, therefore this allows the operator to be able to view and edit the satellite schedule.

This process involves navigating to the schedule dashboard to view the various satellites and their schedules and be able to modify them.

This process heavily depends on the database's stored records of satellite schedules to populate the schedule view for the operator.

Viewing Made Orders Users can view all existing orders sent to the system. This includes image, maintenance, and outage orders; all of which have a dedicated page. This usecase is visualized in viewing made schedules diagram.

Viewing Health Dashboards Users can navigate to the health dashboard to observe the system's performance. They will be able to see the percentage of total orders scheduled,

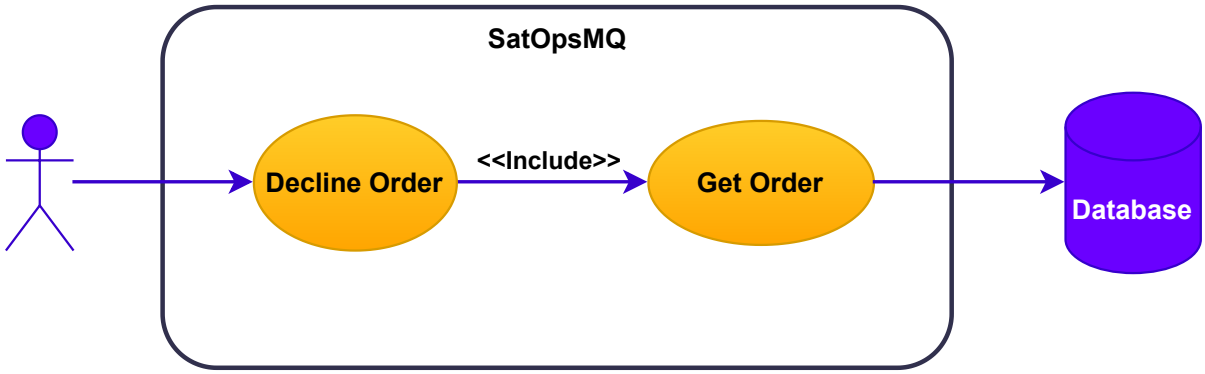


Figure 2.22: Use case diagram for declining an image order.

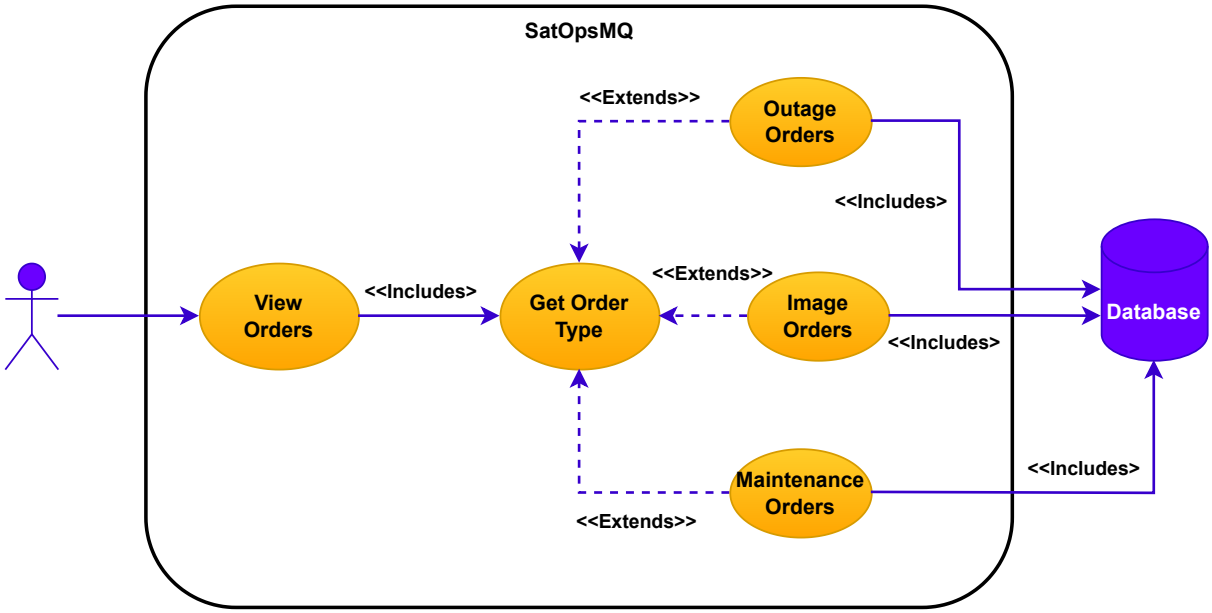


Figure 2.23: Use case diagram for viewing made orders.

the amount of orders each asset is accommodating for, and any system alerts. This usecase is visualized in viewing health dashboard diagram.

Declining Image Orders The final use case is being able to decline an image order. This use case is visualized in declining an image order diagram, and covers the various actors involved being only the database.

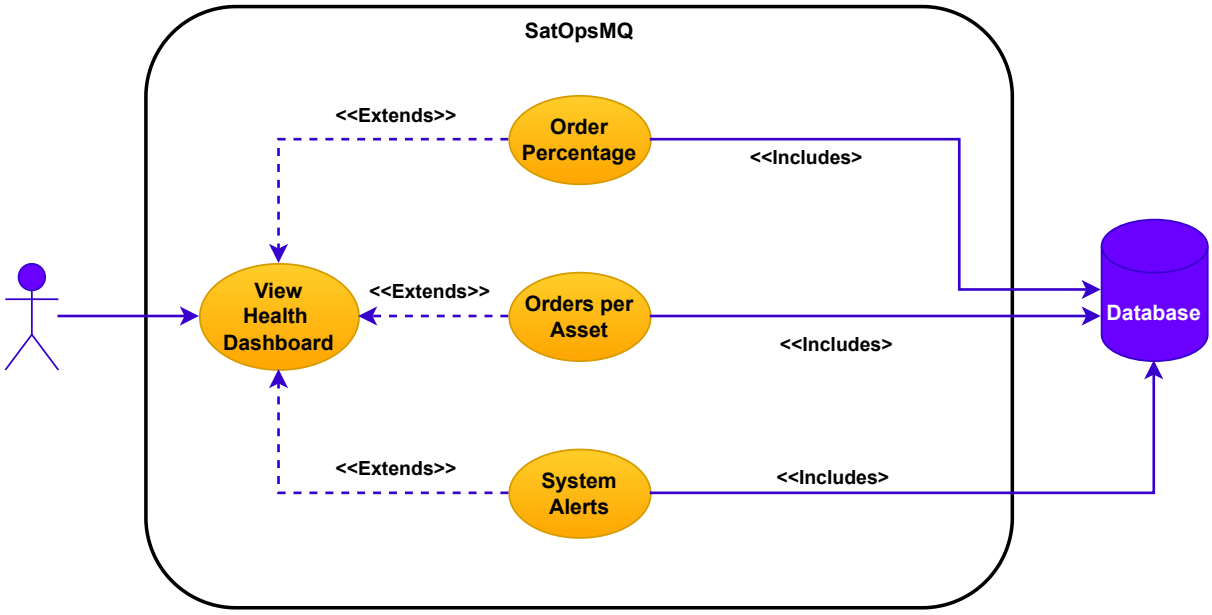


Figure 2.24: Use case diagram for viewing the health dashboard

Another vital use case that is essential and common in an operator's workflow, is being able to decline an image order for whatever reason.

This process involves the operator selecting a specific order to cancel. In order to populate the view for the operator to see the list of orders, the database is involved in order to retrieve those orders.

2.4.6.2 Interface & Component Layout

The system's design is laid out as described in the system component diagram. As shown in the diagram, the EventRelay API's responsibilities can be categorized into two functionalities - enabling user interaction with the system and alerting the system to user interaction through the emission of events. Looking first at how it enables user interaction, we can see from the diagram that the event relay API is responsible for the creation/deletion of the satellite and ground station assets, as well as the database CRUD (create, read, update, and delete) operations associated with creating maintenance and outage orders. It exposes HTTP endpoints with which the client can interact with each of these functionalities. When any client interaction happens, the EventRelay API emits an event, using the RabbitMQ messaging system, to alert the rest of the system of the interaction. Other

services who are interested in an event can subscribe to that event using the RabbitMQ system and are notified whenever the event takes place, after which they can perform the necessary computation required.

The Scheduler service is where activities are scheduled/rescheduled. Using the RabbitMQ system, it listens to the events emitted by the EventRelay API when orders are created or canceled and updates the ongoing schedule accordingly. The scheduler schedules a stream of activities to take place. It is not responsible for aggregating or dispatching these activities but is only responsible for assigning activities to take place at a set time, considering all the factors that we are optimizing for. It then emits an event whenever an activity is scheduled, rescheduled, or canceled.

The FTP Server is responsible for storing the image orders that were sent by our third-party users. This server stores the orders in the form of a JSON file that resembles the structure of the image order payload discussed in the inputs section. The scheduler service is tasked with pulling these JSON files periodically every one hour from this server to account them as a part of the algorithm's execution.

2.4.6.3 Activity Flow

We have outlined the satellite visualizer workflow, as well as the scheduling workflow, through two corresponding activity diagrams. We will first go over the activity flow of the satellite visualizer.

The main requirement here is to have a way of streaming the satellite's state live to the front end, at the current time. Its state includes information such as its current latitude, longitude, and altitude. This information can then be used to render the satellite, in its current location and with its current state, on the user interface. To achieve this, we use websocket connections so that we can keep a live connection to the client, and stream the satellite's current state.

When a websocket connection is created between the client and the server, expressing interest in listening to a satellite's state, we create a listener for that satellite. The system then starts a loop of getting the satellite's state at the current time and sending that to the websocket client. When the system is interrupted by the websocket client disconnecting, however, we destroy the listener. If there are no more listeners for the specified satellite, then we also stop the system that streams the satellite's state to prevent resource leaks.

The scheduling workflow architecture design is described in the corresponding activity diagram. When we receive an order request, we persist it in the database, pre-process

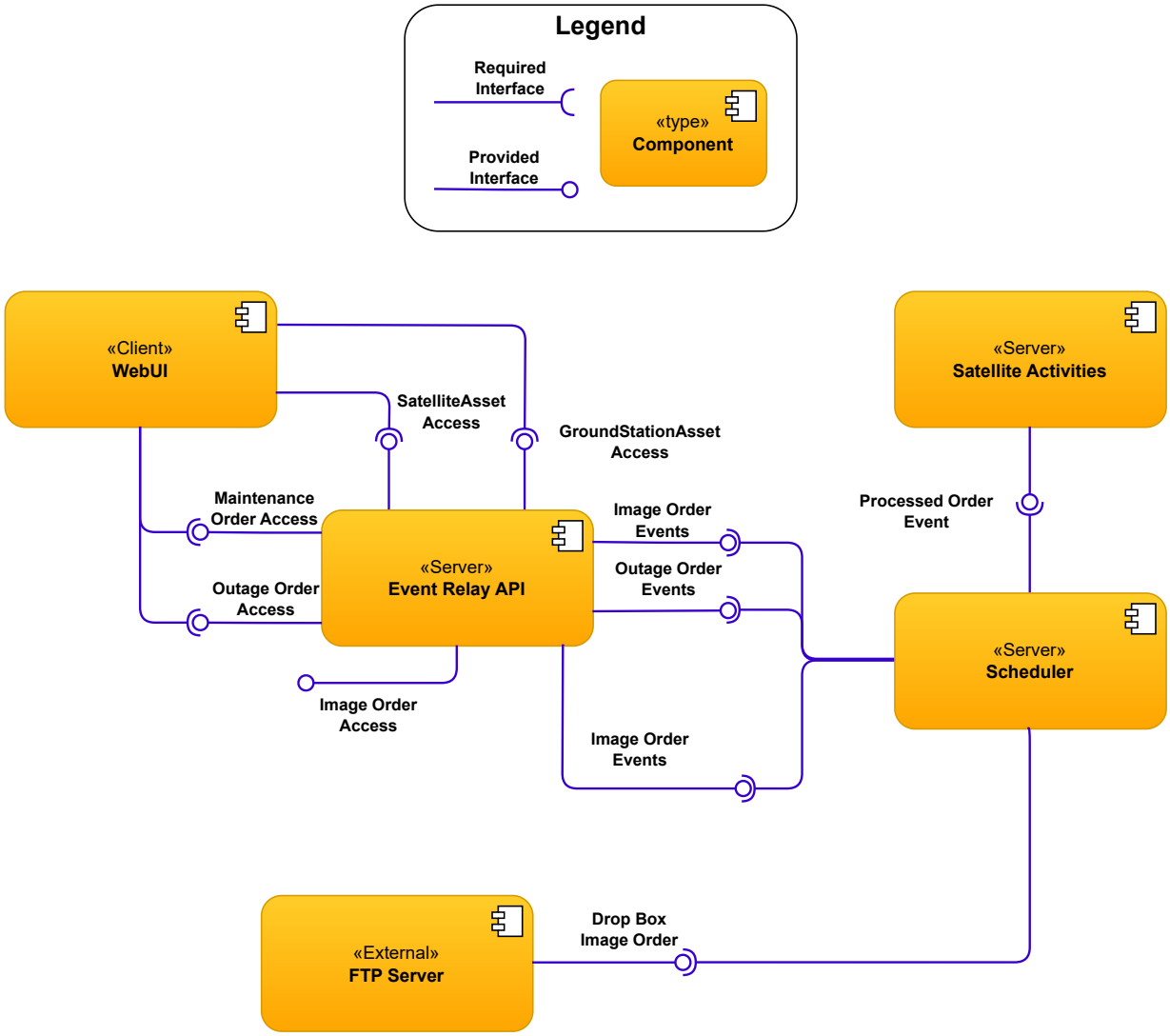


Figure 2.25: SatOpsMQ Component Diagram Showing Required & Provided Interface Connections

the order, and then it is sent over to the scheduler system to plan a schedule for the order. Planning a schedule may involve rescheduling already-scheduled activities, as well as canceling planned activities. The rest of the system is notified of these changes. Afterward, the scheduled activities are batched and sent to the mock ground station for execution.

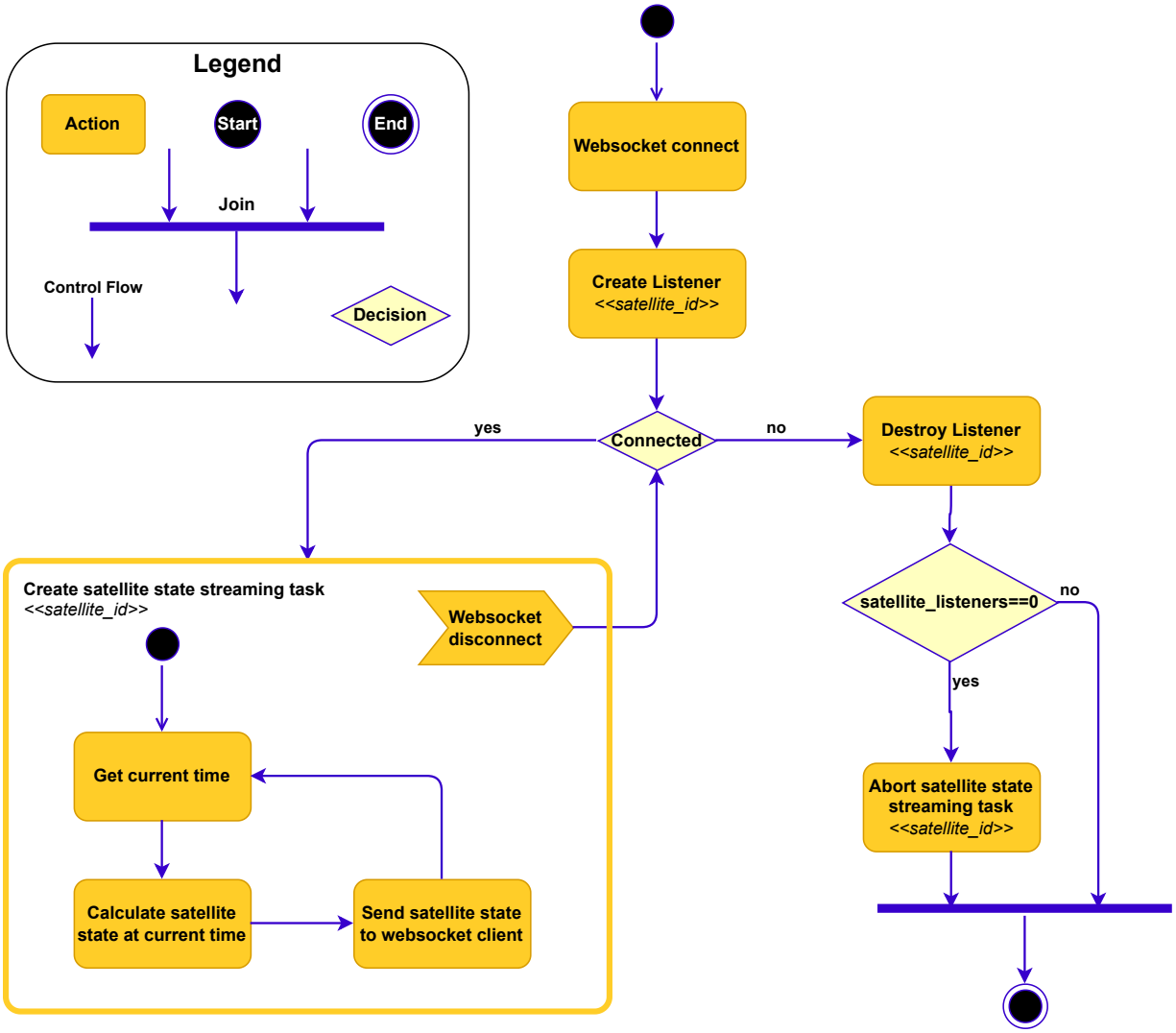


Figure 2.26: SatOpsMQ Satellite Visualizer Activity Diagram

2.4.6.4 Class Layout

Due to the size of the project's code base, only the most significant classes were documented and most of which pertains to the algorithm. The class diagram below outlines the classes that are responsible for calculating the satellite's state. As shown in the class diagram, the satellite state at a given time includes the timestamp, the satellite's latitude, longitude, altitude, field of view, as well as a boolean attribute which is true if the satellite is currently

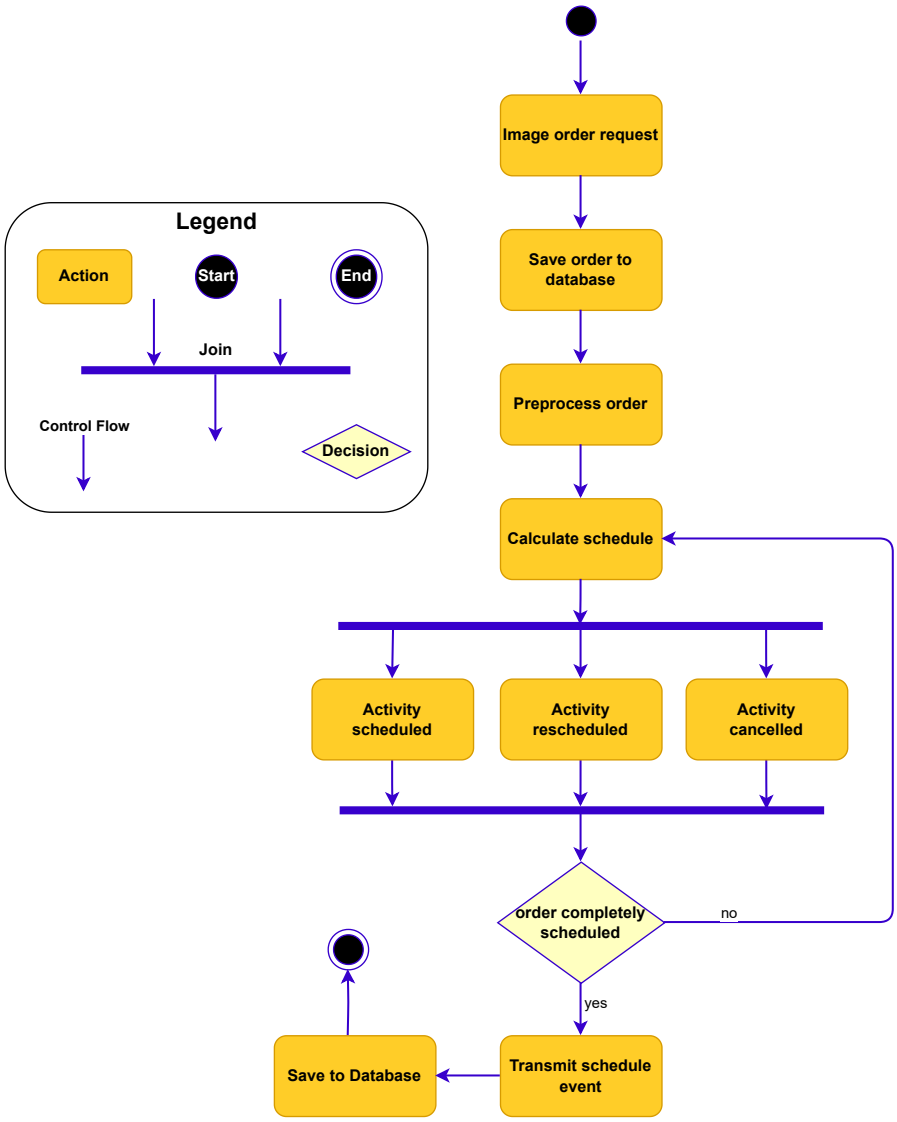


Figure 2.27: SatOpsMQ Scheduling Activity Diagram

exposed to the sun, and false if it is eclipsed.

The SatelliteStateStreamManager is the part of the system that is responsible for managing the listeners and the satellite streaming resources. When no more listeners are listening to the state of the satellite, it aborts the task which streams the state of that satellite in order to save resources, as it is no longer needed.

Finally, we have the SatelliteStateGenerator, which is responsible for calculating the current state of the satellite. The “state_at()", when given a datetime, calculates the satellite's state. This method is used by the “stream()" method to stream the satellite's current state. The "stream()" method is a function that generates the satellite's state at the current time.

We utilize celery tasks to stream the state of the satellite to the listeners. The SatelliteStateGenerator function is used by the celery task to generate the satellite's state at a given time, and then emit the state to all its listeners.

2.4.6.5 Sequence Flow & Layout

The sequence diagrams below outline the same workflows as the activity diagrams do, but makes a clearer distinction on which parts of the system are responsible for each task. First, we will consider the order scheduling sequence of operations.

When an order is requested by the user through the WebUI, the request is sent to the EventRelay API which persists the order and returns the order ID to the user. The EventRelay API then starts the sequence of events which results in the order being scheduled. It notifies the satellite activities service, which pre-processes the order and passes it on to the scheduler service for processing. During scheduling the order, the scheduler service will have to schedule, reschedule, or even cancel different activities. Whenever this happens, it notifies the system of these changes.

The second sequence diagram below outlines the workflow involved in visualizing the satellites. We start with a user opening the webpage to view a satellite. When this page is opened, a websocket connection is made with the EventRelay API, and subsequently, a listener is created for the satellite. Whenever there exists a listener for a satellite, the process for emitting the satellite's state is started and executes in a loop. At every time step, it calculates the state of the satellite and sends it back to the EventRelay API, where it is sent back to the websocket client, the WebUI, where it is used to display the satellite accurately.

At any point in the process, the websocket connection could be disconnected. In such a case, we delete the listener which was created at the start of the websocket connection. If there exists no more listeners for the satellite, then we abort the task which continuously emits the state of the satellite.

Figure 2.28: Scheduler Service Class Diagram for Tasks Package

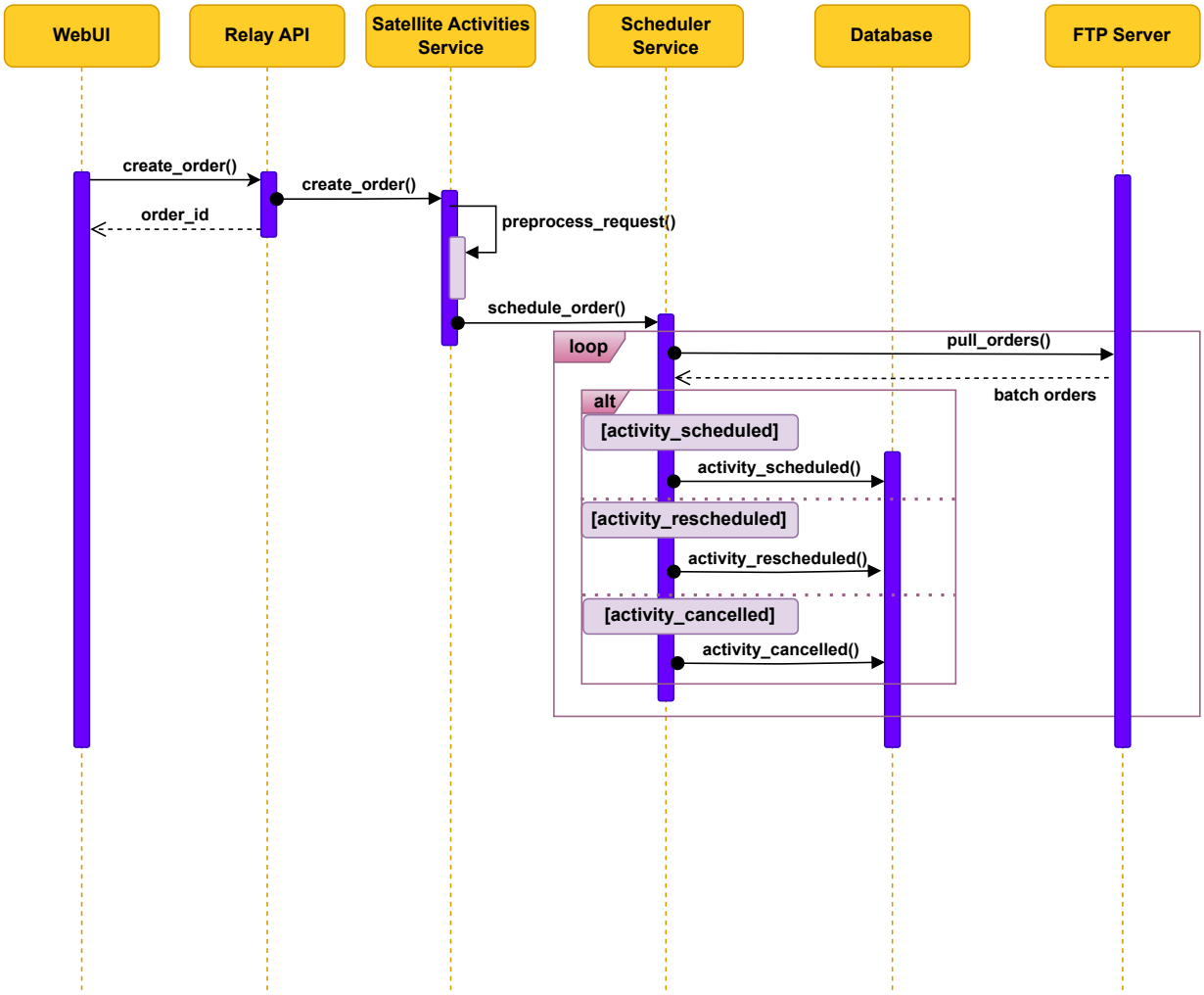


Figure 2.29: Order Scheduling Sequence of Operations

2.4.7 User Interface

The frontend of the system is organized into various views, each serving a specific operational function. Screenshots of these diagrams are available in the additional Figures Appendix A.1 User Interface.

In the maintenance activities view, tables are divided into categories: Memory Scrub, Orbit Maneuver, Parameter Update, and Diagnostic Activities. Each table includes fields for target satellite (which satellite will perform the activity), window start/end Time (the

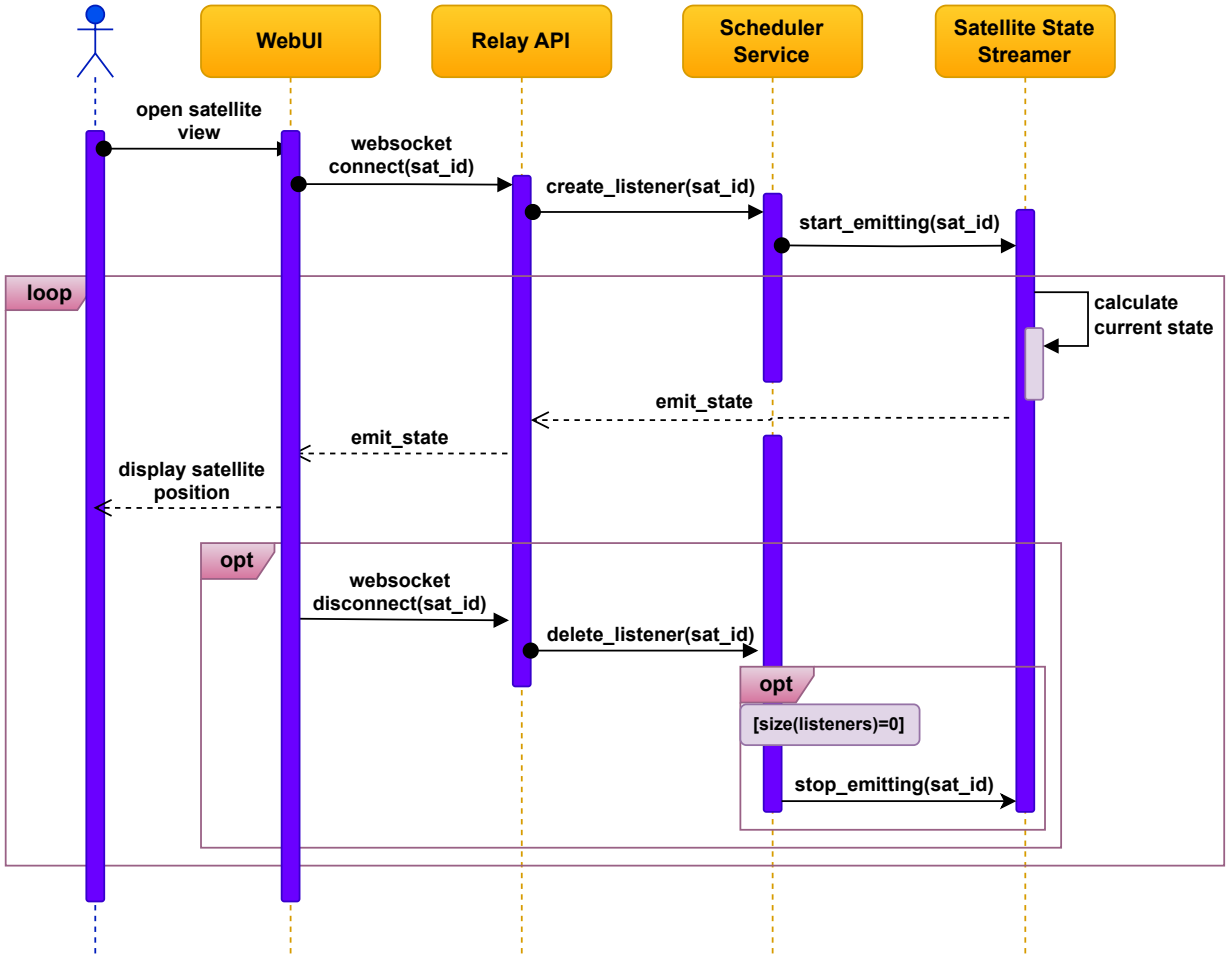


Figure 2.30: Satellite Visualization Sequence of Operations

time window for the activity in UTC-5 / UTC-4), duration (how long the activity lasts), repeat cycle repetition (how often the activity repeats within the period), and frequency minimum/maximum Gap (defining the frequency range for the activity, such as repeating every 2 to 4 hours). A payload outage boolean flag indicates if ongoing satellite activities need to stop during maintenance. A decline button is also provided for manual override of requests.

The image orders view features a table that displays image orders, detailing the Latitude/Longitude of the target location, priority (with 1 being most urgent), image start and end time (the time frame for imaging in UTC-5 / UTC-4), image type (resolution quality),

and revisit time (currently set to False, indicating no revisits). A decline button enables operators to manually override image requests.

The outage orders view features a table that displays the list of outage orders made which includes the target asset (satellite or ground station) and the target period (start and end time).

The schedule view presents different panels. The table view represents the captured events for each schedule, the timeline view shows the scheduled events in a Gantt chart-like format, and the system time panel allows the users to modify the current reference time of SatOpsMQ to help simulate orders in the future. Since an image order may include multiple recurring orders, a dedicated panel for schedule requests is provided to display the parent order alongside its associated child orders. This setup ensures that users can easily track and manage related orders in a consolidated view.

The asset view is comprised of two distinct panels. The first panel presents the real-time status of each ground station and satellite, with data streamed via web sockets to ensure timely updates. In this panel, the satellite table shows each satellite's name, available storage, maintenance status, outage status, and whether it's under regular sunlight or eclipse. The ground station table lists each station's name, the satellite it's in contact with, contact duration, and signal loss details. The second panel, known as the asset maps, offers both 2D and 3D real-time visualizations of the satellites' current positions in orbit relative to the ground stations. This dual-panel setup provides a comprehensive overview, facilitating effective monitoring and management of space assets. An add assets form is also included which allows adding new satellites and ground stations, featuring a drag-and-drop interface for satellite TLE files and a detailed form for ground stations, requiring operational specifics like name, coordinates, and communication rates.

The health dashboard provides a centralized monitoring interface to present key operational metrics for the SatOpsMQ system. Featured prominently is a progress indicator showing the percentage of orders scheduled. Below, a tabulated view titled 'Asset Order Amounts' displays the distribution of orders across five satellites, SAT1 through SAT5, indicating the volume of orders assigned to each asset. Adjacent to this is a section for 'System Alerts', designed to report anomalies, outages, or system messages that require attention.

2.4.8 CI/CD Pipeline

A Continuous Integration/Continuous Delivery (CI/CD) pipeline is a framework that automates the process of delivering software changes from development to production. In our

project, we used a yaml file to leverage GitHub Actions feature as our CI/CD tool, which is a powerful platform for automating workflows [19]. Using this pipeline, we regularly commit code changes to our GitHub repository, triggering automated build processes.

The main idea of this process is that when one of the SatOpsMQ developer commits code into any of the repositories, this action triggers a build notification that informs the team members about the start of the build process. The next step involves the backend server where the code is compiled. If the build is successful, the pipeline progresses to unit testing. Once the new code passes these tests, it moves on to the containerization phase using Docker, which packages the service into a container, making it ready for deployment in the staging environment. This container is then built, tagged with a version, and subjected to further integration, end-to-end, and load tests in the staging environment. This simulates the production environment and helps catch any errors before the release. Finally, once the application has successfully passed through all the checks and tests, the Docker image is tagged and pushed to an Amazon EC2 backend. From there, it can be deployed to the production environment, marking the completion of the CI/CD pipeline. This entire process can be visualized in this figure.

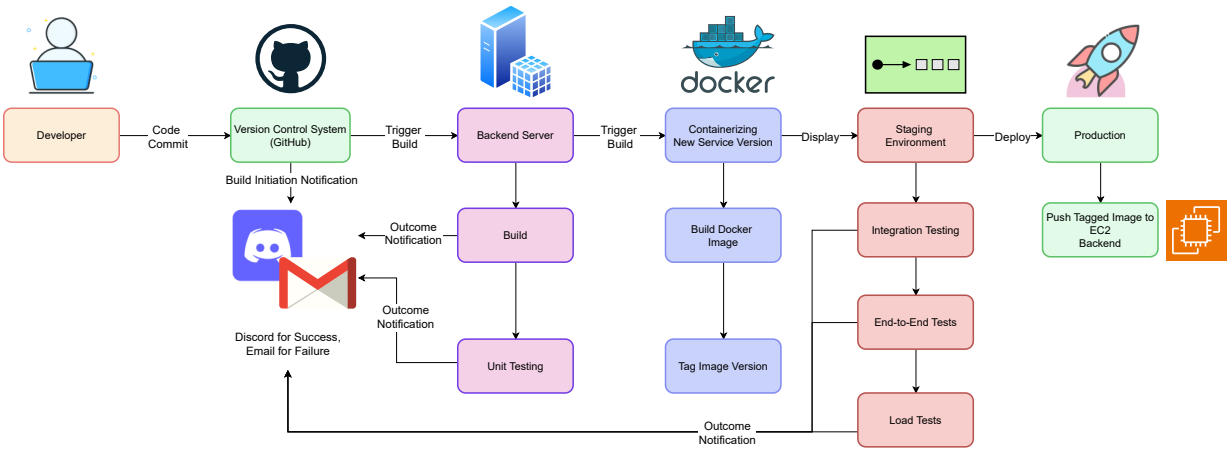


Figure 2.31: The CI/CD Pipeline of the SatOpsMQ's repositories. Note: The publishing of the image to EC2 is only applicable for the backend containers.

If the code build and deployment was successful, our pipeline triggers immediate notifications to our project server on Discord. These notifications are facilitated by a Discord bot that is connected to webhooks which enables real-time updates to be sent to the team. Alternatively, in the event of a failure, the person responsible for the changes will receive an email notification specifying that their recent push has not satisfied all required tests. This

setup ensures that any potential issues are promptly communicated, allowing the team to address them quickly and maintain the stability of our codebase.

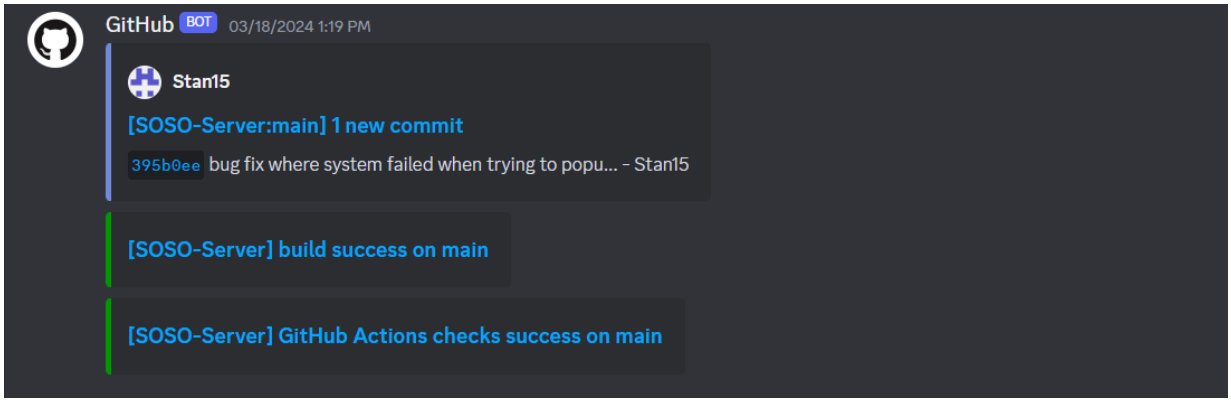


Figure 2.32: Example notification from the Discord Bot for a successful code change.

2.5 As-Built Design Compliance Analysis

2.5.1 Technical Design Budgets

Due to the open-source nature of the project, the CSA stakeholders highly advised against the usage of any software or tool that will cost extra. This is because, in an open-source environment, the goal is for other developers to contribute to our work at their leisure which is hard to meet if there's a cost associated with this participation. This is not a strict restriction in the sense that we cannot use any proprietary software at all, the only intent is to prevent the product from being too tightly coupled with specific paid software such that basic operations cannot be performed without it. In other words, if we are unable to run the system end to end without paying then the design approach is prohibited and considered infeasible. However, the CSA also emphasized that it is still allowed to use paid products, such as cloud services, to deploy the working product. This is what we did for the backend when we decided to launch the working backend in EC2. Even with the usage of the cloud, we still opted for the free tier version of EC2 which gave us 12 months free of charge with 750 hours per month under the t2.micro instance in the us-east-1 region. Still, we experienced some costs related to server backup which cost $0.05 per GB-Month of snapshot of data stored. Our server URL to access our backend resources also did not incur any cost as we decided to use the AWS-provided IP. To monitor each docker container,

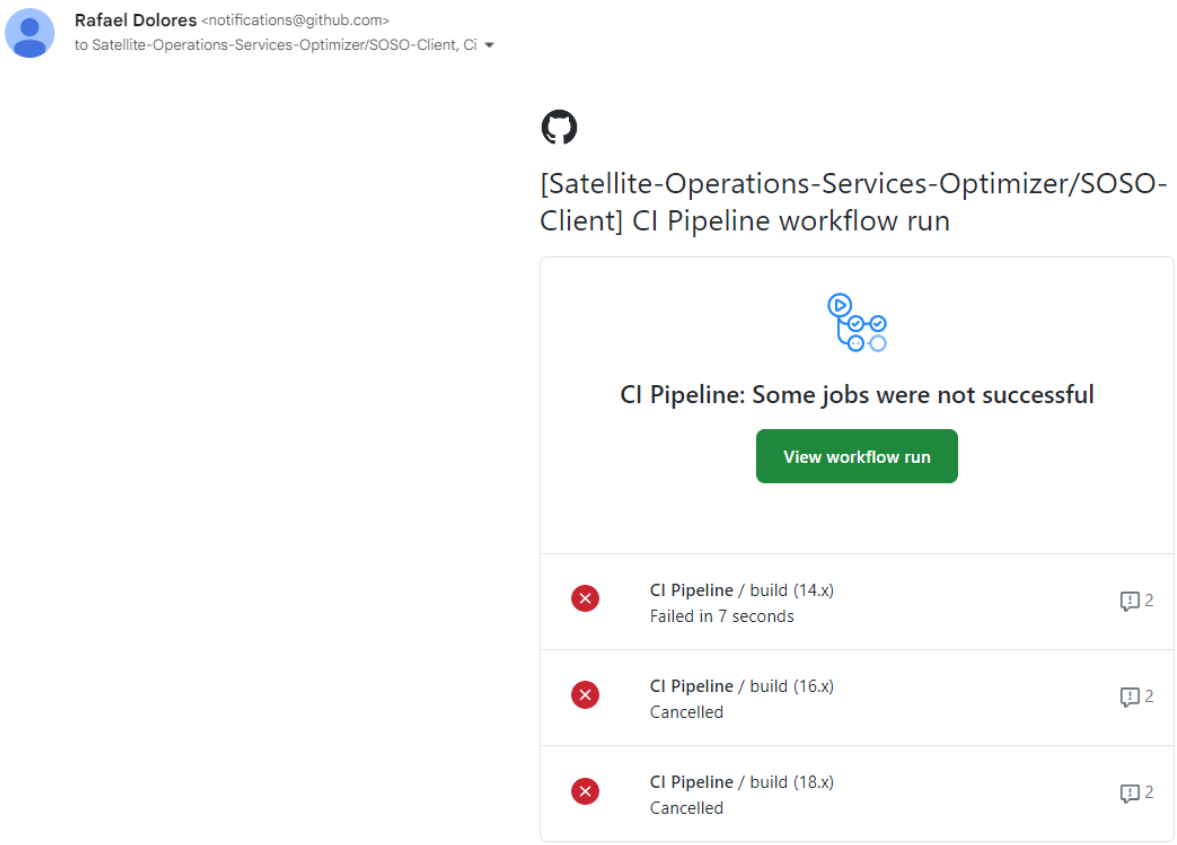


Figure 2.33: Example email notification for an unsuccessful code change.

AWS CloudWatch was used to display the execution logs which was also under the AWS Free Tier. The Table of Resource Usage and Costs below outlines the overall cost of the cloud products used by SatOpsMQ.

For the full comprehensive view of the overall costs of the infrastructure, see the System component costs table.

2.5.2 Computational Resources Usage

For the last month of development (March 2024), SatOpsMQ's overall computational resources usage reveals many key points about the system. CPU utilization maintained a low average of 2.17%, with no recorded peak surpassing 2.5%. This level of utilization con-

Resource	Specification	Consumption	Unit Cost	Total Cost
Elastic Compute Cloud (EC2)	Linux t2.micro instance-hour (free tier)	299 hrs	$0.00	$0.00
Elastic Block Store (EBS)	General Purpose (SSD) storage (free tier)	3.222 GB	$0.00	$0.00
EBS Snapshot Storage	Snapshot data storage (free tier)	1 GB	$0.00	$0.00
EBS Snapshot Storage (Additional)	Snapshot data stored - US East (N. Virginia)	1.933 GB	$0.05 per GB-Month	$0.10
CloudWatch Metrics	1,800 minutes of Live Tail usage per month	150 MB scanned data (max 5 GB)	$0.00	$0.00

Table 2.10: Resource usage and costs. Note that the consumption column is accurate as of April 2024.

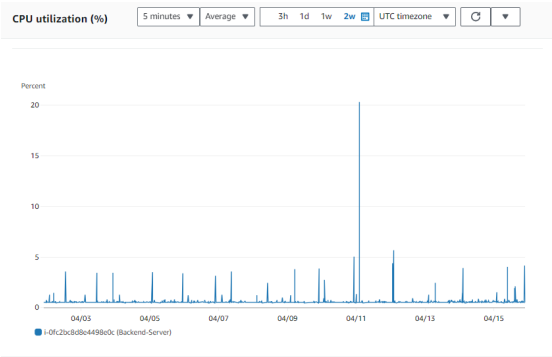
System Component	Description	Cost
Backend	Python, Node.js, AWS EC2	$0.10
Frontend	Next.js, Tauri	$0.00
Deployment	GitHub Actions	$0.00
Database	Postgres	$0.00
Monitoring	CloudWatch, Docker logs	$0.00
Miscellaneous	SSH Keypair	$0.00

Table 2.11: System component costs

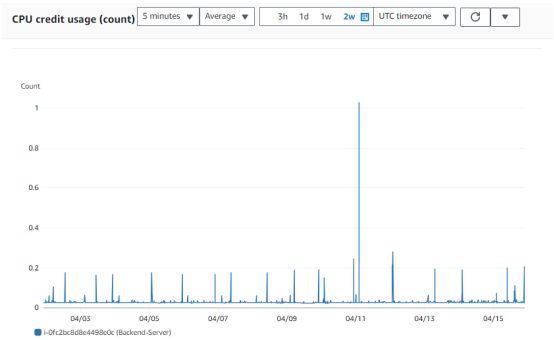
firms that the application is not CPU-intensive and is functioning well within the provided processing capabilities. Regarding network traffic, the incoming data maintained a steady flow with identifiable spikes that align with expected operational activities such as data

transfers or scheduled tasks. The maximum inbound traffic reached approximately 775k bytes, while outbound traffic peaked at 13.8k bytes. This asymmetry is characteristic of server workloads that primarily handle data input over output and is suggesting that network capacity is more than adequate for current demands. Packet traffic was thoroughly analyzed which demonstrates patterns consistent with the byte counts. Inbound packet traffic peaked at 532 counts, with outbound traffic displaying greater fluctuation and peaking at 83 counts, a normal observation that reflects varied communication exchanges with external sources. Lastly, CPU credit usage experienced only minor variations, adhering to the anticipated performance of a T2 instance. However, the CPU credit balance showed a gradual decline. While not immediately concerning, this will be continually monitored to confirm that it remains within an acceptable operational range thus pre-empting any potential impact on system performance. Full sized charts for these performance metrics are available in Appendix B.3 Amazon Web Services Metrics.

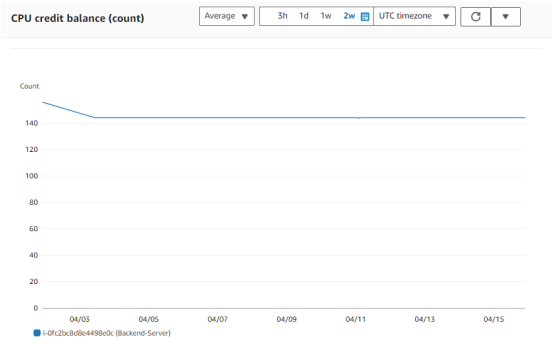
The resource utilization for our Elastic Block Store (EBS) volume has also been analyzed that spanned for a period of two weeks and has revealed many critical insights into the system's performance. Read and write throughputs have exhibited sporadic spikes; read throughput peaked at 102 KiB/s, while write throughput reached up to 104 KiB/s, signifying intermittent periods of high data transfer. This pattern aligns with the expected burstable performance of the EBS volume, which is designed to accommodate occasional surges in data transfer. Operation counts for both reads and writes have also shown variability, with read operations peaking at 3.27 Ops/s and write operations at 2.82 Ops/s. Average read and write latencies were well within the optimal range, maintaining a steady performance and ensuring efficient data access. The average latencies recorded were low, with read latencies not exceeding 10 ms/op and write latencies around 9.05 ms/op, demonstrating the volume's responsiveness. The average sizes for both read and write operations have been relatively stable, with read sizes averaging around 58 KiB/op and write sizes at about 47.4 KiB/op, further demonstrating a well-balanced and consistent I/O size that does not overburden the system. Notably, the burst balance has remained at 100%, confirming that the volume has not depleted its I/O credits and is primed for handling high I/O throughput if required. The average queue length remained consistently low, with an upper limit close to 0.01, indicating that the I/O operations are being processed without significant delay or backlog. Additionally, the time spent idle was consistently at 100%, which indicates that the EBS volume was readily available for I/O operations throughout the entire period without being overtaxed. Again, full sized charts are here.



(a) CPU Utilization by percentage

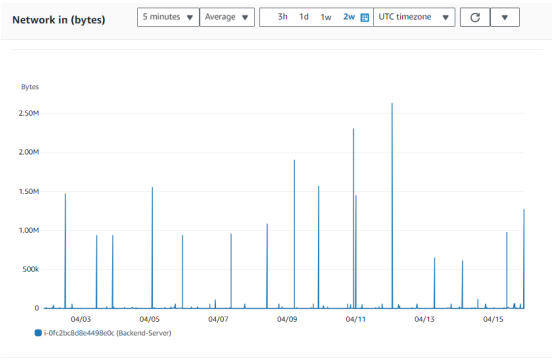


(b) CPU credits used

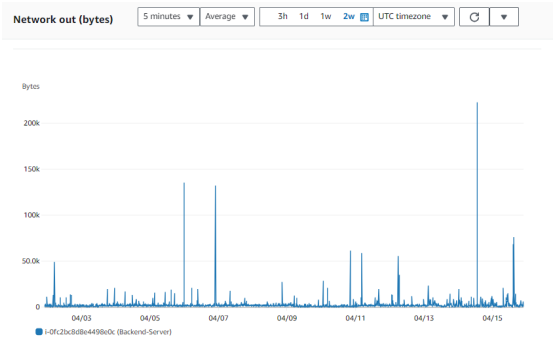


(c) CPU credit balance remaining

Figure 2.34: Overview of AWS EC2 Instance CPU Usage Metrics Over a Two-Week Period

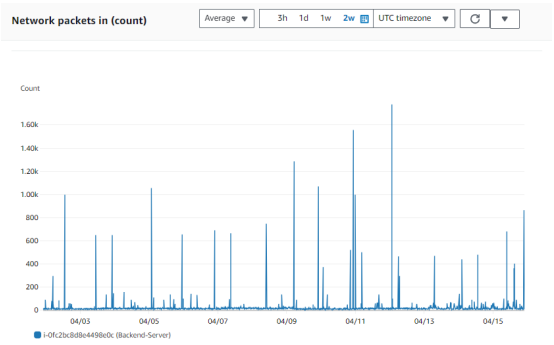


(a) Inbound network traffic by bytes

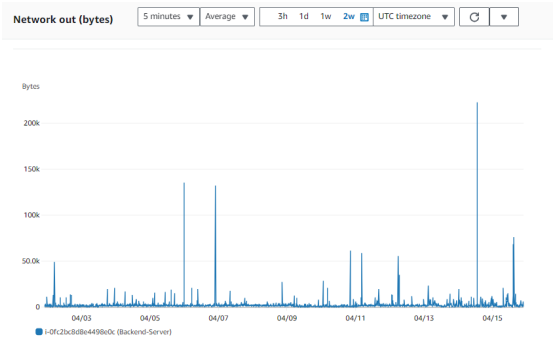


(b) Outbound network traffic by bytes

Figure 2.35: Overview of AWS EC2 Instance Network Network Traffic Over a Two-Week Period

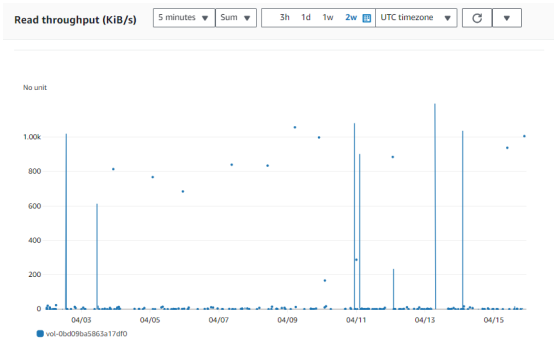


(a) Inbound network traffic by packets

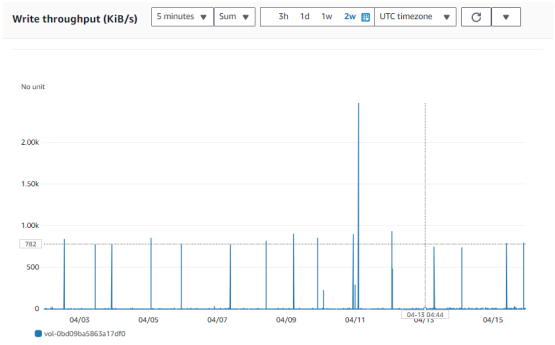


(b) Outbound network traffic by packets

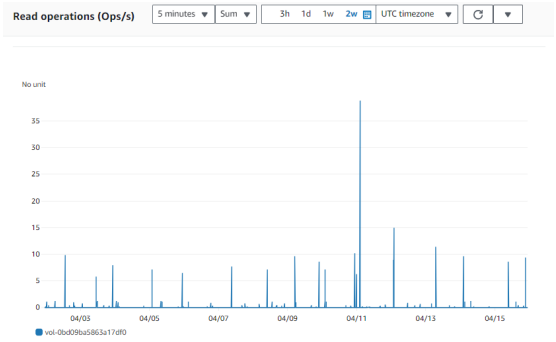
Figure 2.36: Overview of AWS EC2 Instance Packet traffic Metrics Over a Two-Week Period



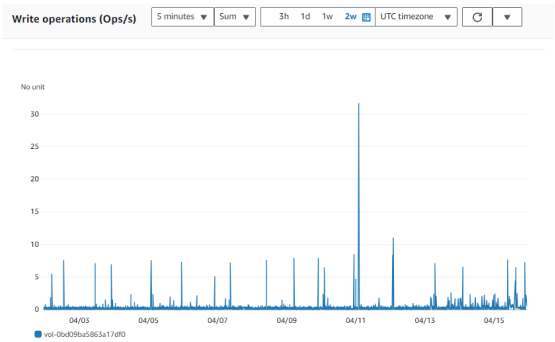
(a) Read Throughput in Kilobytes per second



(b) Write Throughput in Kilobytes per second

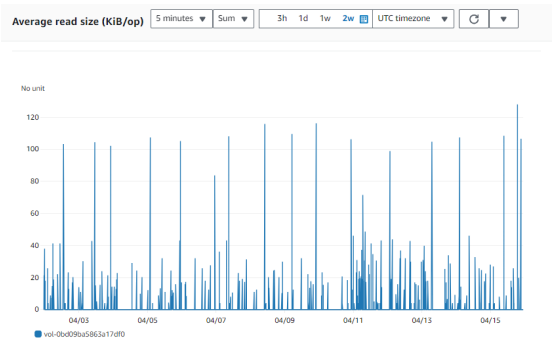


(c) Read Operations Per Second

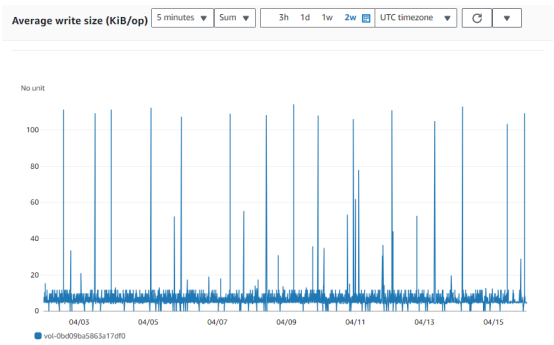


(d) Write Operations Per Second

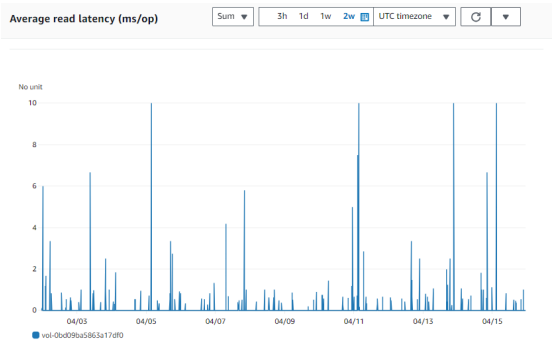
Figure 2.37: Overview of AWS EBS Volume Read/Write Metrics Over a Two Week Period



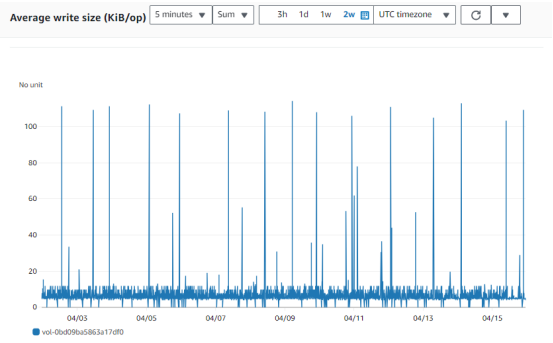
(a) Average Read Size in Kilobytes per Operation



(b) Average Write Size in Kilobytes per Operation

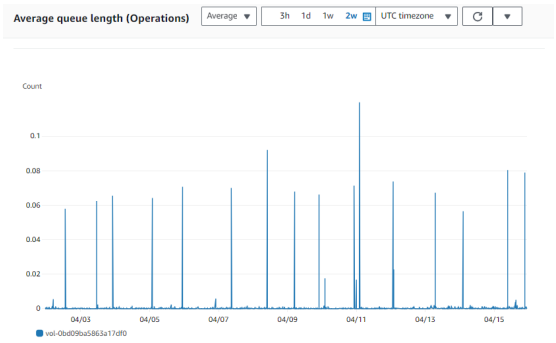


(c) Average Read latency in Milliseconds per Operation

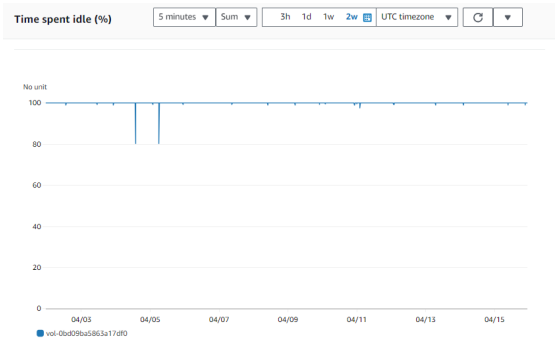


(d) Average Write latency in Milliseconds per Operation

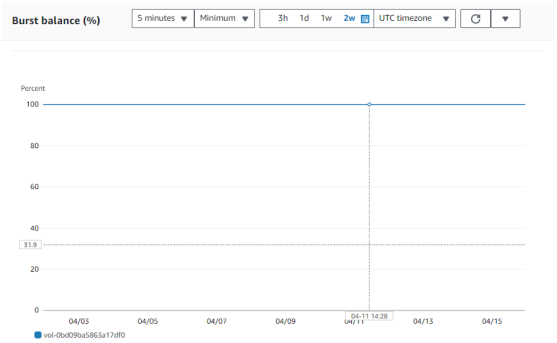
Figure 2.38: Overview of AWS EBS Volume Average Read/Write Metrics Over a Two Week Period



(a) Average Queue Length by Number of Operations



(b) Percentage of Time Spent Idle



(c) Percentage Burst Balance

Figure 2.39: Miscellaneous AWS EBS Metrics Over a Two Week Period

2.5.3 Algorithm Performance

SatOpsMQ underwent significant development with two key iterations of its algorithm: the initial alpha release and the subsequent beta to final release version. The performance and outcomes of each version were documented by storing the output in text files which were then presented to the CSA stakeholders for manual verification during weekly meetings. The feedback mechanism employed by CSA was straightforward and were generally categorized as either satisfactory ("looks good") or unsatisfactory ("does not look good"). The CSA was concerned about four key parameters: fulfilled order amounts, minimized resource consumption, equal asset usage, and execution time.

2.5.3.1 Alpha Release Version

The alpha release of the SatOpsMQ algorithm was primarily focused on establishing functional correctness which led to performances that were accurate yet not optimized for speed. This initial version was instrumental in shaping the basic operational framework but was characterized by slower execution times. The performance data was particularly concerning with the processing times relative to the number of orders processed which are tabulated. This data explicitly illustrates the quadratic nature of the algorithm's runtime.

Number of Orders	Processing Time (hours)
10	0.25
20	0.45
30	0.8
40	1.5
50	2.0
60	2.8

Table 2.12: Processing time as a function of order volume in the alpha release, illustrating the quadratic nature of the algorithm's runtime.

The runtime execution can be better visualized by this graph.

The equation for the line of best fit of the aforementioned graph along with its coefficients were derived mathematically as follows.

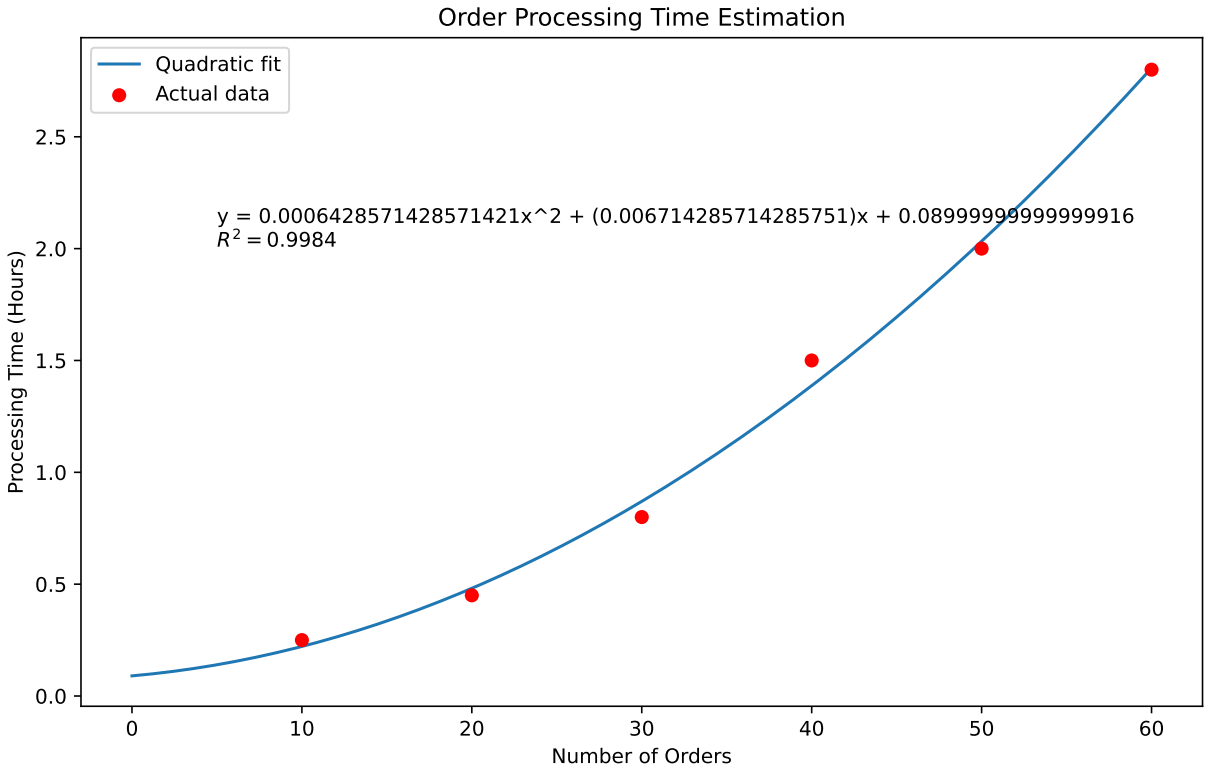


Figure 2.40: SatOpsMQ's Alpha Release Execution Time

Based on the actual run data, we have the following empirical values for y (processing times in hours) corresponding to the number of orders x:

$$x = [10, 20, 30, 40, 50, 60]$$
$$y = [0.25, 0.45, 0.8, 1.5, 2, 2.8]$$

We aim to fit these data points to a quadratic model of the form:

$$y = ax^2 + bx + c$$

This choice is motivated by several observations and theoretical considerations:

- A quadratic equation, with its parabolic nature, introduces just enough complexity to model both concave and convex relationships without overfitting the data with

higher-degree polynomials. This makes it a flexible choice for small to moderate datasets.

- Historically, similar types of data exhibiting growth and saturation phases have been successfully modeled using quadratic functions, which suggest phases of acceleration and deceleration in processing times that a linear model would miss.

The coefficients a, b, and c are calculated by solving the following system of normal equations derived from the method of least squares:

$$\begin{aligned} n \cdot a + \left(\sum x_i\right) \cdot b + \left(\sum x_i^2\right) \cdot c &= \sum y_i, \\ \left(\sum x_i\right) \cdot a + \left(\sum x_i^2\right) \cdot b + \left(\sum x_i^3\right) \cdot c &= \sum x_i y_i, \\ \left(\sum x_i^2\right) \cdot a + \left(\sum x_i^3\right) \cdot b + \left(\sum x_i^4\right) \cdot c &= \sum x_i^2 y_i. \end{aligned}$$

Where n is the number of data points. Solving these equations yields the coefficients a, b, and c.

The coefficient of determination, R^2, is calculated as follows:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

where SS_{res} is the residual sum of squares:

$$\text{SS}_{\text{res}} = \sum (y_i - \hat{y}_i)^2$$

SS_{tot} is the total sum of squares:

$$\text{SS}_{\text{tot}} = \sum (y_i - \overline{y})^2$$

$\hat{y}_i$ are the values predicted by our quadratic model, and $\overline{y}$ is the mean of the y values.

These calculations were done programmatically using Python by this script. Upon solving the equations, the derived coefficients and the R^2 value are displayed to evaluate the fit of the model to the data.

2.5.3.2 Beta To Final Release Version

The second and final release version of the algorithm underwent several optimizations that led to the improvement of the runtime. This table demonstrates the results after the applied optimizations and is better visualized by this graph.

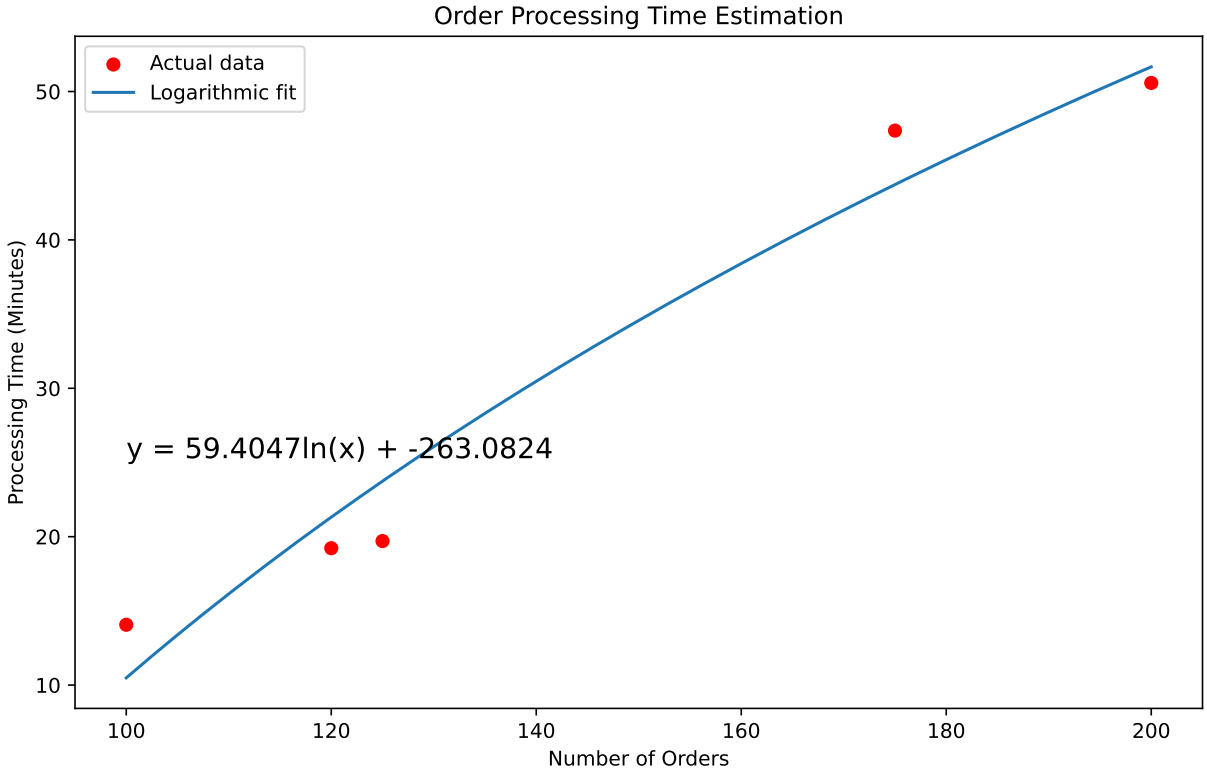


Figure 2.41: SatOpsMQ's Final Release Execution Time

Similar to the alpha release, the algorithm's runtime equations and coefficients were derived mathematically as follows.

The empirical data provided shows the total processing times for different numbers of events which suggests a potential logarithmic relationship between the number of events and the processing times.

The empirical values for processing times (y) corresponding to the number of events (x) are listed as follows:

$$x = [100, 120, 125, 175, 200],$$
$$y = [14.0652, 19.2236, 19.7064, 47.3653, 50.5773]$$

Given the nature of the data, a logarithmic function was deemed appropriate for modeling the relationship, defined as:

$$y = a \ln(x) + b$$

where a and b are parameters to be estimated through curve fitting.

The parameters were estimated using non-linear least squares to minimize the residuals between the observed values and those predicted by the model. The fitting process involves solving:

$$\min_{a,b} \sum \left(y_i - (a \ln(x_i) + b)\right)^2$$

The best-fit parameters obtained from the curve fitting are:

$$a = 59.4047, \quad b = -263.0824$$

The goodness of fit of the model is evaluated using the coefficient of determination, R^2, calculated as:

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \overline{y})^2}$$

where $\hat{y}_i$ are the values predicted by the model and $\overline{y}$ is the mean of the observed y.

The derived logarithmic model is:

$$y = 59.4047 \ln(x) - 263.0824$$

with an R^2 value indicating a strong fit to the data:

$$R^2 = 0.9601$$

This calculation was made possible by this python script.

Number of Events	Total Processing Time (hours)
100	14.065
120	19.224
125	19.707
175	47.365
200	50.577

Table 2.13: Total processing time as a function of the number of events, showcasing the operational efficiency and algorithmic complexity of the system.

2.5.3.3 CSA Feedbacks

For the alpha release, the CSA was thoroughly impressed with the progress, particularly with the amount of orders fulfilled and how minimal the resource consumption of the algorithm was. They noted that we were heading in the correct direction. However, they highlighted that the runtime was infeasible for real production use and needed further improvements. Moreover, the assets were not also being equally used, with some ground stations (i.e., Inuvik) dominating over the others.

The beta release saw a great improvement in the execution time while maintaining the same performance over the number of orders fulfilled and minimizing the consumption of resources. However, the equal usage of assets remained at unsatisfactory levels.

In the final release of the algorithm, the greedy approach was employed which saw the equal usage of assets metric to be fulfilled. The results of each feedback is consolidated in this table.

2.5.4 System Strengths

SatOpsMQ emerges as a robust and versatile desktop-based software system, its principal strength lying in its sophisticated ability to schedule and downlink satellite images within a diverse array of operational constraints. The system stands out for its configurable yet largely autonomous nature, requiring minimal operator intervention for routine processes, thus streamlining the workflow at the CSA. One of the key strengths of SatOpsMQ is its event-driven architecture, which facilitates real-time communication between satellites and ground stations. This approach allows for high responsiveness to the dynamic nature

Parameter	Objective	Alpha	Beta	Final
Order Amounts	Fulfill as many orders as possible, acknowledging that not all orders may be fulfilled.	S	S	S
Minimized Resource Consumption	Minimize the use of resources while achieving the goals.	S	S	S
Equal Asset Usage	Ensure equal utilization of space-based assets across operations.	U	U	S
Execution Time	Runtime must be kept at a viable range.	U	S	S

Table 2.14: Evaluation of optimization parameters by CSA across Alpha, Beta, and Final releases. S = Satisfactory, U = Unsatisfactory

of satellite operations and ensures that all system activities are coordinated efficiently, leveraging the Advanced Message Queuing Protocol for reliable data transmission. The user interface of SatOpsMQ, tailored for operators, provides a critical balance between automation and manual oversight. It enables users not only to monitor the system's state effectively but also to exert control by overriding automated operations when necessary. This capability ensures that human expertise remains at the core of decision-making processes, enhancing the system's operational safety and flexibility.

Another significant strength of SatOpsMQ is its open-source development model. With its source code hosted on GitHub, the project benefits from collaborative contributions and continuous peer review, fostering innovation and quality enhancements. This transparency also supports the system's adaptability and scalability, as it allows sponsors and stakeholders to anticipate and address the technical risks associated with expanding satellite and ground station assets. Moreover, SatOpsMQ is designed to function on basic computer hardware, which reduces the barriers to deployment and operational costs. This broadens the system's accessibility and potential for wider adoption within the CSA's infrastructure. It also opens avenues for SatOpsMQ to serve as a communication hub for data transfer across various systems, demonstrating the foresight in its design to accommodate growth and future applications beyond its initial scope. SatOpsMQ demonstrates a series of notable strengths in its performance and resource management, as evidenced by recent monitoring over two weeks. The system exhibits excellent efficiency in CPU utilization,

maintaining low averages that suggest an effectively managed workload without signs of over-exertion. This efficiency underscores SatOpsMQ's capacity for processing complex operations without incurring additional computational costs or necessitating further optimization for current operational levels. Network traffic, both inbound and outbound, reveals a well-tuned balance between data ingestion and output, crucial for maintaining the system's robustness in real-time satellite communication tasks.

The low variability and peaks in network traffic align with the expected operational patterns, ensuring that the system can sustain its communications without latency or bandwidth issues. The read and write throughput on the Elastic Block Store (EBS) volume indicate a well-provisioned I/O performance, with peaks reflecting expected bursts in activity. Such performance is critical for the data-intensive operations of SatOpsMQ, where timely data retrieval and storage are paramount. Moreover, the burst balance remaining at full capacity is a testament to the system's readiness to handle high I/O throughput, giving it the agility to respond to increased demands instantaneously. The consistency in read and write latencies further cements the system's reliability. With latencies well within optimal thresholds, SatOpsMQ ensures efficient data access and transfer, a core requirement for a system handling satellite data where every millisecond can be critical.

2.5.5 System Weaknesses

Despite its robust design, SatOpsMQ encounters several challenges inherent to its architecture and operational environment. A primary concern is the potential for data inaccuracies which may lead to stale or unusable data. As an event-driven, open-source system, it relies heavily on the integrity of data feeds, which, if compromised, could adversely affect the system's decision-making processes and overall reliability. Selecting appropriate data transmission protocols also poses a challenge due to the multitude of options available and the rapid evolution of technology. The need to maintain flexibility while ensuring secure and efficient data flow is critical especially given the system's role in the timely management of satellite operations. Scalability issues present another weakness. While SatOpsMQ is designed to be scalable, the introduction of additional assets such as new satellites and ground stations necessitates careful planning and resource allocation. Ensuring that the system can handle an expanded network without degradation of performance is a concern that requires ongoing attention.

Moreover, the reliance on the Advanced Message Queuing Protocol, while advantageous for communication, may limit the system's ability to integrate with alternative protocols or newer technologies without significant adaptation. This could potentially hinder the

system's long-term scalability and interoperability with other platforms within the CSA's infrastructure. The open-source nature of the project, while fostering transparency and collaboration, also introduces risks associated with code quality and security. The system's codebase must be diligently reviewed and maintained to prevent vulnerabilities that could be exploited, a challenge compounded by the varying levels of contributions from the community. Lastly, the system's current capability to operate on basic computing hardware, although a strength, may become a weakness as computational demands increase.

With regards to the AWS infrastructure supporting SatOpsMQ, our analysis identifies certain performance metrics that suggest opportunities for optimization. There are observable variabilities in I/O throughput, with spikes in both read and write activities indicating intermittent demand surges. Although these surges fall within the system's current capacity, they highlight the necessity for enhanced I/O consistency. This enhancement is vital for supporting unexpected system load increases that could otherwise disrupt user experiences. Latency metrics reveal fluctuations in both read and write operations. Even though these remain within acceptable parameters, their variability warrants attention, especially in scenarios involving high-load operations or the processing of substantial contiguous data sets. The potential impact on time-sensitive satellite data operations makes it imperative to reduce these latency variations to uphold a steadfast and dependable service. Concerning burst balance, the EBS volume's 100% maintenance is commendable; however, its management is critical for sustaining system responsiveness. Effective monitoring and regulation of burst balance are essential to prevent performance impediments and to accommodate abrupt data transfer spikes without compromising latency or throughput. The volume's idle time, consistently at 100%, suggests an overprovision of I/O capacity, hinting that the resource is under utilization. This overprovisioning, while indicating a non-pressured system, also raises concerns about potential inefficiencies and unnecessary expenditure that may result from overallocation.

Lastly, the operation counts for read and write processes show inconsistencies, which might signal batch processing or irregular distribution of workload. This irregularity could become a systemic weakness if not addressed, potentially leading to temporary I/O bottlenecks and consequent performance degradation. The system must manage load distribution more evenly to ensure sustained I/O performance and system reliability. Addressing these points will be key in refining SatOpsMQ's AWS infrastructure to not only ensure optimal performance under varying conditions but also to manage costs effectively and maintain system reliability.

2.5.6 Testing

2.5.6.1 Testing Procedure

The testing process across all releases are almost similar. Given that the system is purely digital and without attachments to hardware entities, we followed a software-based testing procedure. Our software testing goes through 3 phases: unit testing, integration testing, and end-to-end testing. The differences between each release is in the extent to which of the testing phases were completed or added on to.

Each testing type differs in its scope. Unit testing is focused on testing the atomic units of the system, ensuring the proper functioning of the individual functions. Moreover, unit testing ensures that the system base of functioning is solid and not susceptible to errors or bugs. The absence of unit testing ensures no guarantee of functioning of any component of the system.

After starting unit testing, integration testing can begin. Integration testing is focused on verifying the flow between the components/units examined in unit testing. If unit testing tested functions A and B, integration testing would ensure the proper data flow between data going into A and afterwards into B. Basically, integration testing would group relevant units together and test them as a whole. Integration testing guarantees that the full fledged features of the system work as intended. The absence of integration testing entails the failure to guarantee proper functioning of advertised features, as well as high potential of failure at satisfying the technical and user requirements.

Finally, after unit and integration testing, we begin and conclude with end-to-end testing. End-to-end testing focuses on the system as a whole, and tests the functioning from start to finish. Absence of end-to-end testing results in a severely disconnected system, with some aspects functioning and others failing, with no overview as to why flow of features are malfunctioning.

The procedure outlined previously is implemented using the python unittest library that allows utilizing testing tools/functionalities to test the functions against an expected output. The library helps in also displaying summaries of results after running the tests, and outline whichever of the tests failed to run and how they differ from the expected output outlined by the tester.

The process outlined is the formal process followed for most of the system's functionalities, except for one. The scheduling functionality within the system does not entirely follow the formal process outlined in the previous paragraph. The algorithm responsible for taking creating a schedule for the satellites follows an informal testing process.

The team regularly meets with engineers from the Canadian Space Agency (CSA), any changes or "tests" conducted in regards to the scheduling algorithm is brought to their attention. Their feedback on what was performed is informally the result of testing the algorithm.

Therefore, most of the system follows the formal process of performing unit, integration, and end-to-end testing. Meanwhile, the algorithm functionality in the scheduler component of the system, is informally tested through communicating it's functionality and outputs to specific scenarios provided by the CSA, their feedback on our algorithm's performance is our testing metric.

The testing procedure overall is very effective as it follows the software engineering standard in testing which consists of testing the system at various scopes (as explained in regards with unit, integration, and end-to-end). This ensures proper functioning of the purely software aspects of the system. Meanwhile, with aspects pertaining to space-related matters, such as the scheduling algorithm's functionalities, we consult with the CSA on these matters. The CSA with all their expertise in the matter, can provide valuable insights on what needs to be adjusted, and hence guarantees an effective and accurate function of the system.

This testing procedure guarantees verifiability of the system given that every function within the system is tested at various scopes and with various scenarios. Every function is tested for it's correctness through specific scenarios that we set, and all the outputs are compared against an expected correct output. This ensures every function is functioning correctly. Moreover, because the functions are tested within various scopes (individually tested and tested with other functions), the overall functionality of the system as a whole is thoroughly tested as well.

This testing procedure guarantees validation given that there are frequent meetings conducted with the CSA. The CSA acts as the main customer, and therefore through these frequent meetings, we update them on what was done and implemented, and we get their immediate feedback. Any issues or concerns raised from their end is immediately tackled and dealt with. Therefore, the system is validated on a regular basis.

The structure of this subsection is as follows; each release would have started some phase and completed another which would be the various differences we'd highlight across each of the release testing process subsections. In each Release Testing Process subsection, we examine what phase of testing began, or was completed, and broadly which aspects of the system were tested.

After explaining the testing procedures for every release, we delve into the functioning of the pipeline we've set up, and it's relevance to how it maintains system configuration

and tracks failing processes.

Lastly, we showcase a basic table outlining every function that was tested. We specify in which release said function was tested, we highlight the testing phase it was tested through (unit, integration, or end-to-end), and finally we conclude this entire subsection with the testing results.

2.5.6.2 Failure Process Tracking

As mentioned in the CI/CD section, we initiate a pipeline by pushing our code changes to a shared GitHub repository. This pipeline also functions as our main failure tracker. The pipeline utilizes error logging within microservices to capture exceptions and errors. Depending on the nature of the error, issues tracked by the pipeline is subjected to manual inspection by the SatOpsMQ team which leads to the issues being posted in the repository's issue page. Automated rollback mechanisms are employed to ensure rapid restoration to a stable state in case of critical deployment failures.

2.5.6.3 Issue Tracking

Issues or bugs encountered throughout any phase or release of testing is reported through the GitHub platform within the issues section of the repository. These issues are not only communicated through the platform but also directed to the relevant team members.

The issue tracking process involves providing an issue with a title, and a small description surrounding the issue/bug. The frequently used format for an issue description is to first identify it's location within the code, explain what occurs when the error appears. Afterwards, a quick explanation on how the issue can be replicated, and possible assumptions/theories from the tester as to why the error occurs. Finally, the tester can provide a possible solution to fix the issue, and assign the GitHub issue to the relevant team member who was in charge of the code that is bugged. An example of a GitHub Issues page can be found in this link.

Event Relay P1	
getAllGroundStations	
test_01	Test that API returns all G.S's
getGroundStationByID	
test_01	Test that correct G.S. is returned by ID
test_02	Fed faulty input of nonexistent ID
newGroundStation	
test_01	Create a new G.S. with correct params
test_02	Fed incorrect G.S. params (wrong attr)
test_03	Fed incorrect G.S. params (missing attr)
test_04	Fed incorrect G.S. params (empty)
createImageOrder	
test_01	test image order creation
test_02	fed wrong input (wrong attr)
test_03	fed wrong input (missing attr)
getAllImageOrders	
test_01	test getting all orders

Table 2.15: Event Relay API Test Table Part 1

Image Management Service	
handleImageOrders	
test_01	handle valid order with recurrence
test_02	handle valid order without recurrence
test_03	handle invalid input (incorrect schema)
test_04	handle invalid input (empty input)

Table 2.16: Image Management Test Table

Scheduler	
ensureEclipseEventsPopulated	
test_01	Test with a 12 hour window
test_02	Test with an invalid window (end before start)
test_03	Test with an invalid window (start = end)
ensureContactEventsPopulated	
test_01	Test with a 12 hour window
test_02	Test with an invalid window (end before start)
test_03	Test with an invalid window (start = end)

Table 2.17

2.5.6.4 Unit Tests

2.5.6.5 Integration Tests

2.5.6.6 Unit & Integration Testing Results

Below you'll find the outputs of running the respective tests on each of the microservices. This figure corresponds to the test results of the event-relay-api service. The last-minute deprecated image-management service had the following test results. Scheduler service related tests are consolidated in this screenshot.

```
Database tables automapped and exported to global namespace
test_01_get_all_ground_stations (test.TestAssetRoutes.test_01_get_all_ground_stations) ... ok
test_01_get_ground_station_by_id (test.TestAssetRoutes.test_01_get_ground_station_by_id) ... ok
test_01_new_ground_station (test.TestAssetRoutes.test_01_new_ground_station) ... ok
test_02_get_ground_station_by_id (test.TestAssetRoutes.test_02_get_ground_station_by_id) ... ok
test_02_new_ground_station (test.TestAssetRoutes.test_02_new_ground_station) ... ok
test_03_new_ground_station (test.TestAssetRoutes.test_03_new_ground_station) ... ok
test_01_create_image_order (test.TestImageRoutes.test_01_create_image_order) ... ok
test_02_create_image_order (test.TestImageRoutes.test_02_create_image_order) ... ok
test_03_create_image_order (test.TestImageRoutes.test_03_create_image_order) ... ok
test_04_get_all_image_orders (test.TestImageRoutes.test_04_get_all_image_orders) ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.314s

OK
```

Figure 2.42: Test Results of the Event Relay API

```
Database tables automapped and exported to global namespace
test_01_handle_image_orders
(IntegrationTests.TestImageManagementService.test_01_handle_image_orders) ... ok
test_02_handle_image_orders
(IntegrationTests.TestImageManagementService.test_02_handle_image_orders) ... ok
test_03_handle_image_orders
(IntegrationTests.TestImageManagementService.test_03_handle_image_orders) ... ok
test_04_handle_image_orders
(IntegrationTests.TestImageManagementService.test_04_handle_image_orders) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.084s

OK
```

Figure 2.43: Test Results of the Image Management Service

You'll also find the status of the aggregate testing performed through the pipeline in this screenshot.

```
Database tables automapped and exported to global namespace
test_01_ensure_contact_events_populated
(test.SchedulerTests.test_01_ensure_contact_events_populated) ... ok
test_01_ensure_eclipse_events_populated
(test.SchedulerTests.test_01_ensure_eclipse_events_populated) ... ok
test_02_ensure_contact_events_populated
(test.SchedulerTests.test_02_ensure_contact_events_populated) ... ok
test_02_ensure_eclipse_events_populated
(test.SchedulerTests.test_02_ensure_eclipse_events_populated) ... ok
test_03_ensure_contact_events_populated
(test.SchedulerTests.test_03_ensure_contact_events_populated) ... ok
test_03_ensure_eclipse_events_populated
(test.SchedulerTests.test_03_ensure_eclipse_events_populated) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.071s

OK
```

Figure 2.44: Test Results of the Scheduler Service

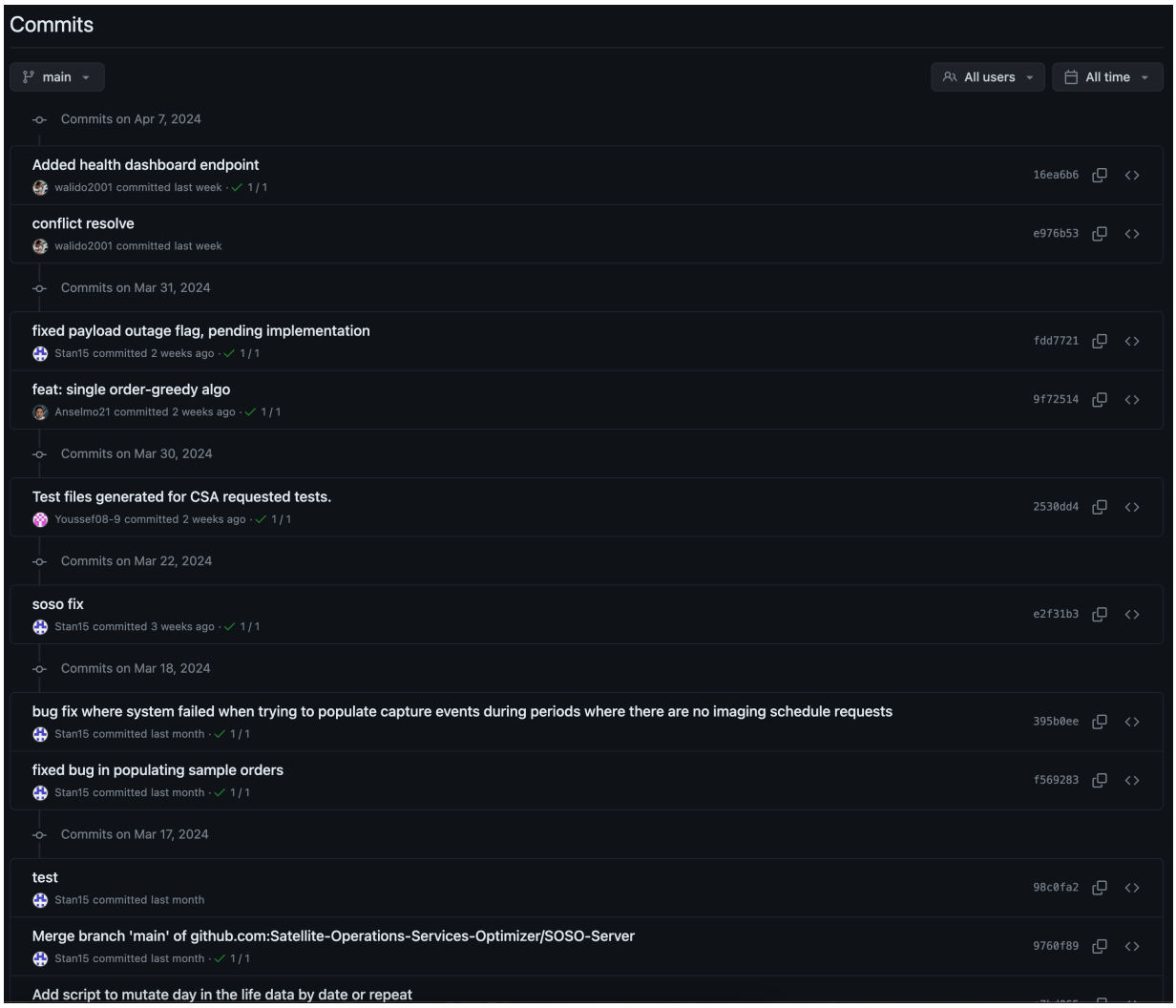


Figure 2.45: Passed Pipeline Tests visible through the checkmark next to each commit

2.5.6.7 Edge Case Testing

In order to test our system properly, we had to be sure that it is able to handle edge cases. To test for the proper handling of edge cases by our system in the system, we manually created test environments, setting up a scenarios carefully crafted to have specific features which test edge cases of the system.

Provided in the figure below is an example of one such test environment which we hand-crafted to test some edge-cases of the scheduler system when it is identifying available slots in to which some image order can be scheduled. For context, given this event is an imaging event, it has to take place when the satellite can capture an image of the requested location - hence, the imaging request must be scheduled in such a way that it starts on a capture opportunity. Keeping this in mind, we can see in the figure that we have three candidate time slots to schedule this event. However, the first candidate slot is invalid as it doesn't give enough time for the imaging event to take place - the event would overlap a time period when the satellite is in outage if it is scheduled at that time. However, we can see that there are two other opportunities where we can schedule the imaging event.

After having created this testing environment, we create an image order with this testing environment as background context, then pass it through the module of the scheduler system responsible for identifying available time slots. We then test that the output returned by this scheduler module matches our expectations.

In this specific scenario outlined in the figure, we test three things:

1. we assert that the scheduler output does not include the first candidate slot, which is an invalid slot
2. we assert that the scheduler output includes the other two candidate slots, which are both valid slots
3. we assert that the scheduler does not output any other candidate slots

This test environment we have described is a simpler example of more complex test environments which we manually created to test edge cases. For example, we have provided a figure showing the passing test report for a more complex test environment. It tests the module which is meant to identify the times when we can uplink an order to the satellite, as well as the times when we can downlink payload from the satellite down to the groundstation. As you can see, we carefully crafted the test environment to create specific opportunities for uplink, as well as downlink, including edge-cases where the system is supposed to know not to schedule an uplink or a downlink. These edge cases are tested using this testing environment, and as you can see in the provided figure, several edge

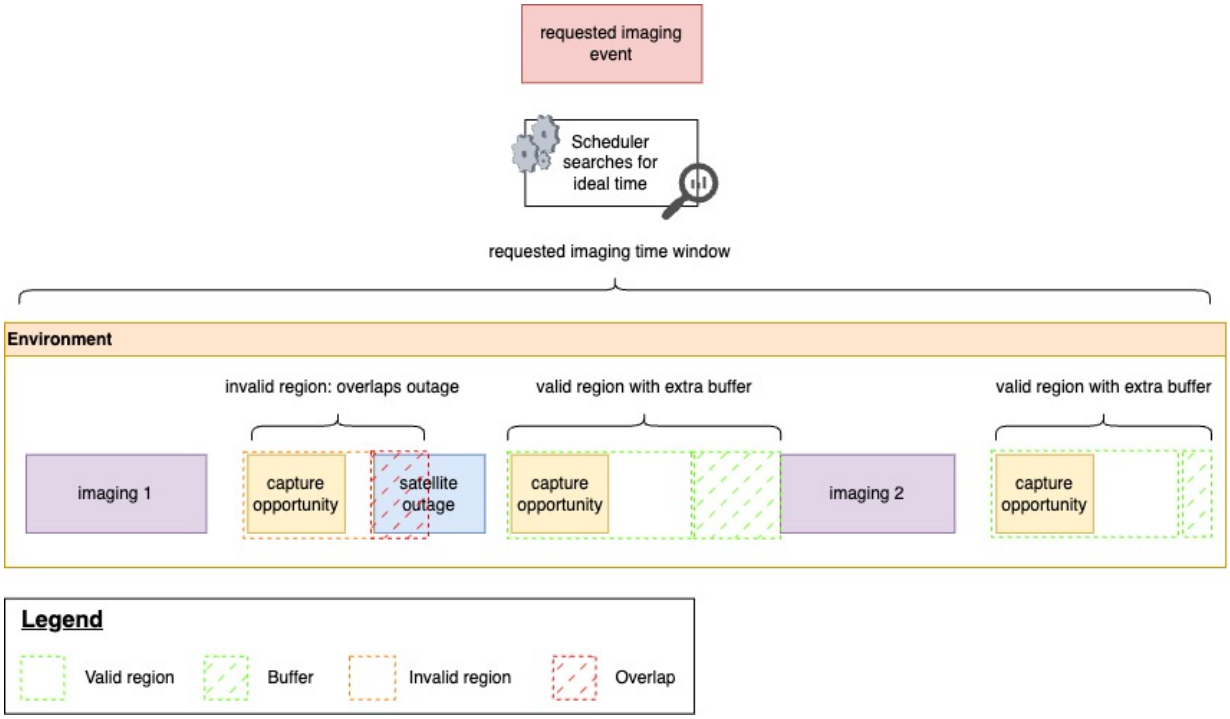


Figure 2.46: Hand-crafted test environment to test for schedule slot identification

cases were tested, and passed. This kind of testing gives us a level of confidence in the correctness of our algorithm.

Though these test environments can get quite complex, even the more complex of these test environments have a limitation. Due to the exploding number of possible ways we can schedule events to take place, and the time-consuming nature of manually creating these test environments, it is not feasible to manually create test environments on the scale of what would be expected in real-world scenarios. These test environments only model isolated categories of edge cases. However, edge cases in the real world can compound on each other, and create even more complex scenarios. It would not be possible to capture all of these complex scenarios manually. We thus decided to use another approach to test our system with real-world scenarios.

2.5.6.8 End-to-end testing with Real-world Scenarios

In order to end-to-end test with real-world scenarios then, we focused on aggregating statistics on the scheduler's output, such as the total number of orders scheduled, the

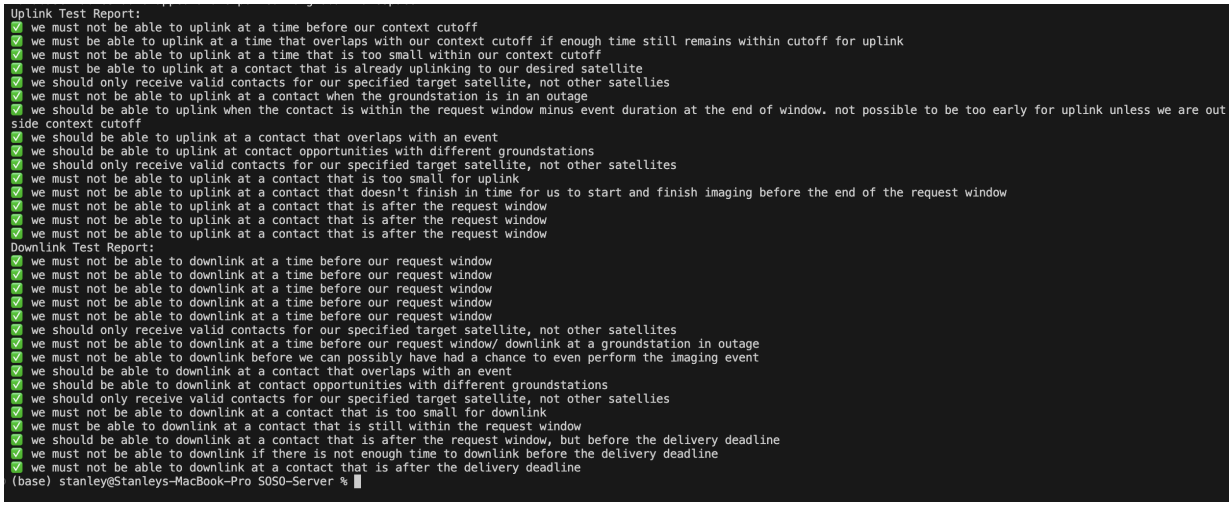


Figure 2.47: Test Report for Identifying Candidate Capture Opportunities

number of orders rejected, the number of events assigned to each satellite and ground-station, e.t.c.

To gather these aggregated statistics on certain inputs to our system, we relied both on sample orders which we manually generated, as well as on sample orders which we were provided with by the CSA. The orders we were provided with by the CSA, we were told, was representative, in quantity and complexity, of orders they would receive in their real-world system on a typical day - we will refer to these orders provided by the CSA as day-in-the-life (DITL) orders henceforth.

Below, we will go over some statistics we gathered from running sets of DITL orders we were provided through our scheduler system, as well as how these statistics helped us identify deficiencies in our system, an how we resolved these deficiencies.

2.5.6.9 Imbalanced Asset Workload Distribution

One scenario where we found great use in this approach of testing is in identifying that our scheduler system needed to be adjusted to better balance the distribution of scheduled events across satellites and ground-stations. Initially, when we ran the DITL orders through our system and printed out the logs, it was pointed out to us by our sponsors at the CSA that most imaging orders were scheduled to the Gatineau groundstation. From these logs, we were able to get this valueable insight which led us to adjust our algorithm to better

balance allocation of workload. This issue is fully documented in the non-conformance disposition section.

The figure below shows a snippet of the logs produced after the algorithm had been adjusted to balance for workload distribution. The full execution log is available for viewing in this text file.

Ground Station Allocation:

Station	Number of slots
Inuvik Northwest Territories	32
Prince Albert Saskatchewan	31
Gatineau Quebec	31

Sattelite Allocation:

Sattelite	Number of slots
SOSO-1	18
SOSO-2	17
SOSO-3	18
SOSO-4	20
SOSO-5	21

Figure 2.48: Logs showing statistics on DITL orders, after correcting for balanced workload distribution

2.5.6.10 Successful Rescheduling During Satellite Outages

We also desired to test if our system correctly handled satellite outages. As outages result in a series of cascading events, (rescheduling of events overlapping with the outage) using

logs are a good way to test this system end-to-end. By running a batch of DITL orders which contained outages through our system, we were able to track how our system handled outages. As can be seen in the attached figure, during the course of running a set of DITL orders through our system, we logged some statistics regarding the system's output, which can be seen in this figure.

```
Step 7: Ingest OrbitMaintenance6 and OrbitMaintenance7
Database tables automapped and exported to global namespace
Waiting for a while (max 10 secs) to allow scheduler system to notice new orders and kick off...
Starting scheduler timer...
Scheduler finished in 0 seconds.
Creating report...
==========Overall stats==========
Total order count: 224
Total request count: 294
   Scheduled requests: 258
     5: Reason: Scheduled request was displaced due to an outage at satellite id=2. Succesfully scheduled request.
     21: Reason: Scheduled request was displaced due to an outage at groundstation id=1. Succesfully scheduled request.
     232: Reason: Successfully scheduled request.
   Rejected requests: 36
     9: Reason: No available time slots for this event.
     2: Reason: Scheduled request was displaced due to an outage at satellite id=2. No available time slots for this event.
     2: Reason: Spent too much time trying to find a valid schedule plan. Failure reason during last try: There were available satellite time slots, and available groundstation contacts, but no valid matching of uplink
gs contact -> satellite time slot -> downlink gs contact could be found.
     1: Reason: Scheduled request was displaced due to an outage at groundstation id=1. Spent too much time trying to find a valid schedule plan. Failure reason during last try: No contacts found to downlink this
request.
     22: Reason: No satellites can capture this area during the requested time window.

======Breakdown by order type======
Total Imaging order count: 200
Total Imaging request count: 200
   Scheduled requests: 167
     5: Reason: Scheduled request was displaced due to an outage at satellite id=2. Succesfully scheduled request.
     21: Reason: Scheduled request was displaced due to an outage at groundstation id=1. Succesfully scheduled request.
     141: Reason: Successfully scheduled request.
   Rejected requests: 33
     1: Reason: Scheduled request was displaced due to an outage at satellite id=2. No available time slots for this event.
     7: Reason: No available time slots for this event.
     2: Reason: Spent too much time trying to find a valid schedule plan. Failure reason during last try: There were available satellite time slots, and available groundstation contacts, but no valid matching of uplink
gs contact -> satellite time slot -> downlink gs contact could be found.
     1: Reason: Scheduled request was displaced due to an outage at groundstation id=1. Spent too much time trying to find a valid schedule plan. Failure reason during last try: No contacts found to downlink this
request.
     22: Reason: No satellites can capture this area during the requested time window.
Total Sat_outage order count: 1
Total Sat_outage request count: 1
   Scheduled requests: 1
     1: Reason: Successfully scheduled request.
Total Maintenance order count: 22
Total Maintenance request count: 92
   Scheduled requests: 89
     89: Reason: Successfully scheduled request.
   Rejected requests: 3
     2: Reason: No available time slots for this event.
     1: Reason: Scheduled request was displaced due to an outage at satellite id=2. No available time slots for this event.
Total Gs_outage order count: 1
Total Gs_outage request count: 1
   Scheduled requests: 1
     1: Reason: Successfully scheduled request.
```

Figure 2.49: Logs showing statistics on the scheduler's result after an outage helps ensure expected behaviour takes place.

As can be seen in the logged statistics for a batch of DITL orders, the ingested outage orders resulted in orders being rescheduled. If you look at the section of the log titled "Breakdown by order type", you can see that as a result of the outages, we have some imaging requests that were displaced by the outages and had to be rescheduled. For two of these the scheduler was able to find a new spot to schedule the imaging request; however for two other imaging requests, they had to be rejected as a new schedule slot could not be found to accommodate them. The full logs from which this snippet was derived can be found here.

2.5.7 Non-Conformance Disposition

2.5.7.1 Issue 1: "Slow-Performing Order Scheduling Algorithm"

> *The main problem I think will be making it so that you can actually test it. The longer it takes to run, the longer between successive runs to try and improve and iterate on your algorithms design. At some point as well you will want to test the algorithm with different sized datasets to ensure the consistency of its performance across varying dataset sizes but you don't want that to take a full day to run ideally.* —Patrick Irvin (CSA), Email exchange

During the first iterations of the scheduling algorithm, we identified a significant challenge when it comes to the execution time. The plotted data from early performance evaluations showed a quadratic increase in processing time as the number of orders increased. For a batch of 60 orders, the algorithm would take approximately three hours to complete. Although the output was accurate, this presents challenges in terms of testing efficiency since validating and verifying subsequent runs would require longer times, hindering future progress. Extrapolating further, the plotted data implies that for a load of 500 orders, which is what the stakeholders expected, the algorithm would take at least one day to finish. This presents a significant bottleneck in operations and could impede the system's ability to deliver timely results under expected load conditions. This issue is better explained through the following graph visual and the associated execution log for this algorithm run is documented in this text file.

2.5.7.2 Issue 2: "Imbalance Usage of Resources"

> *One thing that jumps out at me just looking at this is that there seems to be little balance between the station usage. Remember one of the objectives is to as much as possible utilize the ground stations equally. Right now it seems as if Inuvik is being used almost exclusively with Prince Albert and Gatineau only having a couple contacts.* —Nathaniel Cziranka-Crooks (CSA), Email exchange

In the first conformance reviews conducted by the CSA stakeholders, one very noticeable thing was there was very little balance between station usage. In the execution log, it was evident that Inuvik is being used almost exclusively with Prince Albert and Gatineau only

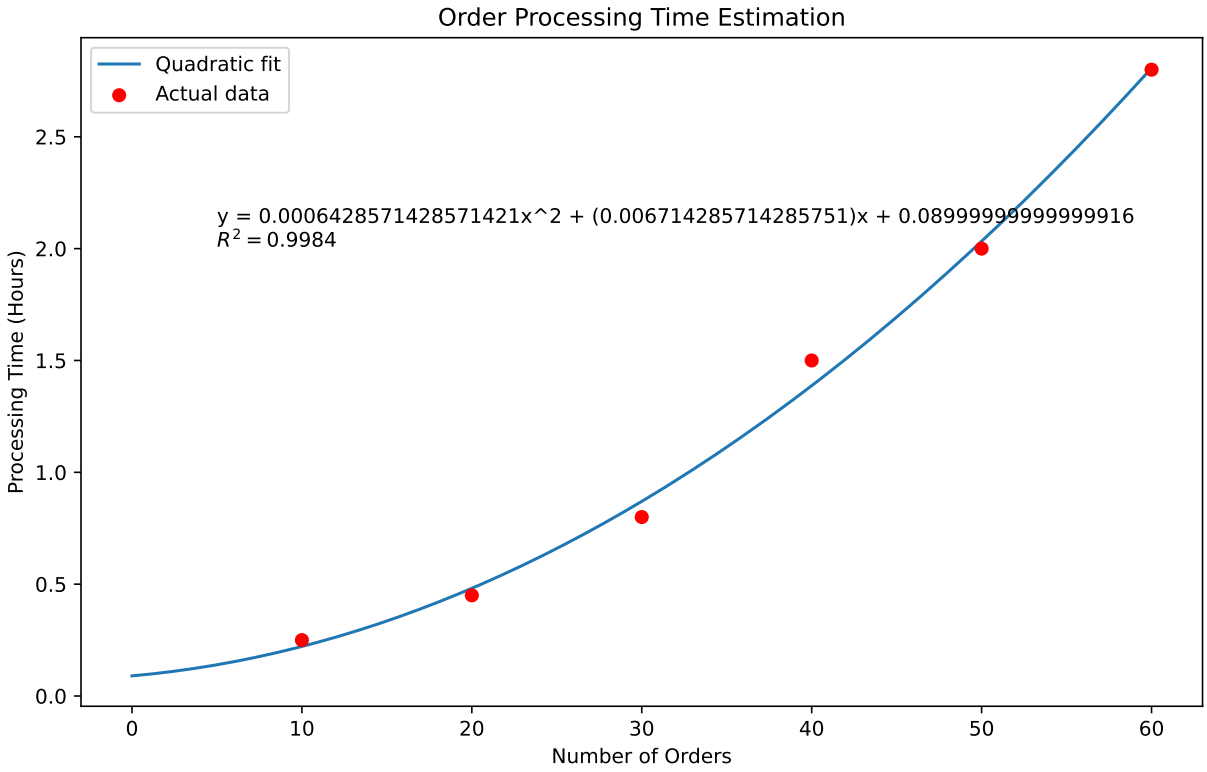


Figure 2.50: Quadratic Increase in Processing Time as a Function of Order Volume during the Algorithm's first iterations

having a couple of contacts. This is an issue because one of the objectives was to utilize the ground stations equally.

The link to this issue can be accessed through this text log by clicking here.

2.5.7.3 Issue 3: "Create Ground Station Attributes Not In Sync"

> *In EventRelayAPI > asset routes > create new ground station.*
> *The groundstationcreationrequest model's attribute names don't match the db names. Therefore the endpoint fails to function properly.*
> *Either the model needs to be fixed and front end axios request adjusted. OR create a conversion function that converts the gs creation request into a gs base model.* —Walid AlDari, Github Issue #4

On February 14th, 2024 within the Event-Relay-API, one of the endpoints responsible for creating ground stations was malfunctioning. The endpoint would not work as intended. When it was called to create a new ground station, the resulting behaviour was that it didn't actually create the ground station nor did it throw any errors.

The bug was detected by running the unit test relevant to the create-ground-station endpoint which seemed to fail when it would scan the database to check whether or not the change was reflected.

The link to this issue can be accessed by clicking here.

2.5.7.4 Issue 4: "Image Management Service Doesn't Use App Config"

> *Image management service in Event relay API needs to be updated to utilize App config properly.*
> *Whenever that service is run, an error is prompted stating "Base.class.ImageOrder is None".* —Walid AlDari, Github Issue #5

Whenever the Image Management service was run, an issue would be thrown upon it's compilation stating "Base.class.ImageOrder is None". The error message was odd given that it wasn't apparent a few days before when the service was running.

This bug was discovered coincidentally when attempting to run the image-management-service for the sake of testing other features.

The link to this issue can be accessed by clicking here.

2.5.7.5 Issue 5: "Image Management Service's Image Order is not in sync with DB"

> *The format of image order has changed and the changes are not reflected in the Image Management Service.*
> *The image order needs to be updated, and the conversion functions need to be adjusted accordingly.* —Walid AlDari, Github Issue #6

The image management service is responsible for converting the standard parsed image order to another less-nested format. The standard image order format contains a "Recurrence" attribute stating properties pertaining to repeating image orders. For the sake of

ease of access, it was much more effective to eliminate this nested attribute, and instead push up the nested properties so that the object is nest-less.

The unit tests pertaining to image-management-service were failing given that there was a slight change to the image order format at the time.

The link to this issue can be accessed by clicking here.

2.5.7.6 Issue 6: "Outbound schedule creation fail"

> *Outbound-schedule creation fails when any one of image-activities, maintenance-activites or downlink-activities is missing* —Ruth Bezabeh, Github Issue #11

Outbound schedule represents the satellite schedule to be sent to the ground station to later be uplinked to the satellite. It is created by aggregating all the orders that have been scheduled for a particular satellite. These could be any combination of image-activities, maintenance-activites or downlink-activities.

when the issue was discovered, the code creating outbound schedule expected all three types of activities and failed when any of them were missing.

The link to this issue can be accessed by clicking here.

2.5.7.7 Issue 7: "Eclipse Opportunity Finding: Function breaks with invalid inputs"

> *In scheduler service > ensure-eclipse-events-populated*
> *The function above breaks when fed 2 types of invalid input:*
> *-> The start time starts after the end time.*
> *-> The start time is equal to the end time.* —Walid AlDari, Github Issue #10

Within the scheduler service, the function that is vital within the scheduling workflow that is responsible for determining eclipse events during a time period was failing to function correctly in response to faulty input.

Discovered during the running of a unit test, the function would throw an error when the time window provided was invalid.

The link to this issue can be accessed by clicking here.

2.5.7.8 Issue 8: "Eclipse Opportunity Finding: Missing Eclipse Windows"

> *Upon testing the eclipse opportunity finding function in scheduler service. There are a few discrepancies to be found between the results outputted by the function, and the results being compared against (STK output)*
>
> *There seems to be a recurring issue where for every window outputted by the function, 2 windows are available in the stk results, but missing from the function results. For instance, say the function outputs 2 windows. The stk results would have 4 windows, 2 of the windows matching the function's windows with an error margin of at most 1 min. The other 2 windows would be between window 1 and window 2 of function's results.* —Walid AlDari, Github Issue #9

Another bug detected within eclipse opportunity function were a few discrepancies between the actual output and the expected output. There seems to be a recurring pattern of double missing windows in the actual output that were present in the expected results.

The expected results were retrieved from an aerospace program called STK, that provides TLE-related info.

The link to this issue can be accessed by clicking here.

2.5.8 Non-Conformance Resolution

2.5.8.1 Issue 1: "Slow-Performing Order Scheduling Algorithm"

The SatOpsMQ scheduling algorithm was optimized through several code enhancements. We integrated a binary heap structure to implement a priority queue data structure for dealing with order priority. Additionally, we improved loop efficiency by adding early exit conditions, such as terminating search operations when a matching slot was found to avoid needless iteration over the entire search space. We also applied a greedy algorithm for rapid asset allocation which estimated the optimal schedule without fully exploring every potential configuration. Moreover, the problem was reformulated based on the knapsack model and tackled using dynamic programming. This approach treated the full schedule as a combination of different sub-schedules, many of which are similar and redundant. The calculation results associated with these repeating sub-schedules were stored to avoid repeated computations. For quick retrieval of sub-schedule calculations, hash maps were utilized to achieve $O(1)$ access time. All of these optimizations have contributed to reducing the average case time complexity from approximately $O(n^2)$ to $O(\log n)$.

2.5.8.2 Issue 2: "Imbalance Usage of Resources"

The fix to this problem is to select the optimal communication slot for downlinking satellite data by considering the congestion levels of both ground stations and satellites. The code initially sets the best available slot to the first option and calculates the average congestion for all stations and satellites involved. Congestion is defined by the number of slots already allocated to a station and the number of events a satellite is scheduled for.

When choosing the best slot, the algorithm compares the congestion of each available slot against the average. It calculates a score that penalizes slots with congestion levels far from the average. The score is derived from the cube of the relative difference from the average congestion, which amplifies the impact of congestion as it moves away from the mean. The slot with the lowest score, implying the closest to average congestion, is considered the best option.

If there's no variation in congestion (i.e., the averages are zero), the algorithm falls back on comparing the raw congestion numbers, preferring slots with lower congestion. This approach aims to distribute the load more evenly across the network, thereby reducing the imbalanced usage of any single ground station or satellite.

The fix to this issue was to cater the algorithm towards a greedy approach by using our own-made performance metrics. The problem was treated as a knapsack problem as shown in this Jupyter Notebook and the full execution log is available in this text file.

2.5.8.3 Issue 3: "Create Ground Station Attributes Not In Sync"

This issue was fixed through first and foremost a team discussion over whether the DB schema should be changed to allow the microservice to function properly, or to adjust the microservice to adapt the new schema from the DB.

The consensus of that meeting was that any changes to the DB, require changes on the end of the microservices. Therefore the microservices utilising specific table schemas have to adapt accordingly. Therefore the implementation of the function was adjusted to account for the updated schema of the ground-station table.

The link to this issue can be accessed by clicking here.

2.5.8.4 Issue 4: "Image Management Service Doesn't Use App Config"

The source of the error was found to be a misalignment within team communicated. There was a change within the entire codebase for the purpose of repolishing the repository, which

consisted of taking out common folders within the microservices, and have the microservices refer to a single outer shared common folder. Therefore, certain import statements had to be adjusted.

An instance of an import statement adjustment was the error throw above, changing the import above to refer to the app-config folder solved the bug.

The link to this issue can be accessed by clicking here.

2.5.8.5 Issue 5: "Image Management Service's Image Order is not in sync with DB"

As mentioned earlier, the image management service is responsible for the conversion of object formats/schemas. Therefore, the image order schema was altered without notification, and resulted in the failure of the image-management service. Upon further communication with other teammates, it was found that the image order was modified heavily and that resulted in the implementation of the image-management service to be rendered obselete.

The adjustments within the image-management service required changing the expected property names within the implementation along with other minor changes.

The link to this issue can be accessed by clicking here.

2.5.8.6 Issue 6: "Outbound schedule creation fail"

It has since been fixed and it is now possible to have a schedule with any number of activities.

The link to this issue can be accessed by clicking here.

2.5.8.7 Issue 7: "Eclipse Opportunity Finding: Function breaks with invalid inputs"

The issue here was resolved by adding faulty input checks (if statements) to the function to specifically handle these edge cases.

The link to this issue can be accessed by clicking here.

2.5.8.8 Issue 8: "Eclipse Opportunity Finding: Missing Eclipse Windows"

It was found that this issue was a misconception. There was a change in time formats when date-time properties were saved to the database, in this instance, time windows were pushed by 2 hours, which in turn gave the false impression that the outputs were false. In reality, the function was properly functioning, however was inadvertently converting date-time formats. Upon communicating the issue with respective team members, the adjustment was made within the implementation and the issue was resolved.

The link to this issue can be accessed by clicking here.

2.6 Sustainability Impacts and Potential Unintended Consequences

2.6.1 Sustainability Impacts

As SatOpsMQ advances the capabilities of satellite operations management at the CSA, its sustainability impacts have been thoroughly evaluated to align with organizational goals and ethical standards:

1. **Energy-Efficient Computing -** using Amazon EC2, SatOpsMQ benefits from the scalable, on-demand cloud infrastructure that allows for efficient resource management. EC2's ability to dynamically scale computing resources based on real-time demands helps minimize idle computational power thus reducing unnecessary energy consumption typically associated with physical server maintenance.
2. **Resource Efficiency with Containerization -** docker containers significantly enhance the efficiency of deploying and running applications. By allowing multiple containers to share the same OS kernel, Docker reduces the overhead traditionally associated with virtual machines, thereby optimizing resource use and enhancing energy efficiency. By reducing the computational overhead, Docker containers contribute to lower energy usage which is crucial for minimizing the environmental impact of technology operations. The energy savings are particularly significant in environments with large-scale deployments, where even minor efficiencies can lead to substantial reductions in energy consumption.
3. **Optimal Resource Utilization -** the system employs a sophisticated algorithm to manage task scheduling and resource allocation efficiently. This ensures that

bandwidth and computational resources are utilized optimally which reduces the strain on infrastructure and prolongs the effective life of hardware resources.

4. **Reduced Hardware Turnover -** by maximizing the efficiency of existing hardware through cloud computing and containerization, SatOpsMQ decreases the frequency of hardware upgrades needed. This not only lowers the cost associated with hardware procurement but also reduces electronic waste.
5. **Open-Source Contribution -** Being open-source, SatOpsMQ encourages a community-driven approach to software development. This model fosters continuous improvement and maintenance which allows for incremental updates and bug fixes without the need for complete overhauls. Such practice extends the software lifecycle and reduces waste associated with disposing of outdated software.
6. **Modular Design -** the modular design of SatOpsMQ facilitates easier updates and scalability. Modules can be updated independently of one another which minimizes disruptions and reduces the resources required for updates and maintenance.

2.6.2 Potential Unintended Consequences

1. **Risk of Data Inaccuracies -** due to its critical role in managing satellite operations, inaccuracies in SatOpsMQ's data processing could lead to operational disruptions or erroneous task execution. These inaccuracies could stem from bugs in the system, faulty integration with other data systems, or incorrect data input by users. To mitigate inaccuracies in SatOpsMQ's data processing, the system should incorporate rigorous validation and verification protocols at all data entry points. Regular audits and updates of integration protocols are necessary to ensure accuracy and seamless compatibility between systems. Additionally, the deployment of automated error-detection software, supplemented by manual oversight, will help in continuously monitoring and promptly rectifying data processing anomalies.
2. **Compromised System Overrides -** while the ability for operators to override automated processes is important, there is a risk that such overrides could be misused or triggered inappropriately due to human error or misunderstanding which may potentially lead to suboptimal operational decisions. To prevent inappropriate use of system overrides, stringent operational protocols need to be established, clearly defining the criteria under which manual overrides are allowed. Operators should receive extensive training on the implications and proper procedures for executing overrides. Furthermore, implementing layered authentication mechanisms will ensure that only authorized personnel can perform overrides, with all such activities logged for subsequent review.

3. **Over-reliance on Automation -** the extensive automation within SatOpsMQ might lead to a skills decay in manual operations among users, reducing their ability to intervene effectively in emergencies or when automation fails. Mitigation of over-reliance on automation involves regularly scheduled manual operation drills to maintain human operator proficiency. A comprehensive training program that includes scenario-based training on automated system failures is essential. Operators should also be encouraged to pursue ongoing education and re-skilling to stay adept at managing new technologies and system updates.
4. **Scalability Bottlenecks -** despite its design for scalability, rapid expansion or unexpected surges in data volume could overwhelm the system, leading to slowdowns or failures, particularly if the scaling of infrastructure does not keep pace with operational demands. SatOpsMQ was only tested for 500 simultaneous orders, and the situation in real life the load could be much larger.To address scalability bottlenecks, it is crucial to continuously monitor system performance and preemptively plan infrastructure scalability. The software architecture should be modular to facilitate efficient scaling, especially during peak times. Conducting regular stress tests will help understand the system's behavior under high loads and lead to necessary optimizations to accommodate unexpected surges.
5. **Privacy Concerns -** handling sensitive information, especially in an open-source environment, raises significant privacy issues. Unauthorized access to or leakage of operational data could have severe privacy implications. Protecting privacy in an open-source environment involves employing advanced encryption and cybersecurity measures to secure data both in transit and at rest. Strict access controls and audit trails should be implemented to monitor data access, ensuring accountability and compliance with international data protection regulations to safeguard sensitive information.
6. **Impact on Employment -** the automation capabilities of SatOpsMQ might lead to a reduced need for human operators, potentially leading to job reductions or shifts in role requirements which could have socioeconomic impacts on the workforce. The potential socioeconomic impacts of automation within SatOpsMQ can be mitigated by collaborating with human resources to forecast job shifts and prepare transition plans. Providing training and support for workforce adaptation to new roles or technologies will help mitigate job reductions and ease the transition for affected employees.

2.7 Baseline Design Analysis & Formulation

2.7.1 Stakeholder Analysis

The development of the project is significantly shaped by the distinct needs and inputs of its primary stakeholders. The Canadian Space Agency (CSA), as the intended primary user, has a substantial influence on feature prioritization. Their operational requirements directly steer the functionality focus in the MVP, ensuring the web-app aligns with their needs for operations and intellectual property. Similarly, the Nanosatellite Research Lab, led by the team's supervisor, provides a steady stream of academic and scientific insights. The lab's contribution, particularly in the form of research papers, plays a crucial role in guiding the backend development of the web-app, ensuring it stays abreast of the latest advancements in nanosatellite technology.

In parallel, the ENG4000 Capstone teaching team's input on project management and usability is instrumental in shaping the web-app's design. Their expertise ensures that the app is user-friendly and adaptable for educational purposes, incorporating elements that facilitate teaching satellite operations and project management. This makes the web-app a valuable tool for instructional settings.

Furthermore, anticipating the potential use by scientific researchers, the design of the web-app is geared towards versatility. This foresight is essential to accommodate diverse research needs, ranging from specific data analysis to satellite communication validation. The adaptability in design ensures that the web-app can meet various research requirements, making it a robust and flexible platform for future scientific exploration.

The stakeholder matrix, which follows this section, details each stakeholder's specific impact and influence on the project, guiding the prioritization of design features. This matrix is instrumental in ensuring that the web-app not only meets operational standards but also addresses the varied needs of its diverse user base. This stakeholder-centric approach ensures the web-app's design is well-aligned with the practical and strategic requirements of all parties involved.

2.7.2 Feasibility Analysis

Embracing universal design principles is at the heart of the solution. This emphasis on inclusivity ensures that the system is accessible and usable by a broad spectrum of users, thereby adhering to global standards and fostering a more extensive user base.

Stakeholder	Impact Level	Influence	Needs	Level of Support
Canadian Space Agency	High	High	Manage closely	Validation of process and methods proposed for development
Nanosatellite Research Group	High	Medium	Keep satisfied	Keeps track of team's progress through the web-app design development
ENG4000 Teaching Team	Medium	Low	Keep informed	Bring space engineering expertise into project where applicable
Scientific Researchers, Space Industry, etc.)	Low	Low	Monitor	Continuous testing and improvement of the product after deployment

Table 2.18: Stakeholder matrix for SatOpsMQ

The web app plays a pivotal role in advancing sustainable satellite operations. By optimizing resource usage, the software indirectly champions cause like fuel conservation and reduced space debris. These efficiencies not only translate to economic savings but also align with larger environmental preservation goals.

To ensure the project's alignment with global sustainable development goals and to sidestep potential pitfalls, conducting a meticulous study on its environmental and societal impacts is imperative.

The software, in its current state, can grapple with a variety of optimization criteria. However, it faces constraints when it comes to real-time processing. This limitation might occasionally hinder the software from achieving the best possible outcomes in certain scenarios.

The current design's primary focus rests on the software realm. As a result, when it comes to integrating with satellite hardware or tailoring it for custom-built ground stations, there might be additional considerations and challenges that need addressing.

In terms of the feasibility of our proposed solution and how it fares compared to competitor products, we look at seven parameters that evaluated how successful the proposed

solution would be [20]:

1. **Define the Opportunity:** The problem that the proposed solution is attempting to solve is the complex and challenging methods currently used manually by satellite operators in the CSA for scheduling and managing imaging, maintenance, and outage requests. Our proposed solution is to develop an open-source web-app that automates the manual process for scheduling and downlinking images, maintenance, and outage requests with dynamic operational constraints based on the CSA operator's preferences.
2. **Define the Objective & Scope:** The primary objective of the open-source web-app is to develop a scheduling algorithm that can be used to automate the downlinking and uplinking of image, maintenance, and outage requests. The scope of the project is focused on sample assets given to us by the CSA such as five satellites and three ground stations that will be used in the web-app.
3. **Conduct Market & User Research:** Currently in the market, other applications can be used for various user needs such as the USGS Earth Explorer, NASA Operations Software, and AGI's STK as proprietary and simulation software solutions that can be used to generate schedules for possible downlinking and uplinking of satellites.
4. Considering that the main users of our web-app will be the CSA operators who are sponsoring the project, we actively meet with them on a weekly basis to affirm the progress made on the web-app and any changes that need to be made to satisfy the user's needs. Moreover, due to the open-source nature of the project, there is a GitHub project that is publicly available that can take any external user feedback and be addressed by the team.
5. **Analyze Technical Feasibility:** The main aspect of technical feasibility for the product is to examine the elements that are needed for the product to generate schedules. Similarly to other competitor products, the open-source web-app will utilize an event-driven system that can use a general scheduling algorithm with the following objectives:
   - Use assets (Satellites & Ground Stations) as much as possible to fulfil orders.
   - Load balancing between assets and orders sent out (maintenance and outage order consideration).
   - Fulfil as many orders as possible (image, maintenance, and outage orders).

   The method in which we will map our assets is through using a dockerized system that can be run on any local computer for the user and other code that will be used to create simulations of the satellite and ground stations and not real hardware compared to proprietary solutions on the market.

6. **Assess Financial Viability:** In terms of financial viability, products on the market have various elements that make their solutions costly such as hosting the web-app, designing the product for the public, and licensing certain satellites/assets for proprietary solutions which could be extremely costly since some companies would want to outsource the work of the development for a platform to display the data gathered by their assets. Other simulation software could also be costly such as AGI's STK where if all features were utilized for the user's assets could cost $200,000 for one user to use the application locally [21]. With our proposed solution, it would be easy for any company to import their assets into our system and demonstrate simulations of the data their assets would gather autonomously with little to no cost. Companies outside of our stakeholders would then have to evaluate if the work done in-house would be more cost-savings in terms of time and money compared to out-sourcing the work.
7. **Analyze Economic Feasibility** Economically, the open-source nature of the web-app presents a cost-effective solution compared to market alternatives. Licensing costs and the expense of proprietary software make competitor products significantly more costly. By contrast, our solution offers an economical alternative that allows for easy integration of user assets and provides autonomous simulation capabilities with minimal financial outlay. The economic assessment considers the potential for cost savings for companies that choose to use the in-house solution over outsourcing.
8. **Analyze Legal Feasibility** Legal feasibility concerns are addressed by adhering to open-source licensing regulations, ensuring that all code and technologies used in the project comply with these standards. The project's commitment to universal design principles also aligns with global standards, promoting broader accessibility and inclusivity. All legal aspects of software development, including copyright, patents, and data privacy, are thoroughly reviewed to safeguard stakeholders.
9. **Analyze Operational Feasibility** From an operational standpoint, the solution's design is reviewed to ensure alignment with the satellite operations' sustainability goals. The focus on optimizing resource usage demonstrates the project's commitment to fuel conservation and reducing space debris, supporting long-term operational sustainability. Regular stakeholder engagement through weekly meetings and public feedback via GitHub ensures that the project remains aligned with user needs and operational objectives.
10. **Analyze Preliminary Design Assessment** The preliminary design review ensures that all concepts are scrutinized for technical viability. It involves assessing the appropriateness of materials, the sufficiency of resources, and the robustness of the selected technologies. This phase serves to establish a strong foundation for the project by confirming the feasibility of the proposed design solutions, thereby

paving the way for successful development and deployment.

2.8 Evaluation of Technical Pack Performance

The teams understanding of this section is that it evaluates the performance of each of the other components for the technical pack:

- Executive Summary
- User Requirements
- Systems Needs or Requirements
- As-built Design
- Engineering Budgets
- Non-conformances from test procedures and their current desposition.

These components and their associated rubric rows will be self-evaluated in the following two tables. This matches the rubric in section 2.5 of the Final Year Project Progress, Mid-Project and Final Project Deliverables Document. The tables below state the rankings we feel we have earned and the references report sections that detail the fulfilled obligation(s) as per the rubric. As well, all evaluations are available in Appendix D.2 Self Evaluation

Criterion	Ranking	Justification
Articulate the Design Problem To Be Solved	Exceeding	Reference1, Reference2
Feasibility of Proposed Approach	Exceeding	Reference1
Formulate a Strategy for Designing an Engineering Solution	Exceeding	Reference1
Decompose a Complex System into Subsystems	Exceeding	Reference1
Criteria to Compare & Contrast	Exceeding	Reference1, Reference2,
Justify Strengths & Limitations, Make Recommendations	Meeting	Reference1, Reference2
Apply Engineering, Fundamentals, Theories, Practices	Exceeding	Reference1
Develop Creative, Innovative Solution	Exceeding	Reference1, Reference2, Reference3,
Review Project Sustainability, Impact, Unintended Consequences	Exceeding	Reference1

Table 2.19: Self-Evaluation General Section: Engineering Design.

Criterion	Ranking	Justification
Ensure State Appropriate for Formal Testing	Exceeding	Reference1
Develop Test Plan for System Verification/Validation	Meeting	Reference1
Develop Test Procedures and Ensure Test Equipment Understood	Meeting	Reference1, Reference2, Reference3
Monitor and Address Deviations From the Test Plan	Meeting	Reference1
Ensure System Configuration, Failure Tracking Defined and Implemented	Exceeding	Reference1
Review System Test Results for Validity	Exceeding	Reference1, Reference2, Reference3
Document and Track Test Failures Appropriately	Exceeding	Reference1, Reference2

Table 2.20: Self-Evaluation General Section: Test Related Criteria

Criterion	Ranking	Justification
Identify Driving Requirements	Exceeding	Reference1, Reference2
Understand Rationale for Requirements	Exceeding	Reference1, Reference1, Reference2
Analyze Requirements Wording	Meeting	Reference1, Reference1
Analyze Requirements Scoping	Meeting	Reference1, Reference2, Reference3
Analyze Requirements Compliance	Exceeding	Reference1

Table 2.21: Self-Evaluation For a Well-Defined Project Section

Chapter 3

Project Management & Financial Volume

In this volume, we discuss the project management framework adopted by the SatOpsMQ team as well we discuss finances and the preliminary business case for the project. The project has followed Agile Methodology throughout its life cycle, however the precise execution of this as a project management framework has been evolved over the course of time. First, it is key to describe exactly what is meant by Agile Methodology in the context of software development. Agile Methodology started with the Manifesto for Agile Software Development written in 2001 by several prominent software engineers. It is a set of twelve principles that emphasize collaboration with customers, working software, stakeholders, and rapid response to changes [22]. Agile has evolved much since its inception but these core principles are still present. The key element of Agile that the SatOpsMQ team emphasized was iterative development in short two week stages, which are called sprints. As well the project had some strict requirements that did not really change, but there are several requirements that changed and grew as the system was incremented upon with the customer's (the CSA) collaboration. Flexibility in the face of these changes was always a must up and until the final release of the project.

After providing an overview of the project management details, the release schedule and sprint schedule of the as-built SatOpsMQ application will be described in detail. The sprints and releases will be summarized and as well the full list of tasks in each sprint will be described. Next, resource allocation of tasks will be discussed as well as the challenges faced doing this. This will be followed by financial aspects including the project's equipment and procurement. To conclude this volume a business case for the system is presented.

3.1 Project Management Overview

3.1.1 Updates & Rationale

In our previous reports, we had presented strict requirements for the application that were not representative of an agile project management process. This was an oversight and has been corrected. Instead, the team decided that it is best to demonstrate the requirements through our systems features road map which shows the changes that have been made throughout the project life-cycle and the delivery of these features to our users/stakeholders. This road map is present as sequence of releases and associated sprints that build towards these releases.

The previous requirements that were in the Fall progress report can be found in Appendix B.4 for reference if needed to compare them to our current agile roadmap.

The approach taken to project management has evolved since the project kickoff in September, 2023. In the first half of this project (September, 2023 to December, 2023) the management followed a very basic version of the Agile sub-framework called Scrum as described by Atlassian, the provider of the team's project management tool called Jira [23]. It is important to note the distinction that Agile is an overall philosophy for project management while Scrum is the specific Agile framework that was used in this project. Moreover, the aspects of Scrum used in this project are custom tailored for the team's purposes as is the recommendation by Atlassian for using Scrum project management [23].

To start with, Scrum project management is well suited for a small software development team such as that for SatOpsMQ (team of seven people). In a Scrum team there are usually three types of roles: a Scrum master, a product owner and developers. The former two roles were not set in stone and followed super strict. They were fluid roles due to the project requiring everyone to be a developer. The Scrum master is responsible for leading the team and facilitating their activities in regular meetings. For this project, Rafael Dolores was the primary Scrum master due to his previous experience with Agile/Scrum, however, other team members took on this role as needed. Next, the product owner is responsible for understanding the overall user and system needs for the system under development and prioritizing tasks to be performed. This role was shared among all team members due to the small team size. Finally, the developers are responsible for the execution of tasks and doing granular planning of tasks during sprints [23]. All seven members in this team were primarily the developers of the product.

3.1.2 Key Elements of Scrum

In the context of Scrum the word sprint has come to mean a short iteration of work that results in the generation of customer value for the system under development. The phases of work in this project were conducted as sprints that lasted two to three weeks on average. As well, the sprints were grouped by four major product releases: The minimum viable product (MVP), alpha release, beta release, and final release. Each sprint was purposed with adding value for SatOpsMQ users as well as working towards the upcoming release.

There are three main Scrum "artifacts" that were used to guide the development of the product and all it's associated tasks. These are the product backlog, sprint backlog and sprint goal. The product backlog is an evolving list of tasks that are to be completed over the course of a project. This list is continuously updated and reviewed by the team when assuming the role of product owner. This list includes items that are perceived to be important for upcoming releases. The sprint backlog is the list of tasks to be done within the time frame of a specific sprint. These items are transferred from the product backlog when planning an upcoming sprint. Lastly, the sprint goal is the ideal outcome for each sprint [23]. The sprint goals are important artifacts to show stakeholders exactly how value has been incremented into the product. Moreover, sprint goals help towards release planning and sprint review.

The major activities of Scrum that tie this all together are: backlog grooming, sprint planning, stand ups, sprint review, sprint retrospective and of course the actual sprints. The purpose of backlog grooming is to ensure there are key items present in the product backlog so that the product vision is in line with the work developers are doing. This activity is the responsibility of the product owner and it is best accomplished by looking at industry trends for similar products, doing market research and collaborating with stakeholders/customers. Sprint planning is the selection of tasks to be performed in the current sprint. It is a meeting of all team members that is facilitated by the Scrum master. The sprint goal is determined in this phase. As well, the work items' completion times should be estimated and the items should be divided such that each member has a feasible allocation of work. The result is a team that has a clear road-map of the work to do over the sprint as well as reasonable estimates of time to completion. It is important that time to completion is estimated collaboratively and realistically [23].

Once a sprint has started it is common to have daily stand ups, which are short team meetings conducted while literally standing up. The purpose of standing up is to ensure the meetings stay short as team members are expected to get tired of standing up and are always ready to leave. Stand ups for the SatOpsMQ team were done weekly instead of daily due to the nature of the project being part-time. In each stand up, team members

are expected to provide updates on what has been done, what is left to do and what is troubling, all in the context of the current sprint. Stand ups are a key aspect to help keep the sprint goals in line with the product vision and user needs. Sprint reviews are conducted after a sprint concludes to demonstrate deliverables added to the product from the most recent sprint. The SatOpsMQ team conducted these reviews internally and with stakeholders present if available. Finally, sprint retrospectives are reflection phases on previously completes sprints/project releases. These serve to allow team members and stakeholders to bring up concerns and identify areas of improvement [23]. These were completed before each release in the project.

Now that these important terms are defined in the context of the SatOpsMQ project we can dive deeper into the categorization of work items, which are almost always called "user stories" in Scrum.

3.1.3 User Stories

In Agile Methodology (and Scrum) all units of work are described as user stories, which are atomic units of work that developers are responsible for. A user story is a straightforward description of a product deliverable from the point of view of the end users of a system [24]. There are three classes of users in SatOpsMQ: satellite operators, imaging customers and developers. Satellite operators use the system and interact with the frontend elements to perform day to day tasks. Imaging customers specify and send orders to be scheduled. Developers are responsible for integrating the system into their organizations workflow. As well, developers are future contributors to this project due to its open-source nature.

Agile user stories are measured with story points, which are estimators of effort required to complete a story. Story points are estimated during sprint planning. Estimation for the SatOpsMQ team was done using Scrum Poker, which is a collaborative exercise to vote on how long a particular story is expected to take [25]. This was done using the ScrumPoker-online tool. A comprehensive breakdown of resource allocation by team member will be discussed in Section 3.3.

In the context of a large scale project, not all user stories are equal in terms of effort and size. For this reason, there are several other terms that get used in Agile to create hierarchies and groupings for user stories. There is no single accepted hierarchy but there are several common terms (along with user stories) such as features, epics, themes, initiatives, and sagas [26, 27, 28]. This can get confusing and project managers advise that these should be used sparingly [28]. A complicated hierarchy should not be a goal and it should be kept as simple as possible. In the context of SatOpsMQ, sagas are not used. It

is important to first define what these terms mean internally and then show the hierarchy used. The term feature we define as a story that is substantial enough to be released on its own [28]. For example, the 3D/2D maps for satellite and ground station visualization was one story but large enough to be released as a full fledged feature. Epics are large user stories that are not subdivided into atomic user stories. An epic can be divided further but it may not be for two reasons. First, it may be immature in the backlog and the subdivisions have not been made yet. Second, it may be intentionally kept as an epic so that a large unit of value is added all together, perhaps to release as a feature (as mentioned before) [28]. A theme is a grouping of stories/epics by some similarity metric, much the same as the themes in fiction novels in a library [28]. Finally, an initiative is a long term goal at the highest level of abstraction [27, 26]. This arrangement can be visualized in the SatOpsMQ User Story Hierarchy Diagram.

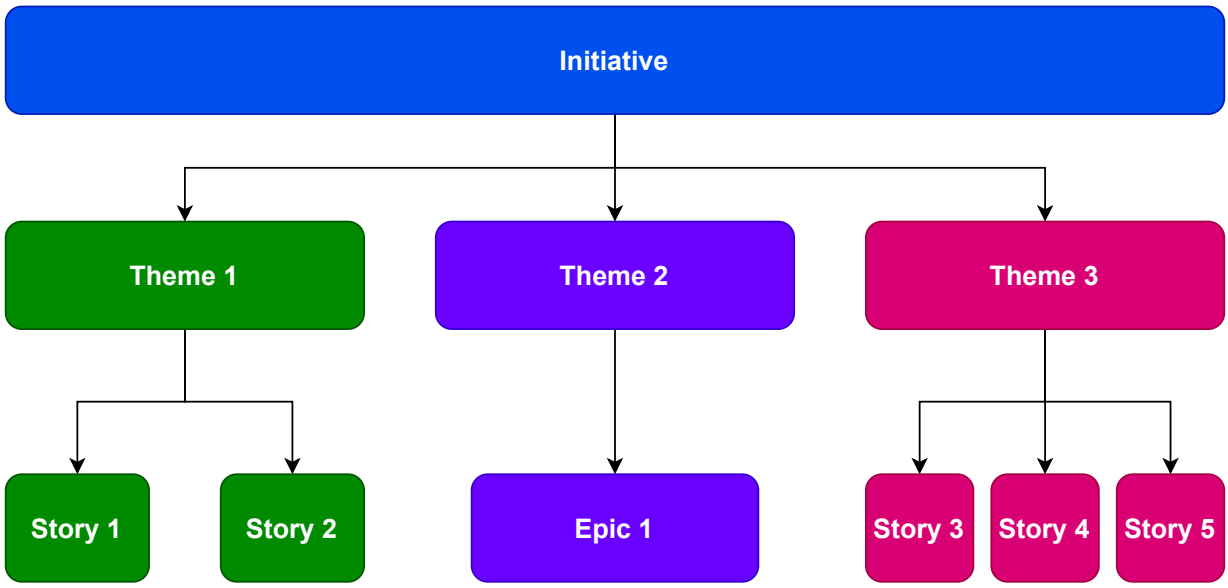


Figure 3.1: SatOpsMQ User Story Hierarchy

In the course of development of SatOpsMQ the organization and classification of initiatives, themes, epics and stories has been continuously reviewed and reworked. In the early stages, this story hierarchy was not as matured or well ordered. As the project progressed, key stakeholders involved recommended that the team formally outline the project management approach used. A significant portion of time was spent addressing this, leading to the groupings that follow. We now describe the various initiatives and themes in the project. These serve as tags attached to individual user stories and epics to ensure that

the overall project vision is on a unified track.

3.1.3.1 Initiatives

Since the inception of SatOpsMQ, the project has had three major initiatives and these have been relatively stable aside from improvements in name. The three initiatives are:

1. Development of an open-source backend scheduling server that can handle 500 image requests per day.
2. Development of an open-source frontend desktop client to be used by satellite operators.
3. Development of an open-source DevOps pipeline, free from vendor lock in, to facilitate backend and frontend devlopment.

These are the initiatives that have served as the three major divisions for all themes and stories present in the project plan. Next we present the major themes that group the subsequent user stories and epics. The themes' associated initiative is also given.

3.1.3.2 Themes

The major themes in the SatOpsMQ project are grouped based on the specific aspect of the system the themes' stories and epics relate to. The themes and their parent initiative are:

1. Development of an open-source backend scheduling server that can handle 500 image requests per day.
   (a) Image order management module to support backend inputs for use by the scheduling module.
   (b) Satellite activities tracking module to facilitate scheduling and allow data visualization.
   (c) Scheduling algorithm for heuristic creation of image, maintenance and outage schedules that balance usage across all assets.
   (d) Scheduling module to facilitate orders, asset activities and scheduler algorithm.
   (e) Event relay API that serves as a bridge between the client and all backend services.
2. Development of an open-source frontend desktop client to be used by satellite operators.

(a) Asset status view to show operators current data about satellites and ground stations.
(b) Image, maintenance and outage order activity view to show operators all orders.
(c) Manual input of maintenance and outage orders by operators.
(d) Satellite orbit and ground station visualization models for operator display.

3. Development of an open-source DevOps pipeline free from vendor lock in.
   (a) Containerization of all system modules for vendor free deployment capability.
   (b) System testing across unit, integration and end-to-end scopes.

With these general initiatives and themes now declared, we can move forward to discuss individual user stories and epics along with the theme which they belong to. These are discussed in the next section in relation to the project release schedule and sprint schedule.

3.2 As-Built Project Schedule

To guide the releases and sprints as well as to visualize the initiatives and themes, we use a now-then-later chart as a guide. This serves as the Agile Roadmap. This chart has evolved over the months the project has progressed but almost all items except a login system have been realized. As well some items have been added since this was first made. This chart serves as a heuristic for guiding releases and sprints to ensure they line up with the initiatives and themes. As well the points outlined in this chart are guided by the product vision board outlining user/system needs. "Now" focuses on items built in the early stages up and until the first release (minimum viable product). "Then" focuses on improvements made to this release in the course of the alpha and beta releases. "Later" focuses on adding the higher order features after getting feedback from testing and stakeholders. We now move to present the summary of the releases.

3.2.1 Summary of Releases

The SatOpsMQ project is divided into four release phases. Starting off with the MVP containing our bare bones implementation of the system, moving onto the Alpha release with our initial version of the scheduling algorithm, then the beta release where testing procedures are applied, and the Final release containing our final product satisfying all ready for deployment.

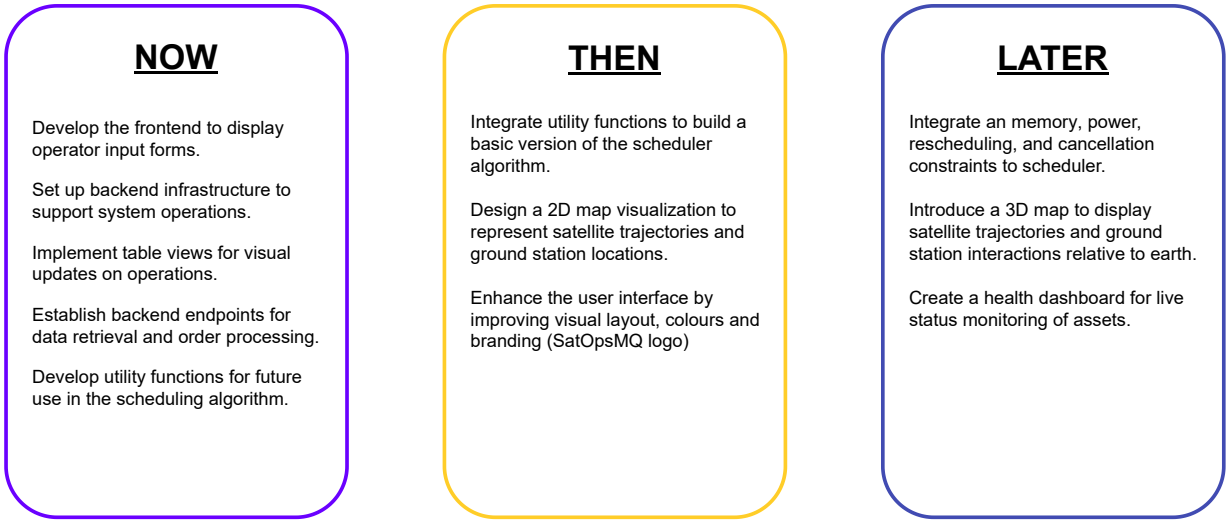


Figure 3.2: Now-Then-Later chart

The phases and their details are condensed in the Launch Plan below. Moreover, for each of the phases, we confirmed the progress of the web-application's features and functionality with our supervisor and the stakeholders/customers from the CSA.

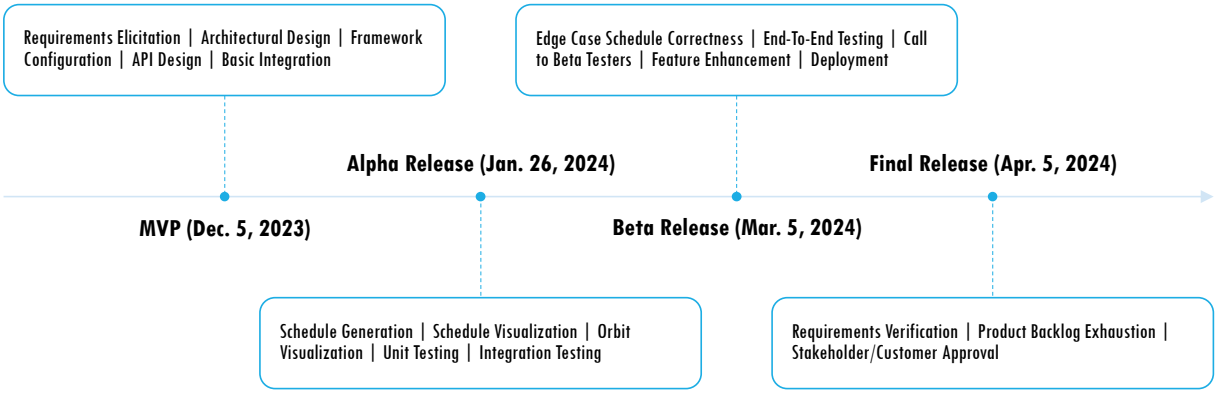


Figure 3.3: Launch/Roll-Out Plan

3.2.2 Minimum Viable Product

The MVP contains a very basic skeleton of the system. The system at this point is not fully functional, nor are the components properly interconnected and verified. However, what is established is the basic required database schema needed to begin development. As well as the back end architecture for streamlining development. Moreover, the technology stack has been finalized and fully conifugred in the code base.

The MVP has a very basic front end interface, consisting of a few pages each displaying a table of sample data relevant to it's category, and a few pages displaying forms to input image orders or outage orders.

The scheduling algorithm has not yet been worked on at this stage, as we were still configuring the system to prepare to determine what parameters and objects the algorithm needs to take into consideration.

The basic APIs responsible for adjusting and populating the database have been setup and organized into their respective microservices. These APIs formed the baseline of our system to prepare for algorithm development and testing.

Stakeholder feedback indicated that the development team is on the correct path but further improvements are necessary. The CSA recommended enhancing the user interface as they noted that its current style appears overly cartoony. For the assets page, they advised incorporating a visualization feature to depict live satellites in relation to ground stations to aid operators in making informed decisions regarding order overrides. Additionally, they emphasized the importance of further refining the algorithm to ensure continued progress.

3.2.3 Alpha Release

At this stage, a major wave of features were implemented ranging from backend improvements and polishing, to scheduling algorithm development, and finally the redesign of the front end UI.

The backend system was re-polished by cleaning out unnecessary junk files and folders to allow for more effective and clean development. The previous state of the backend contained many redundant files/folders coupled with unused microservices which made the overall environment difficult to work with because of the clutter. This was all cleaned out by this release.

Development of the scheduling algorithm had begun and was halfway through completion. The algorithm was capable of producing schedules given a set of image orders. These schedules were also visually displayed to allow for a quick walk-through of the results. The scheduler however had a few shortcomings such as not properly balancing the load of orders across satellites, it had a high tendency to reject orders that were possible to fulfill. Finally, the ability to readjust the schedule when adding a new image order was not functional yet.

Unit and integration testing had begun in this release as well. Comprehensive tests covering all the non-scheduler aspects had been created. Python unit tests and integration tests were applied on the APIs of the system. Any endpoint that connected with the database, or performed specific computations that are used later on by the scheduler were covered. Thereby ensuring that the bare bones functionality of the system was intact and functioning correctly.

The frontend UI had a major rework based on stakeholders request, making it much more visually pleasing and loading it with more features. The UI now included 2D and 3d globe visualizations of the locations of the satellites and the respective ground stations. As well, the schedules outputted by the algorithm are viewable through a Gantt chart.

Stakeholder feedback in this phase was limited; CSA noted that the algorithm's performance was slower than expected in real-world conditions. They placed high priority on enhancing this aspect for the beta release. This feedback was thoroughly documented in this non-conformance section.

3.2.4 Beta Release

At this stage, the focus was on adding more comprehensive tests to the various components of the system including the scheduler. More refinements to the scheduling algorithm were applied to do with improving the execution speed, E2E testing had begun, and system deployment options were considered.

Issues raised in the previous release regarding the scheduler algorithm were being tackled. The issue of load balancing across satellites was handled through specific implementation optimizations, and the ability to readjust a schedule after its formation when adding a new image order was implemented. All these improvements aimed to maximize the capabilities of the algorithm and further automate the process of scheduling to alleviate the load off of a satellite operator. Finally, end-to-end testing had begun to ensure the functioning of the overall system.

In this phase, stakeholders provided specific feedback. They highlighted the need for more comprehensive end-to-end testing, a point that Regina, our supervisor, also stressed. Additionally, it was noted that there was an imbalance in asset usage, with one ground station dominating others. This issue of unequal asset utilization has been detailed extensively in the non-conformance section.

3.2.5 Final Release

The final release involved concluding the End to end testing performed in the previous release and addressing any issues that had risen from that. This release focused on integrating any disjoint components such as the frontend and backend. Further improvements to the UI were made such as the addition of the health dashboard which provides the satellite operator with a glimpse of the overall status of the system. A majority of the algorithm work was dedicated to improving the asset usage which was highly emphasized by the CSA stakeholders from the previous release. This feedback gave motivation to incorporate the greedy algorithm into the code

Finally, this release involved validating that the system had fulfilled all user needs brought forth by the sponsor/customer, as well as showcasing the final product to the respective parties for final feedback.

Feedback for the final release indicated that CSA stakeholders were pleased with the improvements to the algorithm's execution logs. CSA stakeholder Nathaniel commented, "This looks much better in terms of balance!!" Due to the complexity of these refinements, considerable effort was dedicated to assisting the CSA with understanding the failure diagnostics of the logs. This improvement was thoroughly documented in this testing section.

3.2.6 Release Criteria

The release criteria describes the parameters that the team needs to look for to ensure that the web-app is ready for alpha, beta, and final release. The overarching parameters within these release windows are common but vary in their goals as the team progresses through each release.

If any of the five parameters outlined for each release is either met or not met, they will be marked with an [X] to mention that they have been met, if they have not been met, then they will be marked with [O].

For the alpha release, we had the following parameters to look out for:

1. **Functionality:** For the alpha release, the mandatory functions that the web-app should perform is to demonstrate that it can provide visualizations for the satellite and ground station communications and a bedrock for the different web-pages that would be used for future releases. [X]
2. **Usability:** For the alpha release, the users of the web-app should be able to navigate through the web-pages and understand what each web-page represents intuitively. The team has confirmed this with our CSA stakeholder. [X]
3. **Reliability:** The system for the web-app in terms of reliability only focuses on scheduling imaging, maintenance, and outage orders. So far, the team has developed a deterministic algorithm that can schedule all types of orders as specified by the user. [X]
4. **Performance:** The response time of scheduling orders is fast on the machines of the developers as well as deploying the web-app and navigating doesn't take more than 2 seconds to load any web-page. We still need to test the web-app on the machines of the users/CSA stakeholder but can assume that based on the specifications given from the CSA and the specifications of the developer team that we meet this requirement for the alpha release system. [X]
5. **Support-ability:** In terms of supporting various features in the web-app and having it be testable, configurable, and serviceable to the user. The alpha release system isn't completely serviceable to the user because certain features aren't fully integrated or completed. [O]

The beta release covered more in-depth detail on the functionality & support-ability:

1. **Functionality:** For beta release, the mandatory functions the team wanted to focus on were functionality for scheduling orders and fully integrated features between the backend and frontend. [X]
2. **Usability:** For beta release, the team added more documentation to make the user interface for the web-app more intuitive. [X]
3. **Reliability:** For the beta release, the team focused on making the schedules more optimal for newer user needs such as perturbations of scheduling or having to re-schedule at specific windows. [X]
4. **Performance:** The response time of the system with the beta release has yet to be tested with the new features added and integrated. [O]
5. **Support-ability:** All features have been integrated and the web-app is fully configurable, scalable, and serviceable to the user through the GitHub project with instructions outlining how to launch the web-app. [X]

For the final release, the team wanted to make sure that all the necessary features

required by the user were available, tested, and verified:

1. **Functionality:** For final release, the team wanted to focus on polishing and improving the existing features that were already built from prior releases. [X]
2. **Usability:** For the final release, we wanted to ensure that the front-end user interface was fully operational and connected fully to the back-end servers and other algorithms for the Scheduler. [X]
3. **Reliability:** For the final release, the team tried to focus more on the performance of the web-application as a desktop app and performance through full operation of the web-application. [X]
4. **Performance:** Web-application was tested to see if all features could work synchronously together for the user and fulfil all the aforementioned user needs. [X]
5. **Support-ability:** Similarly to the beta release, all features of the web-app and its progress are available on the team's GitHub Repositories as well as all the necessary documentation needed for the user to extract the codebase based on their needs. [X]

3.2.7 Individual Sprint Backlogs

User stories were created to implement project requirements. They were assigned story points to estimate how long each story would take. These were determined by talking in individual team member's estimations and taking an average. Stories are then moved into sprints and have a to-do, In-progress, and Done status to keep track of progress.

Sprints have a set of user stories and must be completed in the chosen sprint cycle duration. We chose two two-week duration for our sprint cycles. We have weekly stand-ups to update each other on our progress and get support on tasks. A screenshot of all sprint backlogs with stories and assignments can be found in the appendix. **The tasks chosen at each sprint were based on the priority that the team has delegated and discussed during sprint backlog grooming**.

3.2.7.1 Sprint 1.1

Date - September 27th, 2023 to October 11th, 2023

In Sprint 1.1 the focus was to learn how to use the agreed upon tools and technologies for design of the solution. We kicked off by finalizing the system architecture and investigating different Database Management System (DBMS) products. Practical tests were

conducted on the RabbitMQ event broker to ensure its suitability for our needs. On the development side, we covered essential aspects such as Python and Typescript fundamentals, REST APIs, and frontend fundamentals. Additionally, we delved into understanding what calculations are necessary for the project and explored UI/UX design considerations. We also investigated alternatives to RabbitMQ such as Apache Kafka, ActiveMQ and Apache Camel before fully committing. Another key component was to initialize the GitHub organization and all project repositories. This sprint laid a solid foundation for the project's development, incorporating backend and frontend fundamentals, crucial architectural decisions, and strategic product investigations.

Priority Justification & Sprint Goals - Prioritizing fundamental skills in key technologies and tools such as JavaScript/TypeScript, Python, and REST APIs ensures that the development team is equipped to tackle the technical challenges of the project efficiently and effectively. Establishing a clear system architecture and choosing the right database and event broker products in the initial sprint are critical for laying a strong and scalable foundation, thereby enabling more focused and productive development in subsequent phases of the project.

Stories & Epics

1. As a developer, I want to become proficient in the fundamentals of JavaScript/-Typescript front end development to get started on system development.
2. As a developer, I want to become proficient in the fundamentals of REST APIs to get started on system development.
3. As a developer, I want to become proficient in the fundamentals of python to get started on system development.
4. As a developer, I want to explore various Event Broker products to select the most suitable for the system.
5. As a developer, I want to explore at least three open source database providers to select the most suitable for the system.
6. As a developer, I want to have a general idea of the system architecture to facilitate better development.
7. As a developer, I want to explore what aspects of the system require calculations to plan accordingly.
8. As a developer, I want to test the RabbitMQ framework to integrate it within the solution.
9. As a developer, I want to to work within a repository on Github to track changes and allow for open-source contribution.

3.2.7.2 Sprint 1.2

Date - October 12th, 2023 - October 26th, 2023

In Sprint 1.2, our team achieved significant milestones in developing the core elements of the project. We focused on essential aspects such as designing the database, setting up the backend infrastructure, and investigating server needs. Additionally, we delved into critical calculations related to satellite trajectory and ground station positional data. Non-engineering tasks included researching current implementations of similar web apps and creating a project charter. On the microservices front, we explored functionalities such as image requests, integration of satellite calculations, and outbound functionality for ground stations. The sprint culminated in the successful completion of designated tasks, marking substantial progress in shaping the project's foundational components.

Priority Justification & Sprint Goals - In Sprint 1.2, prioritizing the development of core infrastructure components like the database setup and backend was essential to ensure that the foundational technology stack was robust and scalable. This foundational setup facilitates efficient data handling and operations crucial for later stages of development, especially for real-time satellite and ground station calculations. The focus on critical calculations for satellite trajectories and ground station positions directly supports the functional requirements of the satellite operators, enabling precise monitoring and management capabilities early in the project lifecycle. Additionally, exploring comparative analyses with existing products and investigating deployment options like NGINX or Kubernetes helped identify optimal tools and technologies, ensuring that our solution remains competitive and well-suited for the intended operational environment. These priorities are strategically aligned to set a solid groundwork for introducing more complex features such as microservices for image requests and integration of calculated data into satellite operations, crucial for advancing the project's technical maturity and operational readiness.

Stories & Epics

1. As a satellite operator, I'd like to perform satellite calculations on the TLE to retrieve specific information I need.
2. As a satellite operator, I'd like to perform ground station calculations to determine positional properties.
3. As a developer, I'd like to compare my potential product against the competition to see where I stand.
4. As a developer, I'd like to investigate which of NGINX or Kubernetes fits my needs better to facilitate deployment.

5. As a satellite operator, I'd like to work with specific data schemas to keep data organized.
6. As a satellite operator, I'd like to access image request data to monitor statuses of each.
7. As a satellite operator, I'd like to access activity request data to monitor activities at hand.
8. As a developer, I'd like my initial codebase to be setup in order to direct other developers on how the system should look like.

3.2.7.3 Sprint 1.3

Date - November 1st, 2023 - November 10th, 2023

Sprint 1.3 encompassed both backend and frontend tasks, with notable progress made in integrating user-based requests for time intervals into mock ground stations and satellites, testing ground station uplinking and downlinking capabilities during access intervals with satellites, and implementing calculations for the field of view (FOV) during image capture for satellite operations. Additionally, a power management system for satellites was successfully introduced, enhancing the overall robustness of the satellite calculations. Substantial progress was made on the frontend by completing the design and implementation for the minimal viable product (MVP). This achievement highlights our commitment to delivering a polished and user-friendly interface, setting the stage for further enhancements and improvements.

Priority Justification & Sprint Goals - During Sprint 1.3, prioritizing the integration of user-based requests for time intervals into simulations of ground stations and satellites was crucial for testing the system's operational capabilities. This focus ensured the development of robust communication functionalities for uplinking and downlinking, which are pivotal for effective satellite operations. The implementation of field of view calculations and a power management system for satellites further solidified the backend's accuracy and efficiency, essential for scheduling and long-term satellite health monitoring. Concurrently, significant advancements on the frontend, culminating in the completion of the MVP, aimed to ensure that the user interface was intuitive and efficient, enhancing user engagement and satisfaction.

Stories & Epics

1. As a satellite operator, I'd like to be able to mock the interactions between the system and the ground station to account for future ground station interactions.
2. As a developer, I'd like any external calculations to be integrated within the Scheduler microservice to keep it organized.
3. As a satellite operator, I'd like to be able to access satellite health data in order to monitor satellites statuses.
4. As a developer, I'd like to be able to test and experiment mock GS uplinking and downlinking capabilities.
5. As a satellite opeartor, I'd like to be able to calculate FOVs for image captures in satellite calculations to aid in scheduling.
6. As a developer, I'd like my time interval requests to be integrated into a ground station.

7. As a satellite operator, I'd like to have a power management system for satellite calculations to aid in scheduling.
8. As a student in Capstone, I'd like to work on the Oct 29 progress report to get an A+.
9. As a satellite operator, I'd like to be able to input a maintenance request to maintain assets.
10. As a satellite operator, I'd like to be able to determine the status of my ground station so that I can fix any issues that arise with it.
11. As a satellite operator, I'd like to be able to view my maintenance requests in a table to get an overview of them.
12. As a satellite operator, I'd like to be able to add satellites to facilitate more image order fulfillments.
13. As a satellite operator, I'd like to be able to add ground stations to facilitate more image order fulfillments.

3.2.7.4 Sprint 1.4

Date - November 12th, 2023 - November 28th, 2023

The primary focus for Sprint 1.4 centered on establishing API connections between microservices. We dedicated efforts to enhance communication and data exchange among various components of the system. Key achievements include implementing Activity Request functionality into the Satellite-Activity Microservice, integrating Image Request functionality into the Image-Management Microservice, and incorporating Outbound functionality into the Ground Station Outbound Microservice. These efforts contribute significantly to the seamless integration and functionality of our microservices architecture.

Priority Justification & Sprint Goals - Sprint 1.4 was strategically focused on enhancing the interconnectedness and operational efficiency of our microservices architecture. By establishing robust API connections between the microservices, the team ensured that data exchange and communication across the system were seamless and reliable. This integration is critical for supporting complex functionalities such as Activity Requests, Image Requests, and Outbound operations specific to ground stations, which are fundamental to the system's capability to process and respond to user commands effectively.

Stories & Epics

1. As a student in Capstone, I'd like to be able to work on the Project Chart.
2. As a developer, I'd like to be able to coordinate and test events sent to the scheduler to check for correctness.

3. As a satellite operator, I'd like to test the system with randomly generated data in the DB to perform quick minor tests.
4. As a developer, I'd like to adjust my data models to fit my needs better.
5. As a satellite operator, I'd access scheduling data to monitor schedules.
6. As a satellite operator, I'd like to submit image orders in bulk files to a server through the File Transfer Protocol.
7. As a satellite operator, I'd like to be able to fetch image order data to track image orders.
8. As a satellite operator, I'd like to be able to add new ground stations to facilitate better image order fulfillment.
9. As a developer, I'd like to add outbound functionality into a ground station outbound microservice.
10. As a satellite operator, I'd like to add activity status data
11. As a developer, I'd like to integrate image request functionality into image management service to maintain organization.
12. As a developer, I'd like to add activity request functionality into the satellite activity microservice to maintain organization.
13. As a satellite operator, I'd like my scheduler to be thoroughly tested and validated by members of the CSA.
14. As a developer, I'd like to finalize mapping architecture for satellite and ground stations to facilitate development.
15. As a satellite operator, I'd like to be able to submit outage request orders to maintain satellite health.
16. As a satellite operator, I'd like to be able to view outage requests in a tabular format to get an overview of such data.
17. As a student in Capstone, I'd like to prepare a mini-capstone day presentation.
18. As a student in Capstone, I'd like to evaluate the graded fall progress report.
19. As a developer, I'd like to integrate FOV functionality and access point generation into the models to be accessible whenever.
20. As a satellite operator, I'd like to be able to alter and adjust image order and schedule data to account for shortcomings or errors.

3.2.7.5 Sprint 1.5

Date - December 27th, 2023 - January 14th, 2024

Focus of Sprint 1.6b was to tackle multiple aspects of the system. Most importantly, was the beginning of the development of the scheduling algorithm. Multiple scheduling

approaches were tasked to various individuals to try out and figure out what's viable. Some approaches included utilizing a heauristic algorithm, utilising Reinforced learning and information retrieval. This sprint is also focused on major front end features such as the implementation of the maps that visualize the satellites and groundstations, as well as designing a logo as a brand for the site. There were also backend tasks involving low level conversions between certain data types and polishing. Major backend tasks such as the implementation of APIs to update the front end, as well as the FTP server implementation were all present in the sprint. These efforts contribute greatly to the core functionalities of the system and at building it's automation aspect.

Priority Justification & Sprint Goals - In Sprint 1.5, the focus was centered on advancing the core technological foundations of our system, particularly through the development of the scheduling algorithm, which is pivotal for optimizing satellite operations. Exploring various scheduling techniques, including heuristic algorithms and machine learning approaches, was essential to identify the most efficient and effective method for managing complex scheduling scenarios involving multiple satellites and ground stations. This development is crucial for enhancing the system's automation capabilities and ensuring operational efficiency.

Stories & Epics

1. As a developer, I'd like to be able to translate recurrence attributes of Image Request to an Image request order.
2. As a developer, I'd like to be able to research scheduling algorithm implementations.
3. As a developer, I'd like to attempt a heuristic approach to scheduling.
4. As a developer, I'd like to attempt a machine learning approach to scheduling.
5. As a satellite operator, I'd like to visually see where my assets are positioned on the globe.
6. As a satellite operator, I'd like to be able to view maps in 2D and 3D.
7. As a satellite operator, I'd like the system to have a logo in order to be distinguishable.
8. As a developer, I'd like to parse image order data for parent and child relationships.
9. As a developer, I'd like to have a clean minimalized back-end structure.
10. As a developer, I'd like to implement topic exchanges.
11. As a satellite operator, I'd like to have a better looking interface.
12. As a developer, I'd like to have a basic understanding of orbitals.
13. As a satellite operator, I'd like to have an up to date view of the data on my assets.
14. As a satellite operator, I'd like to be able to access activity request data.
15. As a student in Capstone, I'd like to work on the Project Charter.

3.2.7.6 Sprint 1.6

Date - January 14th, 2024 - January 28th, 2024

This sprint was focused on further scheduling algorithm development which included trying out the particle swarm optimization in scheduling, as well as outputting satellite schedules. There were also specific backend atomic functionalities that needed to be adjusted and or updated as the project evolved which included refining the event-relay and image management microservices. Finally, there was the collective assignment of working on the peer review tasks coming up. All in all, this sprint was focused on improving the core scheduling functionality of the system to provide the user with a better output of schedules.

Priority Justification & Sprint Goals - Sprint 1.6 was strategically centered on enhancing the core functionality of the system through the advanced development of scheduling algorithms, particularly by experimenting with particle swarm optimization techniques. This approach aimed to optimize the efficiency and effectiveness of satellite scheduling outputs, a crucial aspect for satellite operators who rely on precision and reliability in their workflows. Concurrently, the sprint addressed necessary updates and refinements in backend services like the event-relay and image management microservices, ensuring these components function seamlessly and support the overall system performance. Additionally, the focus on establishing a robust peer review process and continuous integration and delivery pipelines highlights the team's commitment to maintaining high-quality code and facilitating smoother development cycles.

Stories & Epics

1. As a developer, I want to have a more readable codebase for the event-relay API microservice to aid in future development.
2. As a satellite operator, I want all individual units in event relay and image management to produce the desired output.
3. As a satellite operator, I want satellite schedules to be automatically generated to make my workflow quicker.
4. As a developer, I want to use particle swarm optimization to produce schedules to provide a more effective schedule.
5. As a developer, I want to ensure orders are parsed into parent and children accurately.
6. As a satellite operator, I want to updates on satellite eclipses and ground station to satellite contact opportunities to figure out how best to adjust a schedule.
7. As a developer, I want to have an automated continuous integration and continuous delivery pipeline for the system's version control actions to automate testing.

8. As a developer, I want to have well defined scoring functions for genetic algorithm optimization.
9. As a student of Capstone, I want meeting notes in case I need to refer back to the meeting content.
10. As a student of Capstone, I need to work on peer Review for Winter Semester 1.
11. As a developer, I want to define detailed performance metrics to evaluate the system accurately.

3.2.7.7 Sprint 1.7

Date - January 28th, 2024 - February 11th, 2024

This sprint carried on the previous tasks related to scheduling, more specifically the particle swarm optimizations. This sprint also focused on dealing with recurring image orders in the context of the scheduling algorithm. Furthermore, more backend tasks pertaining to adding endpoints that share details about the satellites and respective ground stations were implemented. Finally, research was begun on possible hosting platforms to deploy the system on the cloud.

Priority Justification & Sprint Goals - Sprint 1.7 was dedicated to advancing the particle swarm optimization techniques within the scheduling algorithms, crucial for refining the generation of effective and efficient satellite schedules. This sprint also tackled the integration of recurring image orders into the scheduling algorithm, enhancing the system's automation capabilities for managing complex, ongoing satellite operations. Additionally, the implementation of new backend endpoints to provide detailed satellite and ground station data significantly boosted operational transparency. Research into potential cloud hosting platforms marks a strategic preparation for system deployment, ensuring scalability and robustness. These efforts are instrumental in strengthening the core functionalities and preparing the system for a scalable future, aligning with the overarching goals of enhancing operational efficiency and system reliability.

Stories & Epics

1. As a satellite operator, I want detailed information regarding power constraints on satellites to help with verifying automated schedules.
2. As a developer I want to use particle swarm optimization to produce more schedules potentially containing an efficient one.
3. As a satellite operator, I want to view the health of ground station and satellite assets to help with verifying automated schedules.

4. As an imaging customer, I want to send orders that have recurrences built in to be handled by the systems automated nature
5. As a satellite operator, I want satellite and ground station schedules to be sent to the ground stations in order to send them to satellites.
6. As a student of Capstone, I want user stories to reflect user requirements.
7. As a satellite operator, I want to be given hourly updates on satellite eclipses and ground station to satellite contact opportunities to have a better understanding of asset functions.
8. As a developer, I want to select a hosting platform and purchase a domain based on a trade study to deploy the system effectively.
9. As a student of capstone, I'd like to write meeting notes to refer to in the near future.
10. As a satellite operator, I want full integration of the deterministic scheduling algorithm in the server side code to be able to automate schedules for newly incoming image orders.
11. As a satellite operator, I want to be able to deploy the client side module as an open source container to easily run the interface on any machine.
12. As a developer, I want to define and document a well formed testing environment to guarantee effective correctness checking.

3.2.7.8 Sprint 1.8

Date - February 11th, 2024 - February 25th, 2024

This sprint marks the beginning of our testing phase. Where we have begun writing up unit test cases for the atomic components of our backend system. This sprint also handles optimizations in the scheduling algorithm that pertain to the ability to adjust schedules properly with the addition of new image orders to the processed batch of orders.

Priority Justification & Sprint Goals - The focus of Sprint 1.8 on initiating the testing phase and writing unit test cases for backend components is a strategic step to ensure the reliability and stability of the system's core functionalities. By rigorously testing each atomic component, we aim to identify and resolve potential issues early, enhancing the overall robustness of the system. Concurrently, the sprint's emphasis on optimizing the scheduling algorithm to dynamically adjust to new image orders without disrupting the existing schedule is critical for maintaining operational efficiency and responsiveness. These optimizations help in effectively managing the influx of new requests, thereby minimizing delays and optimizing resource allocation. This blend of rigorous testing and targeted

algorithmic enhancements directly supports the project's objectives to deliver a dependable and efficient scheduling system, capable of adapting to changing operational demands.

Stories & Epics

1. As a Satellite Operator, I want to be able to create randomized orders for testing purposes.
2. As a Satellite Operator, I want new activities to affect the existing schedule only within a limited window in order to quickly send out the schedule and save time.
3. As a Developer, I want to keep track of bugs and issues through Github Issues to establish a formal bug reporting process (Bug Reporting).
4. As a Developer, I want all individual units in the back-end to produce the desired output (Unit Testing).
5. As a Developer, I want the code to reflect the updated database schema and utilize App-Config (Update and refine Code).

3.2.7.9 Sprint 1.9

Date - February 25th, 2024 - March 10th, 2024

This sprint marks the continuation of testing procedures more specifically the beginning of the integration and E2E testing. The sprint also tackles further scheduling algorithm optimizations such as proper balancing of workloads between satellites in a schedule. This sprint also is focused on integrating the back-end components with the front end components which marks a significant step in the system's evolution. All done to prepare the system for demonstration.

Priority Justification & Sprint Goals - Sprint 1.9 is pivotal as it expands the testing framework to include integration and end-to-end (E2E) testing, crucial for ensuring that all system components work harmoniously and meet operational standards before the final demonstration. This comprehensive testing is essential not only for identifying integration issues between backend and frontend components but also for validating the complete user experience and workflow. Additionally, the sprint focuses on optimizing the scheduling algorithm to ensure workload balancing among satellites, which is critical for maximizing resource utilization and operational efficiency. By improving the scheduling logic to only allocate image captures when targets are within a satellite's field of view, the system enhances its operational precision and reduces inefficiencies. This strategic focus on advanced testing and algorithm refinement ensures that the system is robust, efficient, and ready for demonstration, aligning with the project's goals to deliver a high-performing and reliable satellite operations platform.

Stories & Epics

1. As a student of Capstone, I want user stories to reflect user requirements.
2. As a Satellite operator, I want generated schedules to prioritize optimization metrics to meet efficiency standards.
3. As a Satellite operator, I want image captures scheduled only when the area of interest is in the satellite's FOV to determine reject-able orders.
4. As a Satellite operator, I want the data I see to be updated constantly to avoid any related issues.
5. As a Satellite operator, I want assets to be utilized equally to produce efficient schedules.
6. As a Satellite operator, I want to receive schedules by only uploading Image orders or inputting satellite activity orders.
7. As a Developer, I want specific scheduler service functionalities to be tested in order to validate the functioning of the system.

3.2.7.10 Sprint 1.10

Date - March 17th, 2024 - April 3rd, 2024

The final sprint concludes the integration of the front end with the back end. As well as the integration of the scheduling algorithm with the entire system. Further front end features are added such as the health data dashboard outlining various summaries of data. As well as other front end and back end tweaks and fixes.

Stories & Epics

1. As a Satellite operator, I want to see data visualized so I can analyze orders and schedules.
2. As a Satellite operator, I want to be able to access the full E2E system locally to demonstrate to others.
3. As a Satellite operator, I want to receive the correct set of data for dashboard charts to have a general idea about scheduling performance.
4. As a Satellite operator, I want to be able to send and view the outage requests for my assets to maintain their health.
5. As a Satellite operator, I want to be able to view each individual maintenance, outage, and image requests.
6. As a Satellite operator, I want to be able to decline orders/schedules to avoid future/unpredictable predicaments.

7. As a Satellite operator, I want to generate test data to test the system for various "day in the life" scenarios.
8. As a Satellite operator, I want to receive correct outputs for every input to ensure correct functioning of the scheduler.
9. As a developer, I want communication between services and components to be accurate.
10. As a Satellite operator, I want timing constraints to be used in the correct time scale.

Priority Justification & Sprint Goals - The focus of Sprint 1.10 on finalizing the integration of frontend and backend components, as well as the comprehensive incorporation of the scheduling algorithm, marks a crucial phase in the project, ensuring that all parts of the system work seamlessly together. This sprint is essential for establishing a fully operational system capable of real-world demonstrations. The addition of a health data dashboard enhances the system's usability by providing satellite operators with visual summaries of data, crucial for effective decision-making and operational monitoring. Further tweaks and fixes in both frontend and backend are aimed at polishing the system to eliminate any remaining bugs and enhance user experience. These final adjustments are critical to ensure that the system is reliable, user-friendly, and ready for rigorous real-life applications.

3.2.7.11 Timeline & Final Product Backlog

To view the full list of stories as a timeline view from our Jira story dashboard please visit this GitHub repository for SatOpsMQ supplementary materials. There are three views of the full timeline available: weekly, monthly and quarterly. The timeline is the view the developers used on a regular basis along with product and sprint backlogs.

The team was able to complete the web application with all the necessary features that the user (CSA) required and that was satisfactory for the supervisor. The stories shown so far are those that have been completed.

However, we did have some user stories that we could not finish in the project that were considered as "nice-to-haves" for the users. As we completed the project, these were the remaining tasks left to add to the web-application but doesn't necessarily impact the performance/functionality.

Backlog (Jira)

JQL Query: type in (Story, Task) and Sprint is EMPTY and status in ("To Do") order by created DESC
Sorted by: Created descending
1–15 of **15** as at: **03/Mar/24 9:49 PM**

T	Key	Summary	Assignee	Reporter	Status	Sprint
	ENG4SOSO-165	As a satellite operator I want timing constraints to be used in the correct time scale	*Unassigned*	Youssef Hany	TO DO	
	ENG4SOSO-164	As a member of the satopsmq team I want the conceptual architecture of the system to reflect the system accurately	*Unassigned*	Youssef Hany	TO DO	
	ENG4SOSO-162	As a member of the satopsmq team I want to keep track of all meeting notes	*Unassigned*	Youssef Hany	TO DO	
	ENG4SOSO-160	As a member of the satopsmq team I want to document project progress up until 2024-03-02	*Unassigned*	Youssef Hany	TO DO	
	ENG4SOSO-154	As a member of the satopsmq team I want to document the final project, development and project management process	*Unassigned*	Hashir Jamil	TO DO	
	ENG4SOSO-141	As a satellite operator I want to see data visualized so I can analyze orders and schedules	*Unassigned*	Rafael Dolores	TO DO	
	ENG4SOSO-134	As a satellite operator I only want to receive relevant data	Stanley Ihesiulo	Walid AlDari	TO DO	
	ENG4SOSO-130	As a satellite operator I want to be able to request outage orders to a satellite or ground station	*Unassigned*	Stanley Ihesiulo	TO DO	
	ENG4SOSO-122	As a member of the satopsmq team I want the project repository to be align with a DevOps methodology	*Unassigned*	Rafael Dolores	TO DO	
	ENG4SOSO-115	As a member of the satopsmq team I want to keep all project diagrams updated and in the same place	*Unassigned*	Rafael Dolores	TO DO	

T	Key	Summary	Assignee	Reporter	Status	Sprint
	ENG4SOSO-109	As a satellite operator I want to be able to access the system over the Internet	*Unassigned*	Rafael Dolores	TO DO	
	ENG4SOSO-106	As a satellite operator I want to be able to export schedules in a CSV format	*Unassigned*	Rafael Dolores	TO DO	
	ENG4SOSO-105	As a satellite operator I want to be able to remove satellites from the saved satellites	*Unassigned*	Rafael Dolores	TO DO	
	ENG4SOSO-100	As a satellite operator I want to be able to see relevant satellite status data	*Unassigned*	Ruth Bezabeh	TO DO	
	ENG4SOSO-95	As a satellite operator I want to be able to decline orders/schedules	*Unassigned*	Rafael Dolores	TO DO	

Figure 3.4: Backlog of stories left over after Sprint 1.10

3.3 Resource Allocation

In this section we show detailed breakdowns of each team member and their story points broken down by sprint and by month. The number represent only the final values of story points once tasks were completed. Since the initial progress report, our approach to resource allocation has evolved to reflect the dynamic nature of project development and the varying complexities encountered in different sprints. The modifications in resource allocation were prompted by several factors:

- **Task Complexity Adjustments:** As our understanding of the project's intricacies deepened, we refined our estimation of story points to more accurately represent the effort required for each task. This resulted in adjustments to the final values of story points upon task completion, ensuring a more precise measure of workload and contribution.
- **Team Member Capabilities:** We recognized the diverse strengths of our team members and reallocated tasks to leverage individual expertise.
- **Adaptive Sprint Planning:** Our sprint planning became more adaptive to account for unforeseen challenges and opportunities for innovation. This adaptability was necessary to maintain project momentum and to ensure that resource allocation remained aligned with project goals.
- **Feedback Loops:** Incorporating feedback from stakeholders and iterative testing led to re-prioritization of features and tasks. The allocation of story points was updated to mirror these shifting priorities.
- **Risk Mitigation:** As risks were identified and managed, resource allocation was updated to mitigate potential impacts on the project timeline and quality.
- **Process Improvement:** Continuous process improvement initiatives resulted in the reallocation of resources to optimize workflows and enhance collaboration.

3.3.1 Story Point Calculation

To estimate effort and prioritize tasks effectively, we utilize Scrum Poker during our sprint planning as mentioned previously. Scrum Poker is a collaborative estimation technique where team members assign story points to tasks based on their complexity and effort. This method encourages team members to discuss and share their perspectives on each task, leading to a more accurate and consensus-driven estimation. Story points help provide a relative measure of effort, allowing for a clearer understanding of the workload distribution for the sprint. Story point estimation is continually corrected by developers if after the

task they believe their actual time diverged from the estimated time. Moreover, we choose ideal work hours as story points. Each story point is the equivalent of an ideal work hour. Other metrics include ideal work days but due to the part time nature of this project, this metric was not used.

3.3.2 Story Points By Team Member

Here we show the graphical breakdown of each team members story points by sprint and by month. The aggregated monthly break down for all developers is shown here and the break down by sprint is shown here

	Sprint								
Name	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	Total
Walid	33	45	37	26	7	10	13	20	191
Rafael	35	22	83	47	12	10	40	50	299
Stanley	20	30	70	26	12	0	40	10	208
Ruth	5	69	10	6	27	0	8	8	133
James	18	38	15	21	12	15	6	0	125
Hashir	5	58	10	31	17	0	0	5	126
Youssef	21	43	5	26	22	7	8	0	132

Table 3.1: Sprint-wise Story Points progress for each team member

	Nov	Jan	Feb	Mar	Total
Walid	78	63	17	33	191
Rafael	57	130	22	90	299
Stanley	50	96	12	50	208
Ruth	74	16	27	16	133
James	56	36	27	6	125
Hashir	63	41	17	5	126
Youssef	64	31	29	8	132

Table 3.2: Monthly Story Points progress for each team member

3.3.2.1 Walid

The figure outlining the breakdown for Walid's story points can be seen by clicking here.

3.3.2.2 Hashir

The figure outlining the breakdown for Hashir's story points can be seen by clicking here.

3.3.2.3 Rafael

The figure outlining the breakdown for Rafael's story points can be seen by clicking here.

3.3.2.4 James

The figure outlining the breakdown for James's story points can be seen by clicking here.

3.3.2.5 Ruth

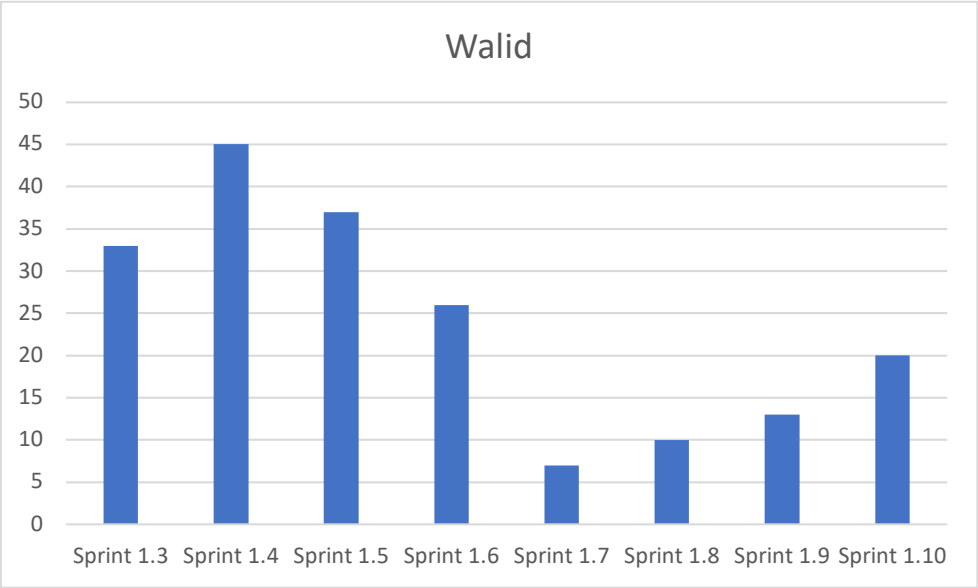
The figure outlining the breakdown for Ruth's story points can be seen by clicking here.

3.3.2.6 Stanley

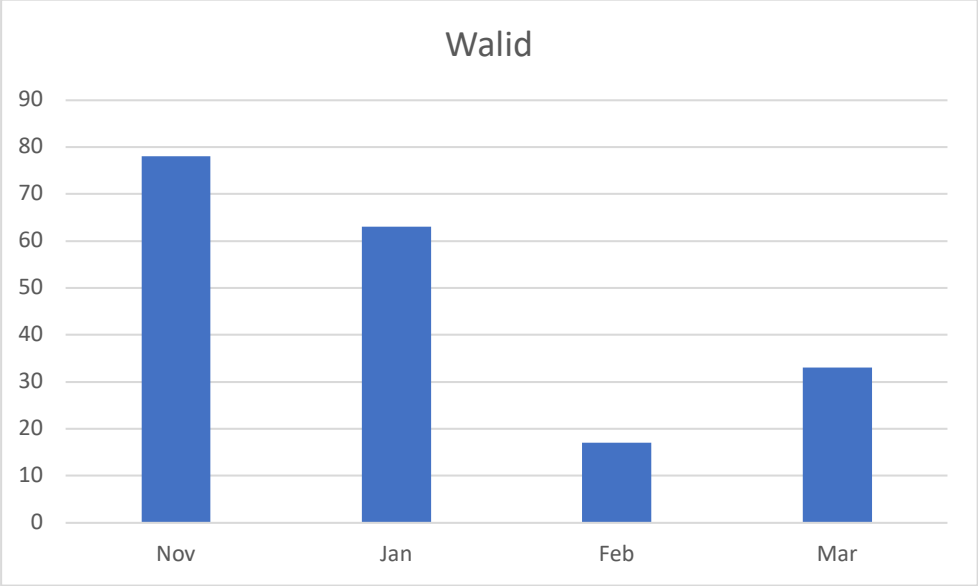
The figure outlining the breakdown for Stanley's story points can be seen by clicking here.

3.3.2.7 Youssef

The figure outlining the breakdown for Youssef's story points can be seen by clicking here.

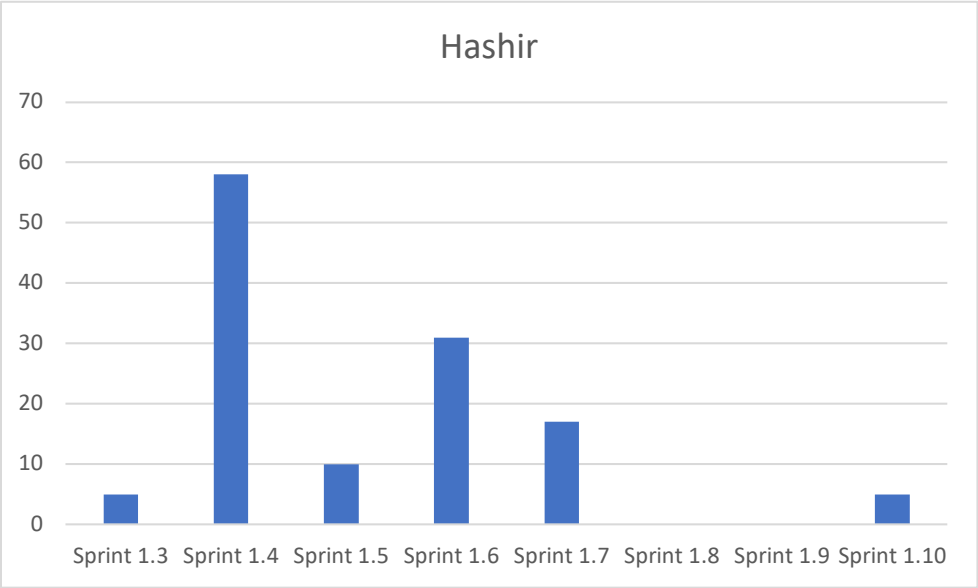


(a) Sprint Breakdown

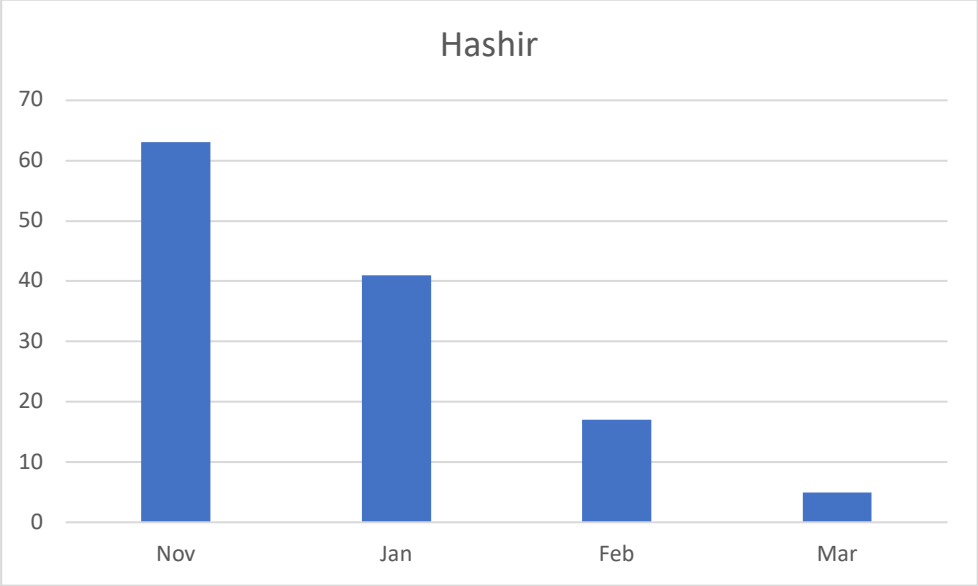


(b) Monthly Breakdown

Figure 3.5: Story Points Breakdown for Walid

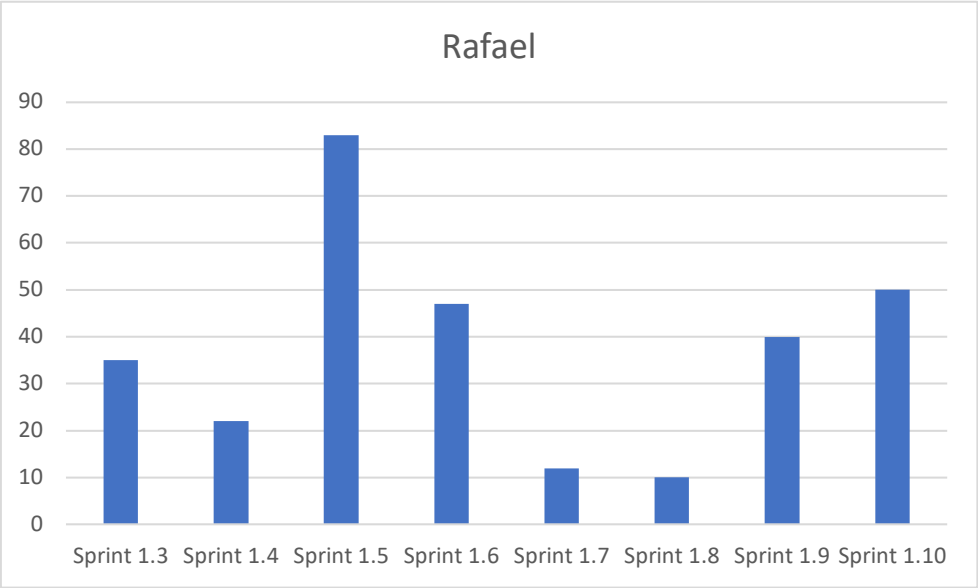


(a) Sprint Breakdown

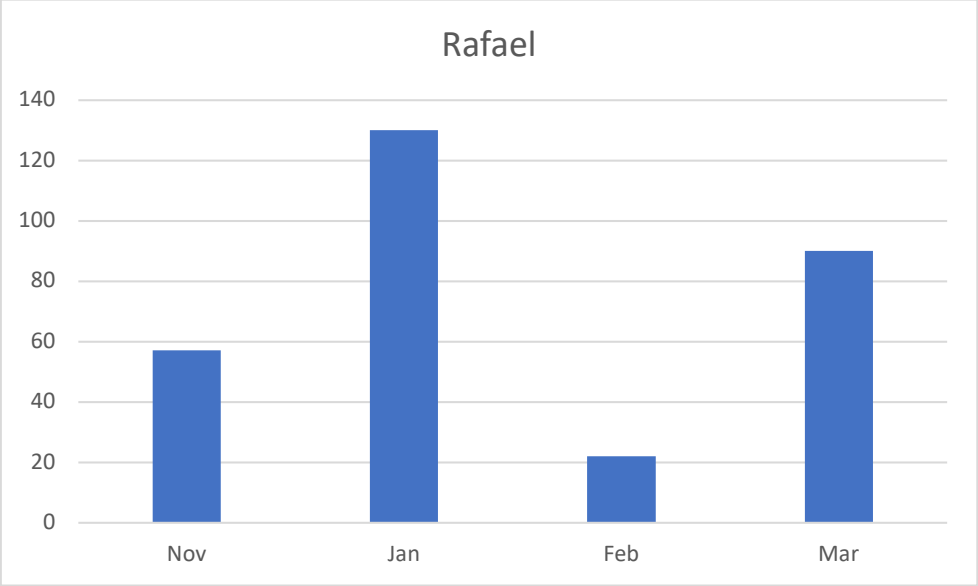


(b) Monthly Breakdown

Figure 3.6: Story Points Breakdown for Hashir

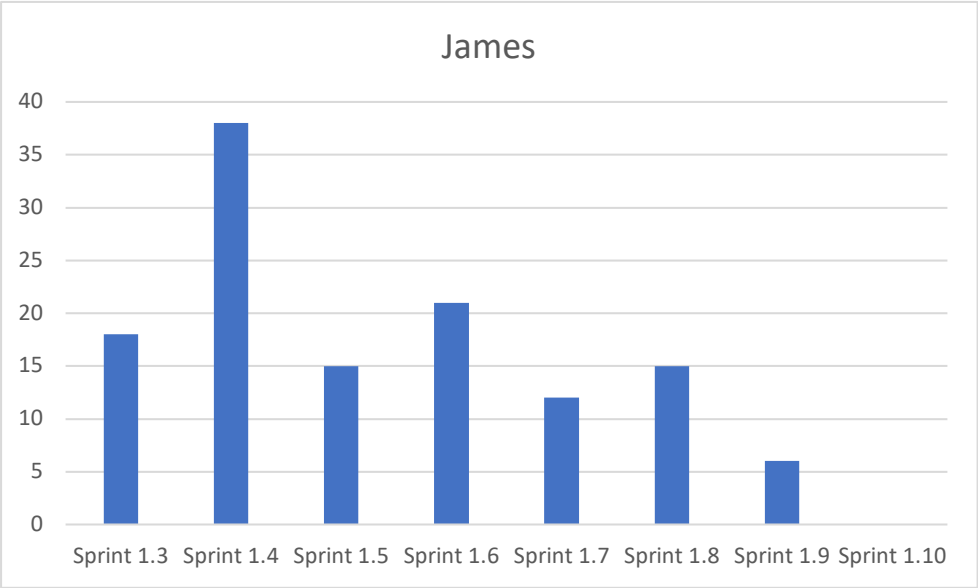


(a) Sprint Breakdown

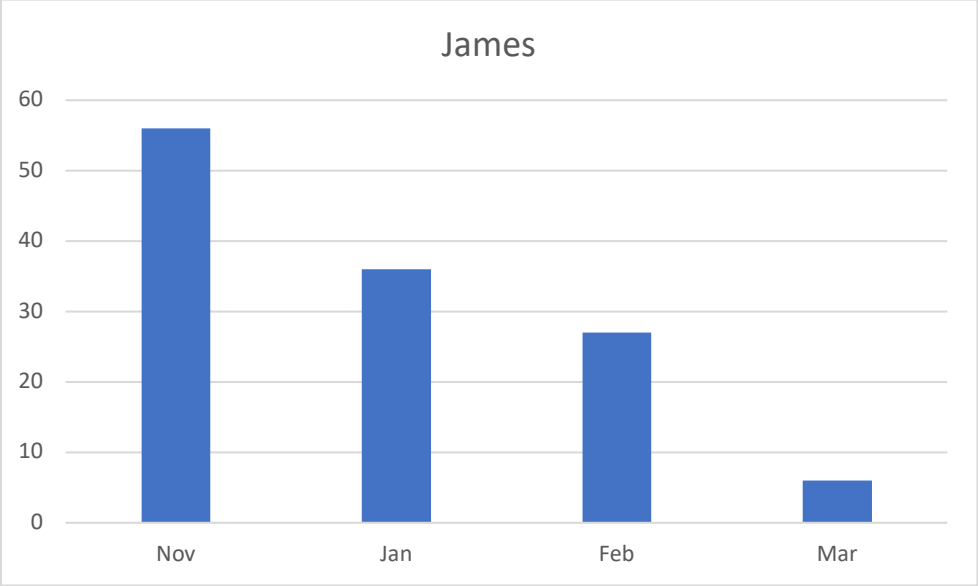


(b) Monthly Breakdown

Figure 3.7: Story Points Breakdown for Rafael

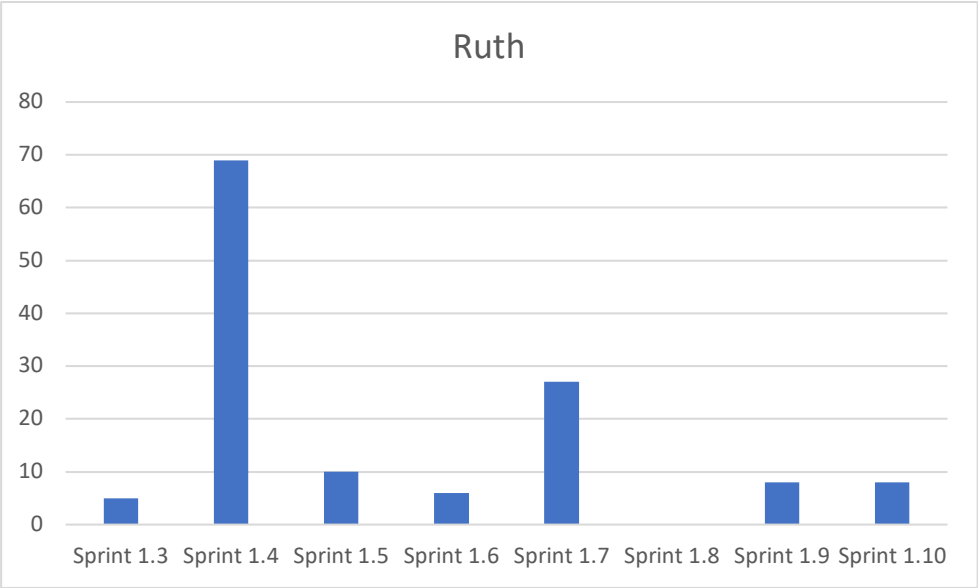


(a) Sprint Breakdown

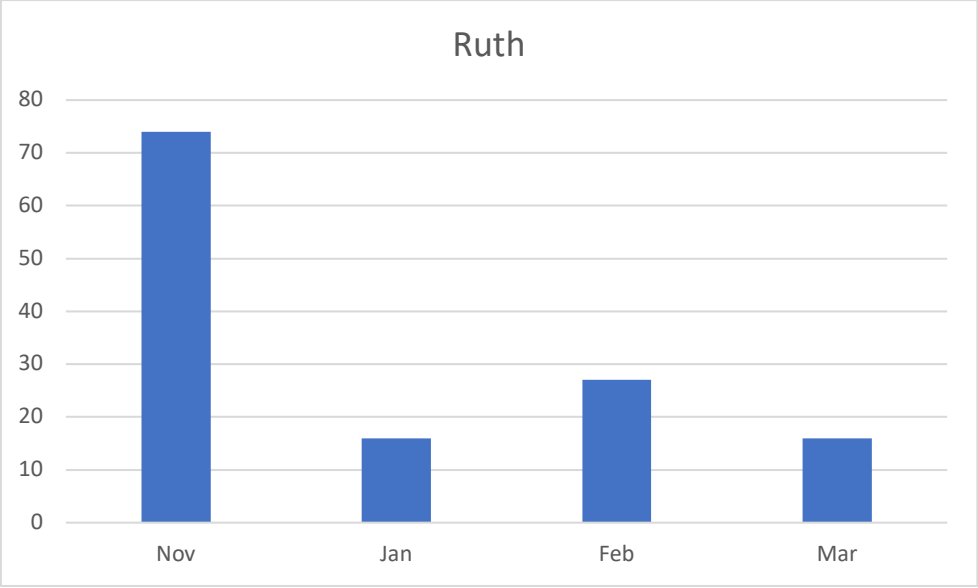


(b) Monthly Breakdown

Figure 3.8: Story Points Breakdown for James

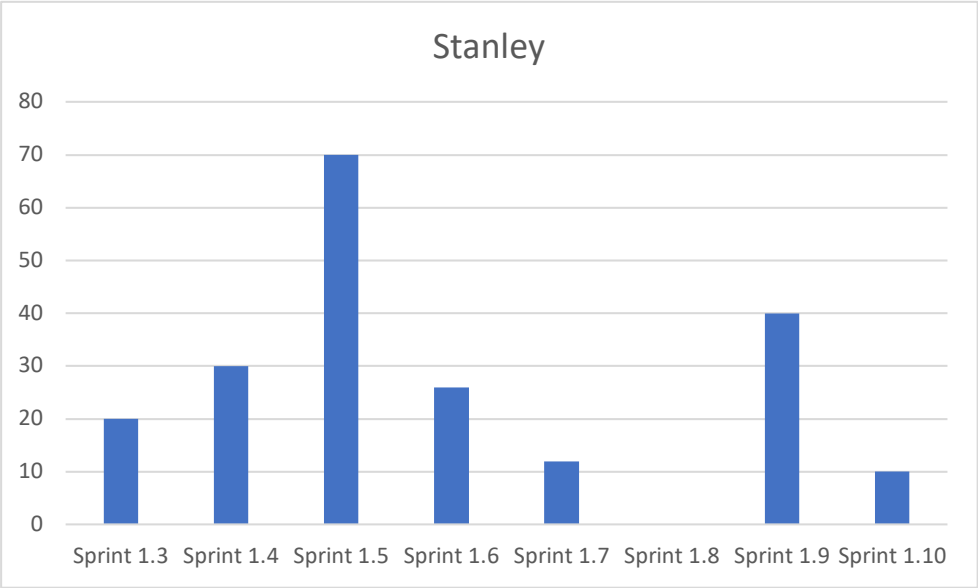


(a) Sprint Breakdown

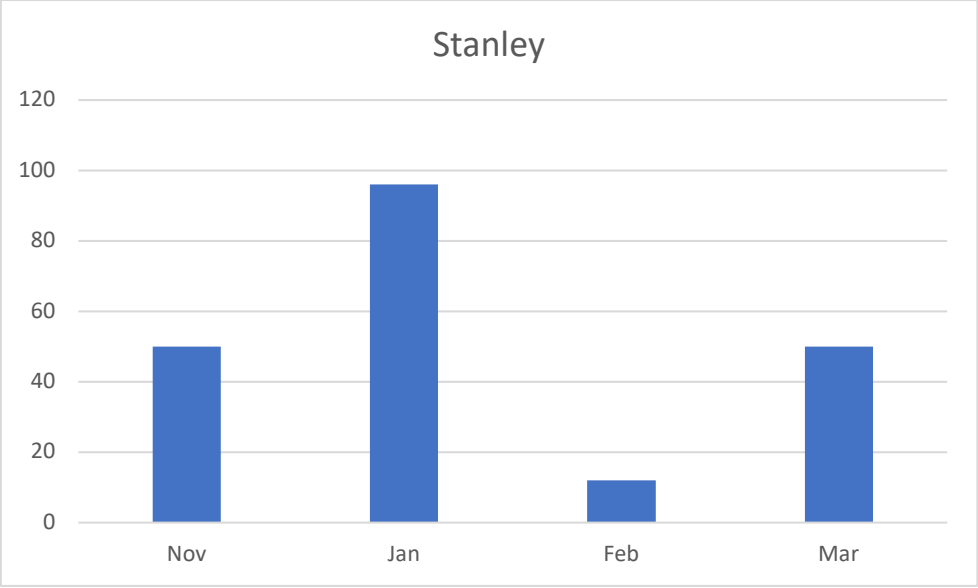


(b) Monthly Breakdown

Figure 3.9: Story Points Breakdown for Ruth

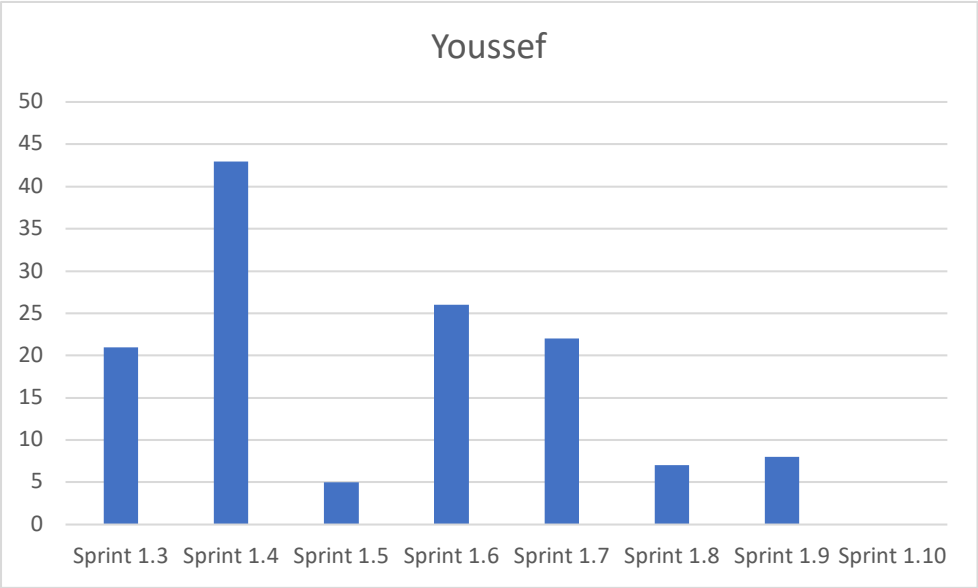


(a) Sprint Breakdown

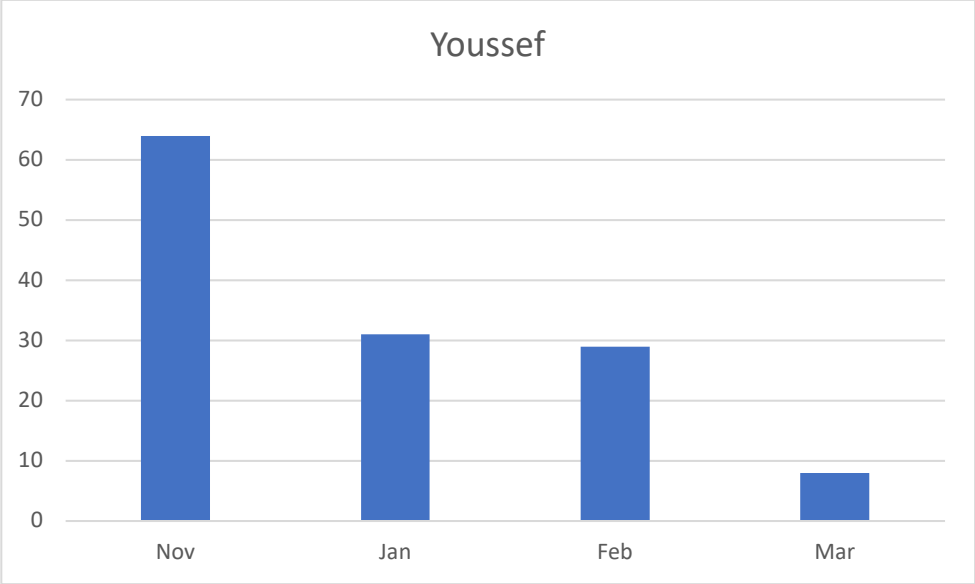


(b) Monthly Breakdown

Figure 3.10: Story Points Breakdown for Stanley



(a) Sprint Breakdown



(b) Monthly Breakdown

Figure 3.11: Story Points Breakdown for Youssef

3.3.3 Code Review

Each team member was allocated specific project areas based on their skills. The team is organized into three groups: backend developers, frontend developers, and designers/architects. This structured division allows each group to focus on their specific area of expertise, ensuring a thorough and effective review process. The code review process is shown visually in the code review methodology flow chart.

For backend developers, the code review process is initiated upon the completion of a functionality. Biweekly meetings, conducted at the end of each sprint via Zoom, serve as a platform for these reviews. In these meetings, backend developers present the implemented code, explaining the rationale behind their coding choices and elucidating how their work integrates with and impacts other system components. This collaborative discussion also involves the frontend team, who play a crucial role in assessing the changes from a user interface perspective. The frontend developers focus on evaluating the responsiveness and interactions of the UI components, ensuring that the implemented frontend aligns with the initial UI sketches. They also verify that changes made on the server side are in harmony with the client application's current state. These biweekly review sessions are pivotal in maintaining a cohesive understanding of the system's overall functionality, ensuring that the development efforts on both the frontend and backend are synchronized and aligned with the project's goals.

Designers and architects are integral to the code review process. After each sub-team, comprising backend and frontend developers, completes their respective code reviews, the changes and updates are presented to the designers and architects. This step is crucial as it ensures the developments align with the intended system architecture and design. The designers and architects provide valuable insights into design patterns and architectural principles, evaluating whether the code not only functions as intended but also adheres to the best practices in software design and architecture. Their role extends beyond mere validation; they actively contribute suggestions for improvements or modifications, enhancing the system's design and functionality. This collaborative review process allows for a dynamic evolution of the system's architecture. Designers and architects assess how the current changes can influence future iterations, offering guidance on how the architecture can adapt and progress. Their input is pivotal in shaping the system to be more efficient, scalable, and aligned with the project's long-term vision, ensuring that each code iteration contributes positively to the system's overall growth and sophistication.

Throughout the review process, the team focuses on several critical areas:

Functionality: Ensuring the code performs as intended and meets user needs. Reviewers consider edge cases and concurrency issues, particularly for user-facing changes or

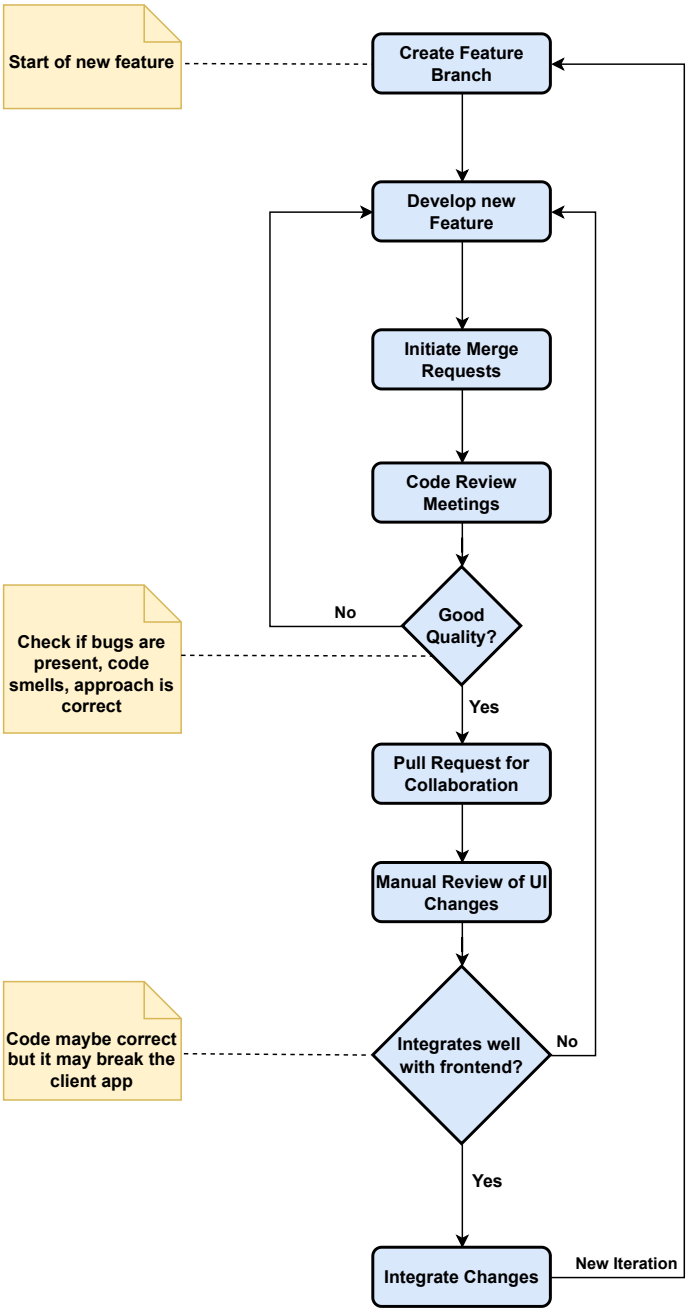


Figure 3.12: Code review methodology flow chart

complex programming scenarios.

Complexity: Assessing whether the code is more complex than necessary. The goal is to achieve simplicity for ease of understanding and maintenance, avoiding over-engineering.

Tests: Evaluating the adequacy and effectiveness of unit, integration, or end-to-end tests. Tests should be sensible, fail when the code is broken, and avoid producing false positives.

Naming: Ensuring that the names used in the code effectively communicate their purpose without being overly lengthy or obscure.

Comments and Documentation: Reviewing comments for clarity and usefulness, primarily focusing on explaining the "why" behind code segments. Documentation should be updated to reflect any changes in how users interact with the system.

Style: Verifying adherence to established style guides, maintaining consistency in the codebase.

Consistency: Ensuring consistency with the style guide and existing code, and promoting coherence throughout the system.

Each line of code is carefully reviewed, and the team looks at the code in the context of the entire system to understand its broader impact. Positive feedback is also given for well-executed elements. This comprehensive and collaborative approach in the Code Review Process ensures well-designed, user-friendly, and maintainable code, adhering to the highest standards of software development.

3.3.4 Story Point Changes

The utility of Agile/Scrum is that it is very open to change. Software development by nature is a constantly uncertain process due to the volatility of a code base and users not being fully certain on how they will use the system until they start using various releases of it. Software (especially web applications with a graphical user interface) is inherently different from solutions in the realm of electronics or manufacturing for example. Our team has always tried to accommodate changes in plans by allowing for leeway in story point assignment and changing which features to prioritize. Some notable changes in stories/story points are listed below. This is not an exhaustive list as there are just too many changes over the course of the eight months to discuss and keep track of. We provide a comprehensive printout of the dashboard view of each sprint in the SatOpsMQ Supplementary Materials Repository for reference. Each sprint can be viewed in terms of burndown and burnup to view story point changes before and after completion.

1. Reduction in number of backend microservices and total modules in the code base.
    - Ground station outbound was removed from the scope and and associated stories were not pursued after the alpha release. Mocking a connection to a ground station did not serve a realistic long term purpose. The focus shifted towards scheduling and visualization of events.
    - The image management service was combined with the FTP server to simplify order upload/delivery.
    - User login/registration was also scrapped early as it was not a feature that fit the user needs and stakeholder/customer vision.
2. Frontend deployment finalization.
    - The initial plan to have the client hosted as in the cloud was not pursued (although the containerization of the frontend client still allows for this if future users of the product prefer)
    - The second option of using electron.js to deploy the client as a desktop application that communicates with a cloud backend was also scrapped.
    - The final option was to use Tauri, a Rust framework as it was found to be superior in performance to electron.js.
3. Scheduling algorithm solution space.
    - The initial plan was to use only genetic algorithm optimization. This resulted in development of scoring functions that would be used in the heuristic algorithm that made it into the final release.
    - We explored a particle swarm optimization but chose to not pursue it further due to computational complexity. This was a total of forty story points spent that did not provide any return to the actual project.
    - As well we explored reinforcement learning as a solution. This was also not fully pursued and was another forty story points.
    - The final algorithm also had many refinements needed and these were further user stories of 40 points (in addition to the initial forty points to make the first iteration)

These are the key changes to user stories/story points over the course of the project. There are dozens of other minor adjustments to story points and specific stories that occurred, however, they have not all been actively tracked as it would be a lot of effort with minimal reward to be this meticulous with story points. The main purpose of Agile is to focus on adding value and not being rigid in the process. We now move to retrospectively analyze and reflect on the sprints in the project using the variouys metrics and charts offered in the free version of Jira software.

3.3.5 Sprint Retrospective & Reflection

In this section we present burndown and burnup reports for each sprint as well the cumulative flow and velocity of the sprints as an aggregate. The purpose of this is to point out what could have been done better and what was done adequately. However, we cannot say with absolute certainty that we did every aspect of project management exceptionally well. Our focus is on understanding our limitations and using this as a learning experience to improve how we approach future projects. The purpose of showing burndown charts is to show how much work is remaining and the purpose of showing a burnup charts is to show how much work has been done. Cumulative flow allowed us to track the number of items finished over the full course of the project. Finally, velocity is the measure of how much work a team can accomplish within sprints.

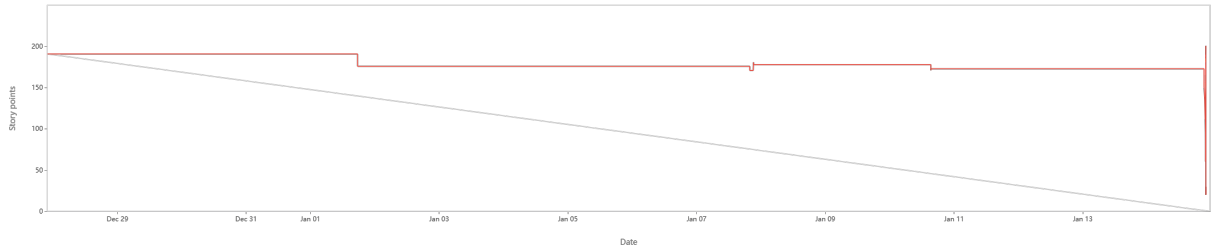
We show burndown/burnup for the sprints taking place Between December, 2023 and April, 2024. The sprints before this have been analyzed in the previous iteration of this report and are provided in the supplementary materials repository for reference. The focus here will be sprints 1.5 to 1.10 as these have not been analyzed retrospectively to date; sprints 1.1 to 1.4 have already been analyzed. Full sized version of these images can be accessed in the burndown and burnup sections of the SatOpsMQ Supplementary Materials GitHub repository.

When reading burnup reports from Jira, the vertical axis of the plot represent story points and the horizontal axis represents the data. The green lines are completed work in terms of total story points up to a given point in time. The red lines represent the scope set in sprint planning as the number of story points to be completed during a sprint. The grey lines represent the ideal burn rate to serve as a guideline for how the work should be approached over the course of a sprint. A good burnup is when the green line follows the grey line gclosely.

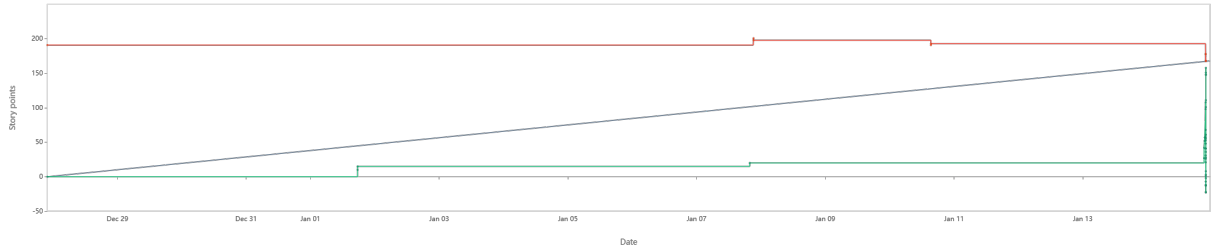
When reading burndown the vertical axis also represents number of story points and the horizontal axis represents the date. The red line represents the amount of remaining work in terms of story point number and the grey line represents the ideal burndown rate of the tasks. A good burndown is when the red line follows the grey line closely.

We will note that due to inexperience with Jira and time constraints some of the charts are not ideally generated. For example, sprints were tuned retrospectively to fix spelling, grammar and dates when mistakes were present. The free version of Jira is punishing in many aspects and this lead to some charts not being ideal. Some mistakes made were declaring all tasks as done near the end of the sprint instead of throughout, editing old sprints leading to charts displayed dates not matching the original sprint dates and ending sprints later than the actual date, which also causes date mismatches.

Sprint 1.5 burnup and burndown shows that we did not meet the ideal work rate but that we did end up finishing all the pertinent stories. During this sprint we were still revamping the project management approach and taking MVP release feedback from stakeholders in preparation for the alpha release.



(a) Burndown

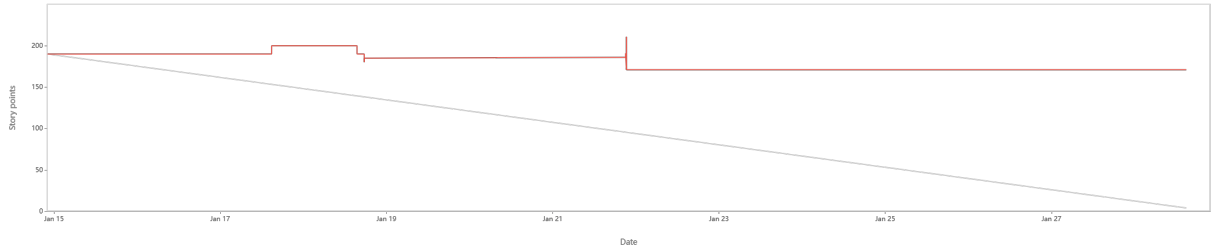


(b) Burnup

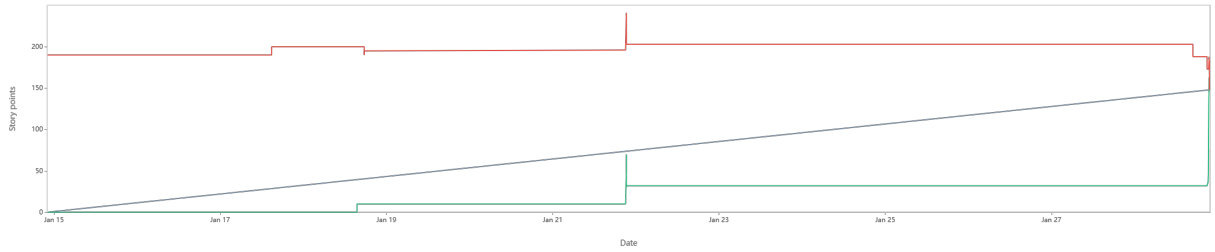
Figure 3.13: Sprint 1.5

Sprint 1.6 burnup and burndown again shows the same mistakes made as that of sprint 1.5. As well the scope was slightly adjusted in the first half of the sprint on two occurrences. It was a general trend that we do not use Jira efficiently and a likely solution for this in the future would be to have a dedicated project manager in a capstone team and not worry about everyone having to do development work as their primary role. As well mitigation steps may include having a clone of the Jira workspace so that if one workspace gets degraded due to mistakes by users there is another one to go to. A superior solution would be to use the paid version or change project management system. One good option for students is the JetBrains software called YouTrack, which is free for university students through their student program.

Sprint 1.7 burnup and burndown shows more of the same but as well, near the end of the sprint the scope was adjusted as story points were added. This made the ideal rate line cross the work scope due to the work scope increasing the red spike at the end. This



(a) Burndown



(b) Burnup

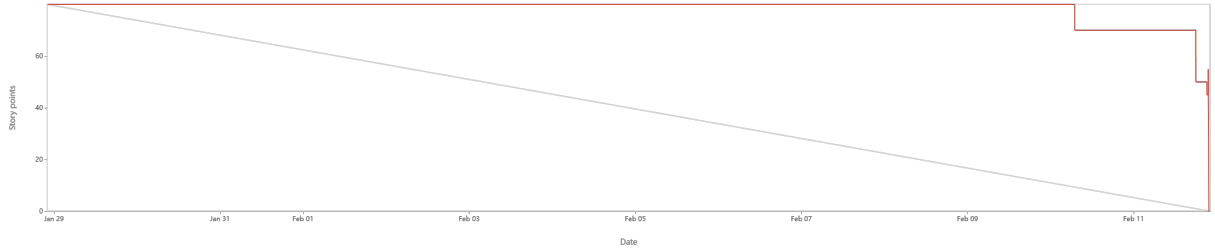
Figure 3.14: Sprint 1.6

can be seen in the burnup and burndown for 1.7. at the right most of the chart.

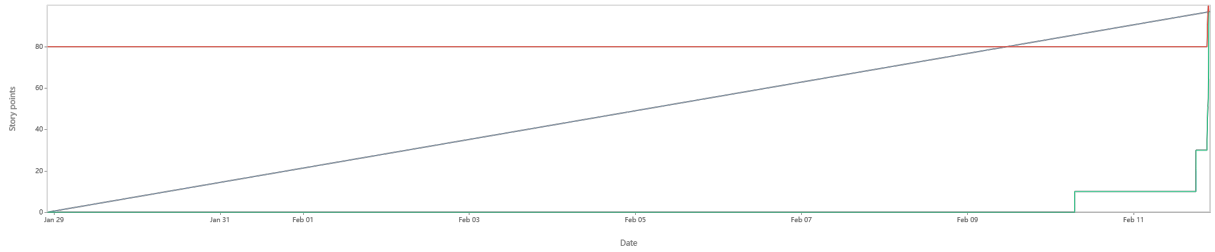
Sprint 1.8 burnup and burndown in this sprint there was a heavy story readjustment. Several stories effort was underestimated and that is why there is a large peak on both the burndown and burnup charts in the first half of this sprint. This was due to several shortcomings when attempting to calculate satellite field of view measurements properly. As well Some stories had their estimates missed and it had to be added later. Two stories went from zero points to ten and twelve points, respectively.

Sprint 1.9 burnup and burndown shows much improvement in how the project management software was used and how story points/stories were respected much greater than in previous sprints. In this sprint the items were completed regularly throughout the whole sprint and the charts show that the ideal rates are followed closely. One minor mistake was that sprint was ended later than initially planned to end. It was extended due to an oversight that the group cannot recall the reason for.

Sprint 1.10 burnup and burndown, from the final sprint, show that we went back to old habits and did not use Jira as carefully as we did in the second to last sprint. The tasks were all denoted complete at the end. As well, the sprint was completed much later than the planned completion date. The reasoning for this was not an extension of the sprint



(a) Burndown



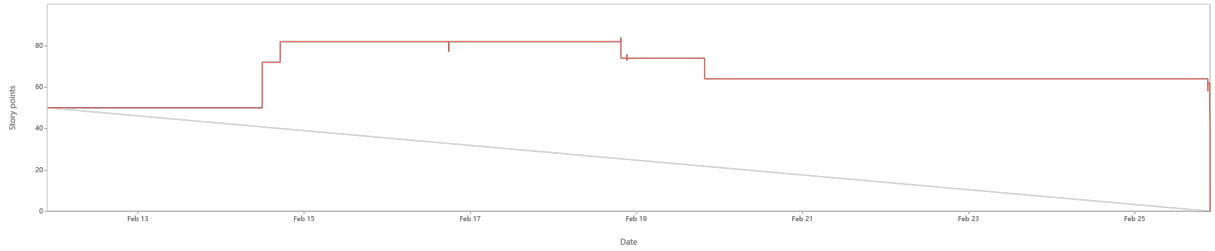
(b) Burnup

Figure 3.15: Sprint 1.7

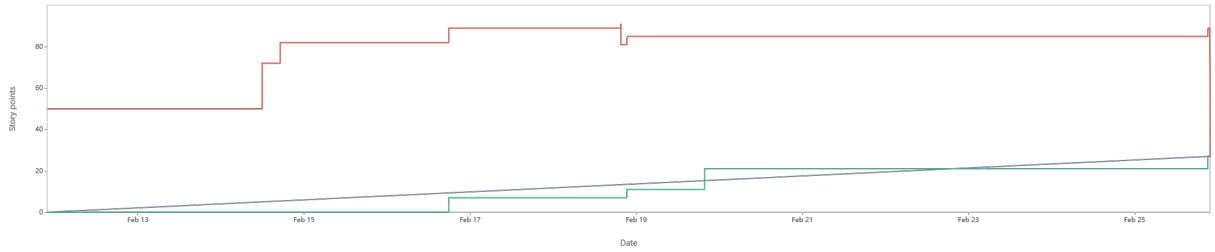
but rather just forgetting to close the sprint on the right day. During this sprint the group was admittedly overworked and time crunched. As well several external issues (such as a labor dispute at the university) played a large factor in the disruption.

The highlights of this analysis are that our team used the project management software decently but also made many mistakes when using it. This leads to the retrospective data not being the cleanest but it does allow for a very good learning experience on how to use Agile, Scum, Jira and similar software carefully. We are content that sprint 1.9 was done well. This means that the team is capable of applying Agile/Scrum but just needs to be careful to not make silly mistakes on the project management software dashboard.

We finish of this analysis by commenting on the velocity and cumulative flow. Jira only shows the velocity for the previous seven sprints. The velocity diagram for SatOpsMQ project management shows sprints 1.5 to 1.10, however it shows 1.1 instead of 1.4 due to a readjustment that was done to sprint 1.1 recently. In the velocity chart the vertical axis is story points and the horizontal axis shows the individual sprints. The grey bar shows the amount of work committed in planning a sprint measured in story points and the green line shows the amount of work completed upon conclusion of the sprint. There is no clear trend in the velocity chart. In some sprints, there was a slight underestimation and in others a



(a) Burndown



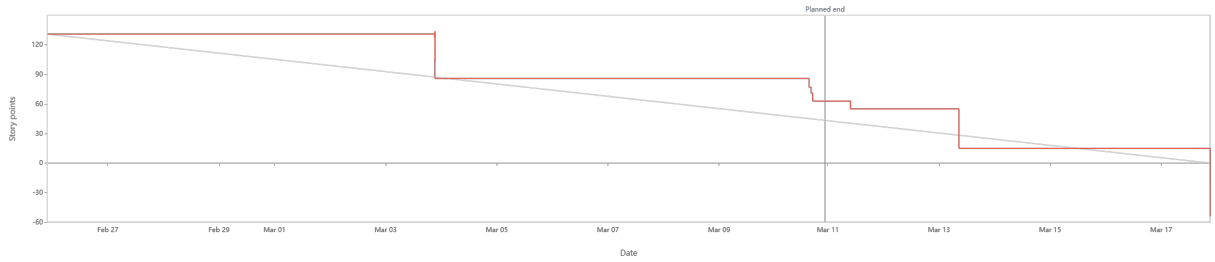
(b) Burnup

Figure 3.16: Sprint 1.8

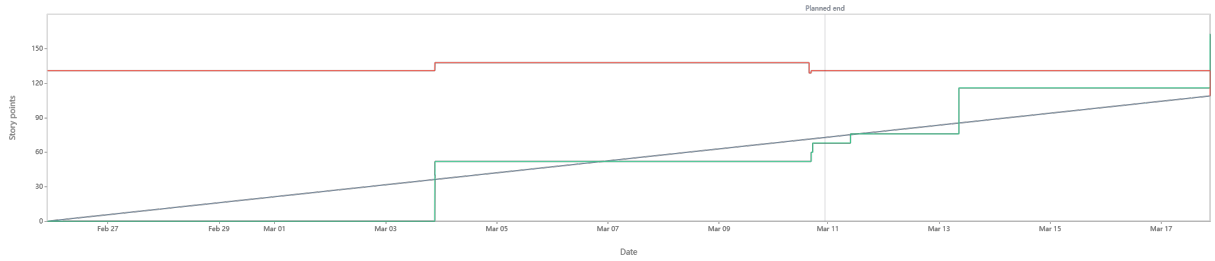
slight overestimation. As well some of the discrepancies are relatively large, showing that the team does seem to have an issue with estimating correctly. Story point estimation is no easy task and it takes years of experience to do it effectively [25].

In the cumulative flow diagram, the vertical axis is the issue count (*i.e.*, number of stories) and the horizontal axis is the time. The purple area represents items with a status of to do, the blue area represents items in progress and the green area represents items completed. This diagram's purpose is to shows how long issues take to move between the three stages. For this project we can clearly see that as time went forward more stories/features were proposed (added to the to do category in purple). As time moved forward tasks were committed to by adding them to the in progress category and then declared done. There were still some tasks left in the product backlog as previously mentioned. These were the nice to have features proposed but not pursued. Of all the tasks that were committed (blue category), by the end of the project, these tasks were completed and we see at the final month that the vast majority of tasks are finished. This implies that the group ended up achieving most of the goals pursued with some leftover goals, but more importantly no goals half finished goals left over.

Full sized version of the velocity report and cumulative flow diagram, visit the SatOpsMQ



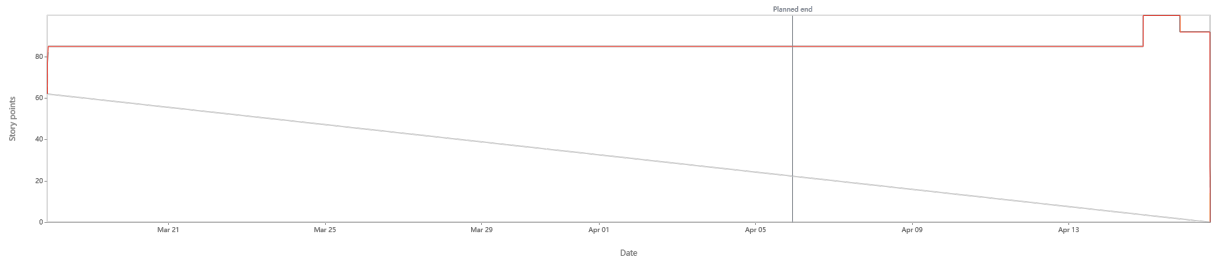
(a) Burndown



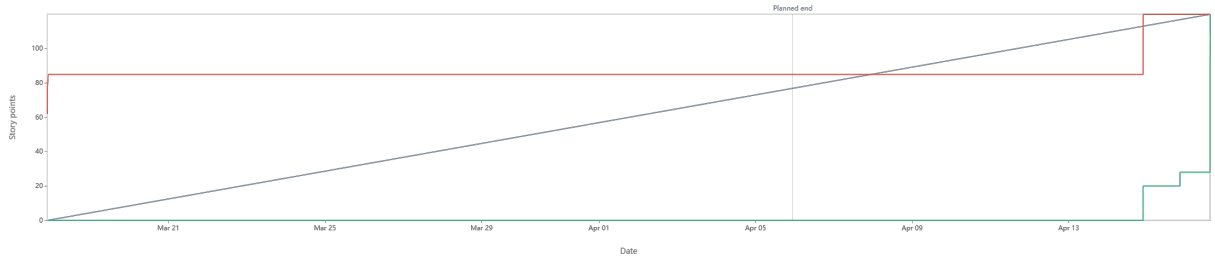
(b) Burnup

Figure 3.17: Sprint 1.9

Supplementary Material Repository



(a) Burndown



(b) Burnup

Figure 3.18: Sprint 1.10

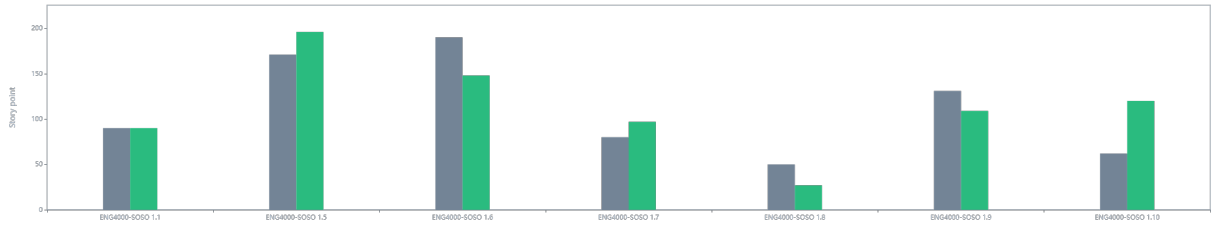


Figure 3.19: Velocity report for Sprints 1.1 and 1.5 to 1.10

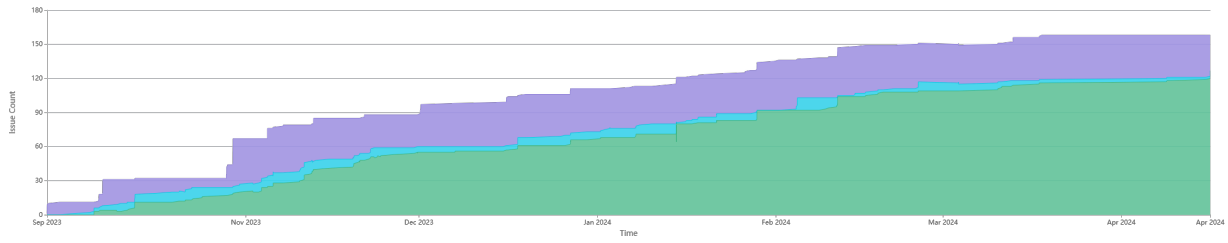


Figure 3.20: Cumulative flow diagram for the full project

3.4 Equipment & Procurement

3.4.1 Potential Realistic Financial Estimate

As mentioned earlier, there wasn't an accumulation of expenses given that the project is of an open-source nature and is developed by a team of university students. However, in this section we'll be examining the potential costs associated with a project of this scale assuming it was developed under a professional environment with hired labour and potential deployment costs.

First cost to handle is the potential labour costs. The project consisted of a team of 7 individuals, 6 of whom are software engineers, and a space engineer. The team met with the CSA (2 space engineers) for advice and input on the status of the project, as well as with the Professor supervising the project. Therefore, it took 6 software engineers, one individual acting as both a software engineer and a space engineer, along with the consultation team of 3 other space engineers with years of experience (2 from the CSA, and the supervising Professor). The average salary for a junior engineer of either software or space fields is around 80,000 CAD a year, therefore that salary will apply to the team of 7 undergraduate engineers. However, the average salary for a senior engineer with years of experience is about 200,000 CAD a year.

The labour costs will be split into 2 subsections. The first section is to handle the labour costs of developing the project, the second section is the labour costs of maintaining the project. Therefore the monthly labour cost of a single junior software or space engineer is 6700 CAD, meanwhile the monthly labour cost of a senior space engineer is 16700 CAD as shown in Table 3.3.

Position	**Monthly Cost ($)**
Junior Software	6700.00
Junior Aerospace	6700.00
Senior Aerospace	16700.00

Table 3.3: Monthly Labour Cost Based On Position

The project development took around 6 months of part time work, condensed into 3 months assuming full-time. 7 individuals worked full time, meanwhile the 3 senior engi-

neers worked part-time given that they consulted once at a weekly basis. Therefore the breakdown is as shown in Table 3.4.

Position	**Count**	**Months**	**Total Cost ($CAD)**
Junior Software Engineer	6	3	120,600.00
Junior Space Engineer	1	3	20,100.00
Senior Space Engineer	3	1	50,100.00
Total			190,800.00

Table 3.4: Labour Costs of Project Development. Months are full-time work.

After project development, the project needs to be continuously maintained. Therefore, we only need around the development team excluding the consultation team. Which leaves us with 7 junior engineers. Resulting in a maintenance labour cost of 560,000 CAD a year, as shown in Table 3.5

Position	**Count**	**Total Cost ($CAD)**
Junior Software Engineer	6	480,000.00
Junior Space Engineer	1	80,000.00
Total		560,000.00

Table 3.5: Yearly Labour Costs of Project Maintenance.

This concludes the labour costs associated with the project. Moving on to the deployment costs associated with the system. The entire system will be hosted on the amazon EC2 platform. The amazon EC2 platform is highly customizable and comes in a variety of options/variances, therefore it comes with an online cost estimator. We have set up a table outlining the parameters picked out and the resulting costs outputted from the cost estimator of amazon's EC2. The resulting monthly cost of the system is 1239.84 USD as outlined in Table 2.7.

This concludes the partial costs of every aspect of the budgest. Below you'll find the total costs associated with the entire project. The total cost for an entire year is 772,878.08

Amazon EC2	
Param Type	Selected Config
Workloads	Constant Usage
EC2 Instance Type	t3a.small
Payment Option	On Demand; with 100% usage
Elastic Block Store	
Storage Type	EC2
Throughput Value	1000 MBps
Storage Amount	100 GB
Snapshot Freq	Daily
Amount Changed	50 GB
Detailed Monitoring	
Enabled	True
Data Transfers	
Outbound Transfer	100 GB per month to other regions
Summary	
Total	1239.84 & per month

Table 3.6: Deployment Options/Params for Amazon EC2

(as shown in Table 2.8).

Budget Type	Cost ($CAD)
Development Labour Costs	198,000.00
Maintenance Labour Costs	560,000.00
Deployment Costs	14,878.08
Total	**772,878.08**

Table 3.7: Total Project Costs

3.4.2 Expenses Summary

In summary, if the capstone project was developed in a professional environment where aerospace and software engineers were hired and deployment costs were taken into account. The total cost of the project would have been 772,878.08 CAD.

However, for the capstone project where we assume it is an open-source project. We don't have many expenses in our cost sheet, the only costs that we incurred were through professional software used to document our project like Overleaf Pro and presentations through the use of Canva Pro.

Cost Type	Cost ($CAD)
Overleaf Pro	20
Canva Pro	66.45
Total	**86.45**

Table 3.8: Actual Project Costs

When we compare the actual costs of the project to how much the project would potentially cost. We can see that we saved the our stakeholder roughly 772,791.63 CAD had this project been done professionally with aerospace and software engineers.

3.4.3 Equipment Disposition

For the equipment disposition, we linked our bill of materials to list out all of the tools used for the project: Bill of Materials (BOM).

3.5 Business Case

In this section, we'll be handling the business aspects of our project. However, as we will explore later in details, there is no financial gain/loss to this specific project given it's open source nature. Mainly, We'll be examining the aspects of this system that make it a potential business product, we'll be performing a SWOT analysis to see how it stands in the market, and finally we'll be examining other aspects of the project pertaining to it's open source nature.

3.5.1 System as a Service

The project at hand, SatOpsMQ, is a satellite scheduling system, responsible for automating and refining satellite schedules to handle image orders. Currently, the project is sponsored by the CSA, to handle/address the issues arising with manual scheduling of such image orders. Based on that alone, SatOpsMQ has a standing as a product or a service that can satisfy a need currently improperly fulfilled within the space market. Otherwise, a system like that would not have been requested by a large influential entity as the Canadian Space Agency.

SatOpsMQ aims to tackle multiple stakeholder requirements including but not limited to the automated process of scheduling image orders. The current suite of programs dealing with image order scheduling are heavily manually operated and/or contain a minor aspect of automation that isn't sufficient or effective enough to improve the entire workflow. SatOpsMQ aims to address that gap in the software suite. Not only that, but SatOpsMQ aims to achieve a balance between automation and manual planning, as it allows an operator to alter an automated schedule. This harmony between manual and automatic operation creates a golden feature that makes SatOpsMQ a viable product/service.

However, SatOpsMQ as mentioned earlier is an open-source project therefore the potential impact it has as such is much greater than if it were a paid for product or service. This impact will be discussed in more detail in section 3.4.3

3.5.2 SWOT Analysis

In the earlier section, we've examined the potential for SatOpsMQ to become a viable product or service. We will elaborate more on this through the SWOT Analysis. However, we will be performing the analysis with SatOpsMQ as an open-source product and not a paid/propietary product.

Quick overview of the SWOT analysis; we'll be first discussing the strengths of the system, then the weaknesses, afterwards we'll examine how each strength or weakness opens up a potential for an opportunity, and finally we'll conclude with the possible threats that may arise.

3.5.2.1 Strengths

As mentioned in the earlier section, the biggest strength of SatOpsMQ lies in it's balance between automation and manual scheduling, where an operator is provided the ease of automated scheduling paired with the ease to adjust any schedules that are formed. This is a vital feature not properly implemented within the market due to a variety of reasons. Most space-software programs out there are proprietary black box systems, therefore the capabilities of expanding the program and understanding how it functions is unknown. Most systems in the market are setup to help with manual scheduling and not automated scheduling, this can be observed through the slow functioning of the programs as well as the primitive UI interfaces provided, more information on the competition can be found in section 1.6.

SatOpsMQ on the other hand is a white-box system that is open source, therefore is not proprietary. This opens up for many possibilities of advancing satellite system scheduling given that anyone with enough expertise can build up on the current code. SatOpsMQ has also been built with modularity in mind, allowing certain components to be swapped out if needed.

3.5.2.2 Weaknesses

The system is heavily reliant on image order input, which is set up and inputted by individuals, which opens up a human error issue. The system's main focus is creating schedules based on image orders received, therefore if faulty image orders were received, the outputs of the system may irregular.

The system has gone through extensive software testing and expert-input based testing (details about testing are available in Section 2.5.6 Testing). However, it hasn't been tested with an actual satellite, instead mock satellite TLE data was used to track satellite movement and in the creation of image schedules. Therefore, the system has not been tested in a real-life scenario with actual satellites and ground stations.

3.5.2.3 Opportunities

The system's current state opens it up to a wide array of opportunities. SatOpsMQ is an open source project therefore it can allow, in the near future, the formation of a community of collaborators that can help accelerate the development of the system, as well as quickly address any flaws/weaknesses affecting it.

SatOpsMQ is also a very modular system, therefore any components that don't seem to address the current user's expectations, can be quickly swapped out to do so. Therefore, any future breakthroughs in technology that relate to any of the components of the system, can be quickly integrated and tested to help further modernize the system.

Finally, SatOpsMQ is CSA-sponsored, therefore this helps greatly when it comes to demonstrating the importance and relevance of the system in any given context. Whether it'll be to gather up a community of contributors, or to spread awareness of its existence and its capabilities to address the problem at hand.

3.5.2.4 Threats

SatOpsMQ is currently maintained by our team of university students for the duration of the capstone course. Therefore, the potential for the system to become stale/deprecated is high if a community isn't built around it soon enough.

3.5.3 Open-source Aspects

In this section, we'll briefly discuss certain aspects of the system's open source nature that we find relevant to mention as part of the business case. We'll open up with the discussion of how to foster a community of contributors, discuss the keep up of documentation relevant to the system, and finally conclude with countering certain challenges brought forth from large companies.

3.5.3.1 Community Development

A vital part of every open-source project/software is the community of users and contributors that could help further advance the system for the better. This is a vital aspect of SatOpsMQ that needs to be grown in order to ensure its survival moving forward.

We can leverage the CSA's standing within the field to gain attention and use that to slowly start building a community of supporters. We can also focus on advertising the capabilities of SatOpsMQ (with the help of the CSA as well) to have companies begin using it within their systems.

3.5.3.2 Documentation Keep Up

SatOpsMQ is currently maintained by a team of University students and therefore the existence of formalized/accessible docs, that the team of contributors could potentially access and use as a guideline for future development, hasn't been established. Aside from this report, and the auto-generated documentation of the various endpoints through FastApi, there is no documentation to outline a specific structure of architecture to follow, or a specific code style/formatting pattern to stick to.

Therefore, a more refined and tailored set of documentation that could add more than this report outlines, and can guide developers to following a strict set of development patterns to use as not to negatively affect the performance/features of this system needs to be set up. A common platform for specifying/listing these details can either be through the Github Docs tab, or through the use of Atlassian's Confluence platform.

3.5.3.3 Takeovers

Given that SatOpsMQ fulfills a current gap in the market, through the implementation of automated features of satellite scheduling coupled with it's open source state, could possibly lead to a takeover from a large company. For instance, a company may try to patent certain parts of the code and adapt it into their own proprietary systems which may then slowly lead us back to the initial set of proprietary systems issues we've discussed earlier.

Therefore, a strong community backed by an ethical strong company could very well help maintain and protect this open-source project from possible legal loopholes of takeovers or theft.

3.6 Risk and Quality Management

We close this chapter with a discussion on the risk assessment plan and quality management plan. We assessed all the possible risk to the project during the project life cycle and developed likelihood vs. impact matrix to prioritize risks based on their estimated likelihood and perceived impact.

The quality management plan comes in the form of our risk mitigation and recovery plans that outline steps that will be taken to lessen the impact/likelihood of the risk occurring and a methodology to re-gain some of what was lost from the occurrence of the risk respectively.

3.6.1 Risk Management Plan

The risk assessment matrix describes the types of risks that we may face in the project because of realistic gaps and issues that we'll encounter and how to prevent and remedy those risks. A detailed risk response plan can be found at this link.

3.6.2 Quality Assurance Strategies

To expand on the quality assurance strategies employed on the potential risks that could occur in the product development, we assigned team members to each possible risk and outlined the recovery process that would be conducted to ensure minimal possible loss to the product.

Furthermore, we elaborated on the risk mitigation type used on each risk as follows [29]:

1. Risk Acceptance: This means acknowledging the risk and its consequences, without taking any action to change it. This is suitable for small or unlikely risks, or when the cost of mitigation or avoidance is higher than the risk itself.

2. Risk Avoidance: This means not doing the activity that causes the risk. This is preferable for large or unacceptable risks, or when the risk outweighs the benefits of the activity.

3. Risk Mitigation: This means managing the risk by reducing its probability or impact. This is applicable for moderate or controllable risks, or when the benefits of the activity justify the risk.

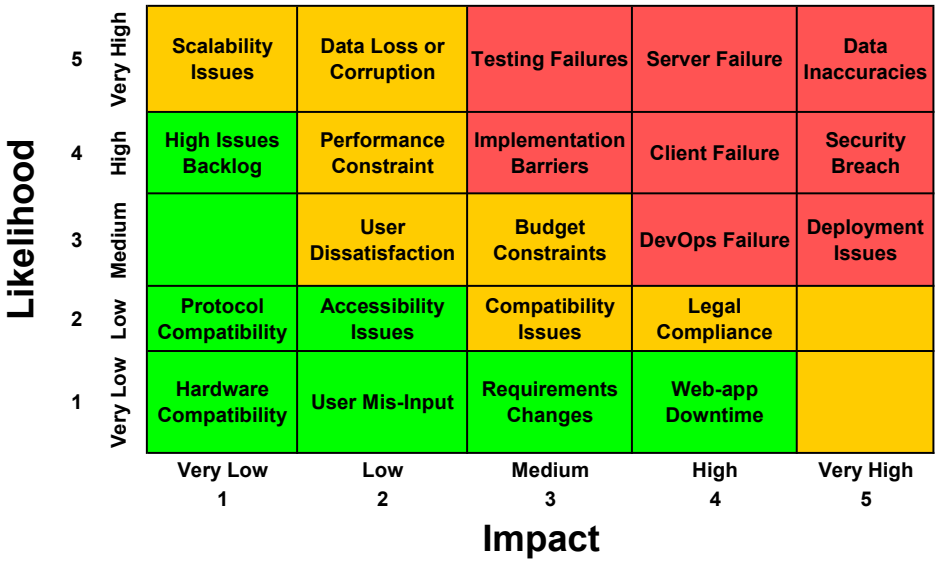


Figure 3.21: Risk Matrix showing likelihood versus impact of various risks in the development process.

4. Risk Reduction: This means lowering the risk to an acceptable level, which is called the residual risk level. This is common for most risks, as there is usually a way to reduce them. This involves taking action to minimize the negative effects of the risk. For example, buying insurance is a form of risk reduction.

5. Risk Transfer: This means shifting the risk to another party or entity. This can be done by outsourcing, insuring, or leasing. This does not necessarily lower the cost, but it can reduce the potential damage. This is useful for high or unpredictable risks, or when the risk is beyond the control of the party.

Risk	Description of Risk	Likelihood	Impact	Probability-Impact Index (PPI)	Mitigation	Mitigation Type	Recovery	Team Member Responsible
Data Inaccuracies	Failure to process input data correctly; therefore, leading to inaccurate output for the data which defeats the purpose of the project.	5	5	25	Refer to CSA resources to validate results and develop safety net protocol to detect anomalies.	Risk Mitigation	Refer back to commitments backlog for changes actively.	All Team Members
Server Failure	The server for the web-app fails to load.	5	4	20	Test sequentially different parts of the back-end to ensure calculations and other parameters work and display accurate data.	Risk Mitigation	Analyze parts that have issues to recover the back-end server for the web-app.	Youssef/Rafael/Stanley/Hashir
Security Breach	Unauthorized access or use of the open-source web-app.	4	5	20	Include specific licensing agreements compatible with typical open-source projects.	Risk Mitigation	Notify stakeholders to understand how to proceed with the issue or revoke access to project.	Hashir/Youssef
Client Failure	The client for the web-app fails to load the front-end.	4	4	16	Test sequentially the UI and other components for the front-end deployment.	Risk Mitigation	Analyze failures and ensure front-end is operational for users after recovery.	Rafael
Deployment Issues	Database failures/Cloud computing errors when deploying the web app. For example, cloud usage overage.	3	5	15	Track performance of the hosting platform and maintain local copy of records.	Risk Reduction	Deploy on another hosting platform based on alternatives chosen.	Hashir/Rafael
Testing Failures	Encompasses all possible failures that can occur when testing the product and using test plans.	5	3	15	Create feasible and comprehensive test plans that will minimize failure.	Risk Avoidance	Backtrack on failed tests and understand root cause of the problem.	All Team Members
Implementation Barriers	Knowledge short-comings and Imbalanced workloads on sprints.	4	3	12	Acknowledge team weaknesses and support any issues internally.	Risk Acceptance	Contact CSA or Supervisor and re-define workload.	Hashir/Youssef
DevOps Failure	The integration between the front-end and back-end hosting fails.	3	4	12	Create a test plan that can account for possible failures ahead of deployment.	Risk Acceptance	Do separate test plans on the front-end and back-end to better understand root cause.	All Team Members
Data Loss or Corruption	Certain data is lost from schedules or specific satellites or ground stations such as timestamps, access points, etc.	5	2	10	Use testing plan for the back-end calculations to ensure minimual or no data is lost in the process of generating schedules or satellite status or ground station health.	Risk Reduction	Check the database and functions used to output data to validate which data was lost.	Stanley/Ruth
Budget Constraints	Lack of funding.	3	3	9	Actively monitor budget management tool for the project.	Risk Mitigation	Contact ENG4K supervisors and supervisor for project for support.	Hashir/Youssef
Performance Constraint	The performance of the system takes a long time to execute.	4	2	8	Ensure that the load balancing between the different services doesn't deteriorate the performance of the system.	Risk Mitigation	Analyze load time for services used in the architecture solution.	Rafael/Hashir
Legal Compliance	Project doesn't fall outside legal boundaries of open-source project.	2	4	8	Ensure that the project's purpose is stated and licensed appropriately for an open-source project to the public.	Risk Mitigation	No recovery plan. Project will be compliant legally.	All Team Members
User Dissatisfaction	The users of the web-app are not satisfied with certain parts of the application.	3	2	6	Create open communication between all users of the web-app to ensure all concerns could be addressed.	Risk Avoidance	Address concerns made by the user to better understand severity of issues.	James Le/Youssef
Compatibility Issues	Issues that may occur when integrating front-end and back-end of the codebase.	2	3	6	Test individual components of front-end and back-end to ensure there are no issues between integration.	Risk Mitigation	Recover specifc components from front-end and back-end to view and understand issues.	All Team Members

Figure 3.22: Detailed Risk Assessment Plan with Probability-Impact Index (PPI) for SatOpsMQ, Page 1.

3.6.3 Change Management Process

As we are composed of a team of 7 members working on the product's development. Our change management process is as follows:

1. The team uses Jira software to manage the project and track changes.

2. The team creates feature branches for each new feature or fix, ensuring the stability of the main code-base.

3. The team initiates merge requests and pull requests to review and discuss the changes

Scalability Issues	The web-app is not scalable to the degree that is expected by the stakeholders.	1	5	5	Ensure that the team reads the appropriate literature and trade study to create the most optimal architectural solution.	Risk Mitigation	Configure some parts of the architectural solution. Otherwise, hard to recover.	Rafael/Stanley
High Issues Backlog	Transfer of many issues from one sprint to the next creating a large backlog of issues.	4	1	4	Use stand-up meetings and work sessions to ensure work is being completed for the issues being addressed in the project.	Risk Reduction	Only recoverable is to push more issues on active team members to keep up with the work.	Rafael/Youssef
Accessibility Issues	Accessibility features do not work on the web-app for the front-end.	2	2	4	The colors used for the front-end will be color-blind friendly front-end.	Risk Mitigation	User feedback from the product will drive the team to ensure it accommodates all users of the web-app.	Rafael
Web-app Downtime	After deployment of web-app to the public, web-app may need some server maintence to clean up the data.	1	4	4	Create open communication between all the users of the web-app and the developers to let users know of any maintenance that would occur.	Risk Acceptance	Ensure that data is actively scrubbed and the website is operational for the users.	All Team Members
Requirements Changes	Sponsor changes objectives/goal of the project.	1	3	3	Actively communicate with Sponsor to ensure the project is on track.	Risk Acceptance	Create backlog tasks to address new requirements.	Youssef/Ruth/Walid
Protocol Compatibility	Incompatibilties with the different web protocols used on the web-app.	2	1	2	Test & Deploy an initial set of web protocols such as File Transfer Protocol (FTP) for JSON files of TLEs and confirm that they work accurately.	Risk Mitigation	Use test plan to understand failures and root cause to ensure it doesn't occur again.	Hashir/Walid
User Mis-Input	If the user inputs a file for a satellite, or enters an incorrect data format for the ground station. There will be an error on the front-end.	1	2	2	Develop the front-end such that it is intuitive for the user to understand which data entry to use.	Risk Mitigation	Error message will inform the user about the mistake and the team can assist the user.	All Team Members
Hardware Compatibility	The web-app can't interface accurately with hardware/real satellite or ground stations.	1	1	1	Priority of project is to simulate satellite and ground station communication. Interfacing with hardware is optional but will provide relatively decent results but won't take into account grandeur elements of hardware.	Risk Acceptance	Develop plan that will further define the hardware needs if needed by specific users.	Youssef/Rafael

Figure 3.23: Detailed Risk Assessment Plan with Probability-Impact Index (PPI) for SatOpsMQ.

before integrating them into the main branch.

4. The team plans to use Jenkins to automate the build, test, and deployment stages, creating an efficient pipeline that enhances productivity.

5. The team also plans to use Discord for pipeline monitoring, providing real-time updates and alerts on the development process.

3.6.4 Review of Risk Management Process For All Releases

As we have completed the project, we begin to analyze the risk matrix developed in the beginning and our change management process to address each risk during the project life-cycle.

3.6.4.1 MVP Risk Occurance

For the minimum viable product, we only had one risk that occurred which was the hardware compatibility between the developers systems. This was caused to a reason outside of what was described for the risk. It was because the developers had different packages and different versions for each package that lead to hardware compability issues between the developers when running the web application.

3.6.4.2 Alpha Release Risk Occurance

For the alpha release, we had two risks that occurred at such as:

- Requirements Changes: The CSA noted that they wanted to add more requirements for the Scheduler algorithm that processes and schedules the user orders. At alpha release, our deterministic algorithm was able to compute schedules but not perturbate the schedules fast enough as required by the user. As part of the risk acceptance strategy, we added user stories in our future sprints to address this.
- DevOps Failure: The team was unable to integrate the basic skeleton of the back-end servers with the front-end client which lead to some unit testing needed to further validate each component.

3.6.4.3 Beta Release Risk Occurance

For the beta release, we also faced more risks that occurred as we were developing the web-application.

- Data Inaccurancies: We noticed that when scheduling orders in the gantt chart format in our schedules that they were offset by roughly 30 minutes or 6 hours everything and we were able to narrow down the reason for this change. It was because the datetime parameter used in our code that would read and translate the UTC time wasn't coded properly and had to be refined.

- Client Failure: The team faced an issue where after deploying the web application, all other pages would load a perform accurately. However, the web page for the schedules wouldn't load up and we were able to attribute the cause of this issue to the micro-service for Event-Relay API not running. We were able to fix this issue by running the Event-Relay API after deploying the front-end client.

3.6.4.4 Final Release Risk Occurance

For the final release, we faced more risks that were worth noting such as:

- Testing Failures: Closer to the end of the testing phase for the final release, the CSA noted that there was an issue with the edge case testing where the assets would not be equally used to process user orders. We were able to address this risk by improving the scheduling algorithm to distribute the orders in the schedule to the satellites more evenly.
- User Dissatisfaction: When analyzing the acceptance of image orders that were being processed in the system, the CSA noticed that there was a high acceptance rate for images that should not be able to be processed by any of the satellites. This made the team revisit the code-base to improve the field-of-view (FOV) ground tracking code to ensure that it outputted a realistic result of the FOV based on what the CSA expects.

3.7 Evaluation of Management Pack Performance

Similarly to the evaluation of the technical pack performance in chapter 2, we will look at the self-evaluation for the management pack. The components below are for the rubric. As well, all evaluations are available in Appendix D.2 Self Evaluation

The next table includes the components for the list of sections mentioned in the report outline.

Criterion	Ranking	Justification
Develop Agile Roadmap	Exceeding	Reference1
Breakdown, Prioritize Product Features Into Increments	Exceeding	Reference1, Reference2, Reference3
Assign, Track, Asses Team Tasks & Performance	Exceeding	Reference1, Reference2, Reference3, Reference4, Reference5, Reference6, Reference7, Reference8
Review Sprint Completion Status	Exceeding	Reference1, Reference2
Review Sprint Effectiveness	Exceeding	Reference1, Reference2, Reference3, Reference4
Ensure Appropriateness of Sprint Plan	Exceeding	Reference1 Reference2, Reference3,
Review Project Costs	Exceeding	Reference1, Reference2
Obtain Stakeholder Feedback on Releases To Inform Next Development	Exceeding	Reference1, Reference2, Reference3

Table 3.9: Self-Evaluation For Agile Project Management Section

Criterion	Ranking	Justification
Project Schedule	Exceeding	Reference1
Sprint Plans	Exceeding	Reference1
Final Project Costs	Exceeding	Reference1
Preliminary Business Case	Exceeding	Reference1
Lessons Learned/Failure Report	Exceeding	Reference1, Reference2

Table 3.10: Self-Evaluation For Agile Project Management Section Delvierable List List

Chapter 4

Lessons Learned Volume

In this volume, we describe the insights and lessons gathered throughout the lifecycle of the SatOpsMQ project. We reflect on deviations from the initial plan, notable failures encountered, overarching lessons, project phase retrospectives, and future recommendations.

4.1 Deviations From Planned Activities

The solutions to the problem are rigid in terms of what the system is and should be capable of, therefore there was not much space for deviation. However, there were changes in how our solution could be presented, what technologies to use, and how to make it efficient without compromising the desired function.

In the beginning phase of the project, we planned to use AWS lambda for our web-app. This allowed for easy scalability managed by the provider without having to worry about it during development. A server-less system would have also made it easy to test the performance of the system. However, this route did not match the sponsors' requirement that the system should only use open-source software and force them into a vendor lock-in. Hence we decided to use docker containers instead which provide much of the same benefits but allow users to host the system wherever they want. Around the end of the project, we also decided to have an executable desktop app that would allow for an easier setup, and offer flexibility in the way the customer wants to use the final product.

Throughout the development cycle, one component that kept evolving was the database schema. It was initially designed to represent the data provided by the sponsors and the relationship between them. This worked well at first but as we delved deeper into the

requirements and our understanding of missions grew, the schema continued to be updated. We later made a complete redesign to make it more efficient to access commonly used data as well as add tables to cover newly added requirements. The final design is now flexible in the types of orders it can store and accurately represents the relationships between the tables.

The last major change was in the way that individual orders were scheduled. The current system used by our sponsor is not capable of adding new orders as they come in but instead, it reschedules all existing orders. This was a significant problem to be addressed since it would take the satellite operators about an afternoon to schedule orders. Therefore we planned for the system to accommodate perturbations by scheduling individual orders wherever an available time slot was found. However, after a review of the solution, we concluded that the root cause of the problem was the time it took to reschedule the orders. Our system takes minutes to do so while just assigning an empty slot compromised the performance metrics of the schedule. Ultimately we decided that having a high performance while also significantly reducing scheduling time was better than a slightly faster scheduling that did not meet our performance goals.

4.2 Project Failures

Throughout the project, maintaining frequent communication with our sponsors proved invaluable. This allowed us to address issues and implement their feedback, ensuring the system's accuracy and reliability. Several critical issues concerning the accuracy of calculations for eclipses, contact opportunities, and satellite field of view were identified and promptly resolved. Left unattended, these issues could have led the system to produce faulty results.

However, some edge cases were not accounted for in the final release. As mentioned in the section detailing deviation from planned activities, our system's approach to rescheduling orders posed challenges. Rather than finding available time slots within an existing schedule when new orders arrive, our system reschedules all orders whenever new orders come in. It does this instead of finding available time slots within an already existing schedule. This, on its own, is not a problem as it was discussed and approved by our sponsors. However, we took the same approach when rescheduling an order that had to be moved because of an outage occurring at the assigned ground station. When this happens the order can either not be sent to the satellite (uplinked) or downloaded from the satellite (downlinked) back to the ground station as the station is in outage. Therefore, when an

outage occurs the only affected uplink or downlink is required to be re-assigned to a different ground station. That is not what the system does. Instead, it rescheduled the whole order resulting in the unnecessary assignment of imaging time as well as the downlink or uplink contact that had not been affected by the outage. This wastes resources on redoing computations, especially in instances where the station in outage has a large number of orders to be rescheduled.

Another edge case was during an ongoing maintenance activity on a satellite, it could not capture an image if the maintenance activity's "payload outage" field was True. This field is used to determine whether an image order can occur at the same time maintenance is happening. In the current implementation of the system maintenance and image orders cannot overlap at all. While this ensures that no image orders are falsely assumed to be processed if they are not, it is too strict of a restriction and does not allow images to take place when "payload outage" is set to true or image capture is permitted. Therefore it removes valid time-slots away from possible orders and potentially results in a schedule with lower performance metrics.

Both of these failures were due to overlooking details in the implementation process. Even though they were discussed and documented initially it is easy to miss details as time goes by. Therefore they could have been avoided by revisiting the requirements and constraints when implementing each feature.

4.3 Lessons Learned

4.3.1 Project Phases Retrospective

The following sections are a reflection on how we should have in hindsight run each of the project phases differently.

4.3.1.1 Minimum Viable Product Phase

We recognize several crucial adjustments that could have improved the outcomes of the project during the MVP phase. During the beginning stages of the project, we dove into development prematurely. While our enthusiasm for delivering a complete product helped us produce our current system, it led us to overlook critical details, such as identifying edge cases for certain system properties. Unfortunately, this oversight came back to haunt us later on, as we found that some of our initial designs were incompatible with newer

requirements. Consequently, we learned the importance of dedicating time to ironing out requirements and regularly revisiting them. This lesson is invaluable and one we'll carry forward into future projects.

Another one of these improvements includes starting the development of the scheduling algorithm earlier. This would be important considering the significant time and effort it took in the following phases. This early start would have ensured more thorough testing and integration, ensuring smoother progress in subsequent phases. Additionally, investing more time to refine the front-end design to enhance user-friendliness would have been beneficial, given that the initial design established in this phase has remained largely unchanged. Furthermore, ensuring that the code base was well documented and structured would have contributed to laying a solid foundation for the project's progression, minimizing confusion for the next stages of the project.

4.3.1.2 Alpha Release Phase

Reflecting on the alpha release, which primarily focused on testing, several key areas for improvement emerge. Initially, prioritizing the connection between the backend and frontend of the project should have been a priority to ensure the establishment of an end-to-end system. This collaboration was crucial for testing, as without it, certain types of testing, including end-to-end testing, could not have been implemented. Unfortunately, only unit testing was conducted at this stage, whereas a more comprehensive approach, including unit, integration, and end-to-end testing, would have been ideal. Establishing communication and coordination efforts earlier in the project could have helped align frontend and backend efforts, preventing inefficiencies and complications associated with connecting components late in the project.

Furthermore, creating a proper testing report with input from all team members would have been essential. As individuals progressed in their respective tasks, collaborating on a testing plan would have allowed for a more thorough approach to quality assurance. By collectively designing testing criteria and responsibilities, the team could have ensured that all aspects of the system were thoroughly evaluated, reducing the risk of overlooked issues and improving the quality of the product.

4.3.1.3 Beta Release Phase

The beta release phase, like the Alpha release, focused a lot on testing. In the beginning, it was difficult to test end to end since the scheduler had not been fully integrated into the

system. It had been implemented separately as it was more important to figure out how it should work without having to worry about the other components it needs to interact with. This allowed us to complete it faster at the cost of having a longer integration process. On the other hand, it would have been beneficial if the scheduler was divided into and implemented as smaller functionalities so they could be integrated and tested in parallel. However, this also would have made it difficult to have a working scheduler in the short time that it took. If we could re-run this phase we would have split the scheduler into smaller chunks and planned for their testing and integration ahead of time.

4.3.1.4 Final Release Phase

The final release of the system saw updates to the front-end design and an addition of order visualization for each satellite. These changes were implemented according to sprint plans and completed on time. However, one opportunity for improvement would be frequently showing the CSA the progress of the newly added features. Although we regularly met and shared our updates, receiving their inputs more often when planning and implementing the UI components would have been great.

During this phase, we received a set of sample orders that helped us discover bugs and address some feedback relating to the produced schedules. While these were incredibly valuable, fixing the issues required more time than was initially allotted showing us that we should have been prepared for unplanned tasks. Going forward it would be useful to schedule some story points for additional tasks.

After we received the sample orders, we also generated our own orders for further testing. These proved to be incredibly useful as we discovered bugs and made improvements to the code. Having these orders earlier would have made issues in the final product be resolved sooner. Therefore having pseudo-randomly generated data to test with in the as soon as the scheduling algorithm was finished would have been useful in revealing blind spots that were missed by the planned data we used initially.

4.3.2 Future Recommendations

An area of improvement lies within project management, particularly in planning for releases. Although we set goals for each release, our focus leaned more toward task completion rather than delivering specific features to the users. Upon discussion with the course directors, we came to realize that our project had requirements resembling a waterfall model but could be approached in an agile manner. Consequently, our releases blurred their

boundaries, emphasizing the continuous development toward the desired final product. It would have been more effective to prioritize and highlight the features to be delivered in each release, in addition to the overarching final product.

Finally, a very important lesson that we took is to have well thought out reasons for our design decisions. This can be applied to any part of the project but was very apparent in choosing the system architecture. Microservices architecture is a very useful design principle and works well for Python backends, however, we used developer compartmentalization as a reason for microservices to be built. This is not a good reason to justify microservices architecture and we ended up having separate services that did not need to be separate. What we should have done was analyze the need of our system and all the required functions of each service if it indeed needs to stand alone or be merged with another one.

4.4 Team Members, Contributions & Reflections

In this section, we highlight in detail the composition of the team, their backgrounds, their contributions, their training, and their performance reviews. The descriptions in the subsequent sections are not written by any team member in isolation, rather these were written and edited as a group to ensure that they are free of bias. The individual peer review reflections are written from a first-person perspective. The peer reviews were completed anonymously using the ITP Metrics web application. All result reports can be found at SatOpsMQ ITP Reports.

The project team is composed of seven dedicated individuals, each bringing their unique perspective and motivation. Six members are software engineering students, driven by interest in developing a web-app tailored to the demands of satellite operations. The collective goal is to harness collective technical knowledge, building a tool that is not only functional but also scalable and user-friendly. Among the group is a space engineering student, whose passion for aerospace adds a depth of understanding to the team's approach. This member ensures that the software solutions align with the real-world needs of satellite operations. Together, this blend of software prowess and space engineering insight propels the team forward, each member motivated by the aspiration to contribute meaningfully to the field.

The roles outlined below serve as a foundational guide for the project's progression. However, these assignments are not rigid; team members are encouraged to collaborate across roles and assist in various capacities to ensure the efficient and timely completion of

the project. Below we outline each team member's background, contributions, tasks, and responsibilities.

Name	Role
Walid Al-Dari	Full Stack Developer
Ruth Bezabeh	Backend Developer
Rafael Dolores	Software Architect & Full Stack Developer
Youssef Hany	Systems Engineer
Stanley Ihesiulo	Senior Backend Developer
Hashir Jamil	Full Stack Developer & Database Architect
James Le	Frontend Developer

4.4.1 Walid Al-Dari

4.4.1.1 Personal Statement

Walid aimed to get his hands dirty with the Space industry through this project. The project contains various aspects pertaining to satellite operations which make the solution finding intriguing and worthwhile, which has fulfilled Walid's need to problem solve.

Walid has front-end experience as a web developer through various internships and projects. Walid had minimal industry experience in backend development, feeling very capable of this task, He contiributed to the project has heavy backend components which have added to Walid's backend experience.

4.4.1.2 Contribution Details

Walid has been responsible for handling backend tasks focused on the flow of specific data categories, he has also been present for frontend development tasks as he is knowledgeable in this domain. His backend work for the minimum viable product has involved implementation of the image request API/handler as well as managing the file transfer protocol server responsible for interacting with JSON files holding image request data. In terms of the report content, he has worked mostly on the use cases section of the technical content

chapter, as well as all of the testing section in collaboration with James Le, any section relevant to the financials of the project, as well as in overall report refinement of all sections.

Walid has been heavily involved in the testing process of the project. This includes unit and most integration tests in the backend. Walid verified the functioning of basic endpoints of the project that send basic satellite/ground station data. This ensured that the basic atomic functionalities of the system were functional ensuring smooth transitions between testing phases. Walid also took the time to verify the functioning of 2 aspects of the scheduling process, being contact opportunities and eclipse opportunities. Finally, he has wrote up integration tests that utilize the endpoints across various parts of the backend system. Throughout testing, Walid made sure to communicate all issues to the respective team members, as well as document the bugs encountered and the processes to fix them.

Walid has also contributed in the development of the front end. He has designed and built the health-dashboard component of the front end. This component is in charge of displaying a general overview of data that Satellite operators can use to get a quick glimpse over the functioning of the system. Walid has also developed the backend components responsible for bringing this front-end interface to life, and tested them thoroughly.

4.4.1.3 Strengths & Weaknesses

Walid's major strengths lie in accountability and honesty. He is very quick to ensure that any shortcomings or poor performance are declared by him before others. This is a very important quality to possess in a large-scale project like this, especially in an Agile management system. Walid's other strengths lie in writing clean code with minimal complications and confusion as well as having an eagerness to take on daunting technical tasks.

Walid's major weakness, in the beginning of the semester lies, in starting tasks later rather than sooner, which leaves things to the tail ends of sprints. The tasks are performed and of high quality, but it makes the integration of features challenging. Moreover, Walid needs to ensure that the extensive level of project visibility that exists within the team's framework and the associated course documents is taken advantage of. He has tried to be aware of the major milestones and expectations in this work environment.

However, in the winter term he has significantly improved in terms of starting tasks earlier and meeting deadlines. Walid now starts tasks much earlier in order to meet the deadlines, as well as tackle any sudden issues that arise beforehand. This helped the team in meeting overall goals on time and decreasing the likelihood of inconvenient delays.

Overall he is a strong team member with a very high potential. Walid has addressed all

the feedback he's recieved head on and has had an exponential improvement in productivity and output as the year progressed.

4.4.1.4 Peer Review Reflection

Walid had taken all the feedback he has received in the fall semester into consideration and made sure to tackle all issues brought up by his fellow peers. Walid has had improved performance and much more increased involvement in project development making him a solid contributor in the project.

Walid has noticed how his slow performance in the team was affecting their ability to meet goals quickly and ensure a smooth consistent flow of deliveries in the sprints. And so after digesting all the feedback regarding that performance from his peer's criticisms, he has hit the gears running in the winter semester. He has now been meeting deadlines consistently, and has sped up his overall performance in the sprints, as well as became more involved in what was happening in the project development and not just the fulfillment of ENG 4k deliverables. This meant the team was able to get over hurdles and meet goals much quicker than before.

He has no major issues in terms of conflict management. He addresses most issues head on and makes appropriate compromises when needed while also being accommodating of other members of the team.

Walid has no issues with communication as he is always honest and direct with what needs to be said or what needs to be tackled. But, Walid has missed around 2 meetings without prior notice, however he made sure to communicate afterwards that he was mistakenly distracted with other course work which led to his absence. In meetings, Walid also had a tendency to rebuttal the answers made by stating he already knew how a certain function or tool works which was thought to be rude at times. He however made sure to apologize when it was brought to his attention.

To address this feedback, Walid made sure to set constant reminders about future meetings in order to avoid accidentally missing them. As well as to be more self-conscious about the slightly aggressive tone or overall responses to certain suggestions.

4.4.2 Ruth Bezabeh

4.4.2.1 Personal Statement

Ruth is a software engineering student who was very enthusiastic about getting to work on an impactful project. Although she is new to space related concepts, she was eager to learn about the topics and be on a multidisciplinary team. This experience has been a great opportunity to exchange knowledge and contribute to a system that brings together multiple skills and is happy to have experience that. She has contributed to multiple integral parts of the system and is excited to apply her newly learned skills in the future. She is grateful to have worked with a great team and produce a successful project.

4.4.2.2 Contribution Details

Ruth worked in the backend, building the service that manages incoming satellite activities. It includes receiving, validating, and storing requests as well as pre-processing orders to determine valid scheduling times. She has contributed to the database design in making sure it aligns with the data and relation requirements. She has also been working on refining the project requirements collaboratively through continuous meetings with the team as well as the sponsors. This has resulted in clearing ambiguous requirements, misunderstandings, and project expectations.

During the winter semester Ruth worked on implementing an outbound service. This included aggregating orders, uplinks and downlinks for each satellite and routinely sending them to our mock ground stations. She created the endpoints for the ground stations that also validate the incoming schedules. She performed unit and integration tests on the service ensuring expected results. Lastly she generated various data sets to be used for testing schedule validity and performance.

4.4.2.3 Strengths & Weaknesses

Ruth's strength lies in her proactive nature. Despite not being initially familiar with the project's technology, she displays a commendable effort to swiftly learn and apply new concepts. She always stays on top of where the team is and tries to make sure the project does not stray from the requirement specifications by asking relevant questions that steer the team toward clear project objectives. She doesn't shy away from addressing issues, preferring instead to collaboratively brainstorm solutions for the team's benefit.

While Ruth is dedicated, her lack of experience is shown when it comes to building optimized and clean code. She also takes more time to complete some tasks that might require her to do side research. Moreover, although she's proactive and thorough in her work, increasing collaborative work sessions could potentially amplify the impact of her contributions. She is also somewhat reserved and this slows the progress of the project but also personally in building her skills. The team and she would benefit from increased communication.

4.4.2.4 Peer Review & Reflection

From the Fall semester peer review, Ruth has received feedback that mainly focuses on her willingness to learn and put in work. While this is a positive, it also serves as a reminder that she has space to grow. Ruth has taken this feedback to build on her skills and plans to spend even more time doing so in the future. She had previously received feedback highlighting code structure as an area of improvement. she has implemented the suggestions and has been writing code that is modular and easier to read. Her code now prioritizes readability and optimization and she will be taking those into account in her future work. For conflict management, Ruth had received a low score in avoiding. While it was good to want to address disagreements, she has learned that sometimes avoiding disagreements is beneficial when they do not have a significant impact.

From the final peer review Ruth received feedback recognising her commitment and meaningful contribution to the project. she demonstrated reliability in her tasks, inputs during meetings and support for the team. One recommendation she received was to show her code and tests during meetings so the rest of the team could be updated and give feedback. she has since done that more frequently but she believes there is always room for improvement.

4.4.3 Rafael Dolores

4.4.3.1 Personal Statement

Rafael is a very passionate and experienced software developer with extensive past experience in data science and full-stack development. He has extensive experience with Amazon Web Services, Python, and JavaScript, having done several industry-grade projects with event-driven systems. He is motivated by creating high-quality software that helps improve the user experience as well as ensuring that the software is friendly to future developers

and stakeholders who may end up using it. Rafael has focused on learning technologies in this project that he did not have previous experience with such as CesiumJS, RabbitMQ, Tauri and Kubernetes. He has been making sure that the rest of the team focuses on the technologies he has experience with so that they can learn it very well as he focused on working with the advanced tools due to his strong interest in those. However, Rafael has made efforts to ensure he is available to help the rest of the team by imparting his experience in the basics of full-stack development so that they have an easier learning curve. Rafael is happy to deliver a very high-quality system that the stakeholders enjoy using.

4.4.3.2 Contribution Details

Rafael has been an extensive contributor to the project. He led the way in designing the overall system architecture and worked with several other team members' research findings to fine-tune the components of the finalized technology stack. He led the way in implementing the microservices framework that the project is based on. This ensured that the rest of the team had time to learn the ins and outs of the tools being used and that by the time they were ready, the development work to be done was not difficult for them to begin. Moreover, Rafael has also led the way in developing the frontend of the application which has been received very well by the project stakeholders.

Over the break and during winter semester Rafael developed the scheduling algorithm the system uses. He sought feedback from the sponsors and further optimised it incorporating their inputs and improving the schedule performance metrics. He did a lot of research into using a reinforcement learning algorithm and other optimization methods. He redesigned the front-end, implemented asset visualization using cesium. Rafael also deployed the system on AWS and created a desktop app using Tauri.

Finally, Rafael has worked extensively on documentation (including videos that the team can use as training), reports, presentations, and stakeholder milestones/meetings to ensure that everything runs smoothly. In terms of this report, he has worked across all major areas as a writer and has also served as an editor for the report overall.

4.4.3.3 Strengths & Weaknesses

Rafael has strengths in several areas that have contributed to the successes the team. Rafael excels in leading by example and motivating others to do the same. He is great under pressure as he always works ahead of deadlines, ensuring he is fully up to date on all aspects of the project. Rafael ensures to ask his teammates and the stakeholders

important questions that may be difficult to ask when in the heat of the moment. He is a very strong programmer with exceptionally clean code that adheres to the correctness of system requirements. His experience has made him well rounded and his skills have been demonstrated in all areas of the project.

A previously noted weakness of Rafael was not holding his teammates accountable and going too far to help others in their assigned tasks. He has since been communicating his expectations and letting team member accomplish their own tasks. A minor weakness that can now be seen is in sharing some of his skills so that the project could benefit from a team shares knowledge and answers each other's questions.

4.4.3.4 Peer Review & Reflection

Rafael has exceptional peer review feedback both fall and winter semesters. He has near-perfect ratings from his team members and this lines up with his own personal reviews of his own performance as well. There are minimal discrepancies between the two review sources. However, he felt he has areas he can improve upon. First, he has taken steps to improve on live presentations and peer review activities. In terms of his conflict resolution approaches he has scored moderate to high in all areas of dominating, integrating, avoiding, accommodating, and compromising. This shows that Rafael has a high degree of social awareness and understanding when it comes to the SatOpsMQ team dynamics. Rafael's area of improvement in terms of conflict management was in the compromising approach it was the lowest score but still at a moderate level, being in line with the population average. To improve on this, Rafael has tried to ensure that he practices solutions that are fair to all parties involved in a discussion.

In addition he has received praise from his teammates for his dedication, reliability and leadership over the course of the year. He has consistently upheld the project's high technical standards and his commitment has been evident in his work. He also received feedback from the team to be more open to help members when they have questions as well as to commit his work more often. He values the feedback and will be taking them into account for future projects.

Finally, Rafael has ensured that he helps keep the team accountable for any issues and shortcomings rather than take pressure upon himself to correct the mistakes and oversights of the team. Rafael also values that his team speaks highly of him. He is very content with his team's high approval of his dedication.

4.4.4 Youssef Hany

4.4.4.1 Personal Statement

Youssef is a very passionate software developer with an extensive history of academic and industrial work in the space engineering field. He has worked in business analysis for aerospace manufacturing processes, systems design, and software development for scientific computing. He is most excited about the domain of this project as optimization and space science are his favorite problem domains. Youssef loves converting abstract problems into implementable solutions by learning and using computational methods. He is a skilled Python programmer and very familiar with Python for scientific problem-solving. He wants to ensure that high-quality software is developed by focusing on the optimization side of the project requirements. Moreover, Youssef wants to take the lead on the project management work due to his interest in systems engineering.

4.4.4.2 Contribution Details

Youssef worked extensively on the background research for the technology being used and the problem domain. He has helped the other team members catch up on their knowledge of satellites and their practical functions. Next, Youssef has taken the lead in developing the algorithms needed to create and optimize schedules that the system aims to tackle. He has worked on extensive portions of this report by taking head on large portions of chapter two and chapter one as well as helping to edit the overall report. Finally, Youssef has led the charge in creating presentations and videos to present to the project stakeholders.

4.4.4.3 Strengths & Weaknesses

Youssef's strengths are working ahead of time, being fully aware of key project milestones and always staying positive to motivate the rest of the team. He has done all of his work on time with ample space to review and improve the work. Moreover, he has ensured that he is available to help others review their work when they need this. Youssef is very skilled at staying on task to ensure the results of his work are of high quality. Moreover, he has been a key role model to lead the team to success.

In terms of areas to improve on, Youssef can start learning the processes of refactoring and integration of developed code to ensure that all the work he does makes it to the final production-level software that is created. Another minor area of improvement for Youssef

is to ask more clear and concise questions during stakeholder meetings so that confusion does not arise.

4.4.4.4 Peer Review & Reflection

Upon reviewing his peer review feedback Youssef has seen that his perceptions are in line with the rest of the team. The team thinks very highly of him and believes he is tackling the work very well. There are no real discrepancies between his ratings of himself and the team's ratings of his performance. He has been commended for his immense dedication and hard work.

Youssef has also noted that his conflict management approaches seem to focus on the dominating and integrating paradigms. He uses the compromising, accommodating and avoiding styles less often. He believes he can take a more holistic approach to these approaches to apply the other styles more appropriately to the situation at hand when needed.

4.4.5 Stanley Ihesiulo

4.4.5.1 Personal Statement

Stanley is another highly experienced software developer with previous industry experience in full-stack development. He wants to ensure that this project is delivered with a high-quality final product. He is very interested in the interactions software systems have with the physical systems in the real world. His major areas of interest are cyber-physical systems and software architectures. He has extensive experience in past open source projects having created a framework that converts informal HTML-based sheet music into formal sheet music, called TabToXML. He also has experience with porting large-scale software from one programming language to another, having gone through this process with TabToXML. Stanley is excited to work on an even-driven system using RabbitMQ and wants to ensure that the best practices for this programming paradigm are obeyed. In addition, he also wants to ensure that the project is scalable and adheres to the requirements.

4.4.5.2 Contribution Details

Stanley has helped develop several developer tools for the backend of the project that ensure that the team has an easier time developing on top of the backend. He has created

internal tools that allow for the simple integration of RabbitMQ into the system by creating a custom library that extends RabbitMQ and is usable by all services in the backend of SatOpsMQ. Moreover, he has created a unified workflow for database connection and dockerization of the system so that the team can develop easily across all platforms they may be working from. This has ensured no issues in terms of collaborative development for all teammates in relation to their own personal work environments/styles. Stanley has also helped create extensive documentation in the form of diagrams and training videos for the team to use for their work. Moreover, Stanley has helped to ensure that the stakeholders and team agree on requirements as well as modifying the solution so that it conforms to the requirements. In addition, he has helped push back on requirements and helped the team sober their expectations on what is feasible to develop to avoid feature creep in the product backlog. Finally, Stanley has contributed much to this report by leading the way in the technical content chapter and the diagrams contained there.

4.4.5.3 Strengths & Weaknesses

Stanley's greatest strength is his proficiency in technical problem-solving. He is very good at working independently to get all programming tasks completed, as well as completing any documentation-related tasks and stakeholder obligations. He made sure to create solutions that are well balanced and simple so that all stakeholders and team members can understand the solutions as well as build upon them. Stanley was also very empathetic and understanding of other teammate's needs. He makes sure to communicate with the team in a very polite manner while trying his best to motivate and help them with what they need.

Stanley's areas of improvement include starting tasks ahead of time and doing them little by little slowly. He has admitted that he likes to do long bursts of work to try to finish his tasks. This works for him but it may cause problems in the team as tasks are interdependent and team members need to integrate each other's work to ensure it is fully functional and compliant with system requirements. Another weakness Stanley could have improved on is keeping better track of meeting times and communication with the team on urgent matters. Overall, Stanley was a very critical member of the team in terms of the technical breakthroughs he has been able to accomplish in this project.

4.4.5.4 Peer Review & Reflection

Stanley takes peer review very seriously and has an extensive reflection upon these reviews. Upon reading his peer review reports he realizes that in certain areas he is hard on his

ratings versus the rest of the team and in other areas, he may have a slight blind spot in rating himself higher than his teammates believe. However, the blind spots are very minuscule and he realizes the major discrepancy was being hard on himself. The major point of improvement in his view is to just communicate better and make sure to keep a constant flow of dialogue with the rest of the team to ensure no last-minute surprises are experienced by others.

Stanley had no major issues in conflict management strategy. His common approaches involve compromise, integration, and being accommodating of others. He was not focused on using a dominating style or avoiding conflict. Stanley was very good at handling conflict around design choices and discussing trade-offs. This is evident when he discusses the fine details of the solution. He was passionate about the approaches that must be taken and reminded the group members of the consequences of design choices while also showing evidence in the literature for these cause-effect relationships. Whenever approaching conflicts and whichever approach he may engage in he always listens to others and focuses on the facts rather than speculative reasoning.

4.4.6 Hashir Jamil

4.4.6.1 Personal Statement

Hashir is a competent all-around team member with a desire to be a versatile jack of all trades. Hashir has past experience in academia and research as well as amateur software development experience on personal projects. He has been honored to be a part of this project as it was highly related to his research interests and personal interests. Hashir has always been a strong proponent of open-source development and simple lightweight solutions that can be easily extended by future contributors. He feels that overall many things were done right in this project but that he could have contributed more story points for various development tasks. As well he regrets the shortcomings on the project management aspect of this endeavour. Notably, the fact that the project management philosophy had to be reworked was his major critique of the team. He agrees rework was a good learning experience and not anyone's fault. As well, he believes the team has a much better understanding of Agile project going forward in future projects. Hashir will stay on as a regular contributor to the system as it is expected to have a life cycle beyond the final release. Furthermore, Hashir is very interested in the communication of this project to others. Some closing tasks will be a presentation with to the CSA with the team, creation of youtube videos explaining the code base and future work in this domain.

4.4.6.2 Contribution Details

Hashir has worked extensively on the database design/implementation and the Event Relay API. He has also been a major contributor to the early research for the external components to use in the system, such as RabbitMQ and PostgreSQL, assisting in the finalization of the system architecture. He has also led the creation of many parts of the system documentation, diagrams/visuals, and presentations/videos created for stakeholders. In the later half of the project he worked on asset visualization, several refactoring tasks and a revamp of the project management framework. He also worked on restructuring the project management approach of the team to fit agile more closely.

In terms of this report, he has created several technical figures, worked on extensive portions of all chapters, and served as a facilitator for the report with all other team members. Moreover, he has served as an editor for the report overall. Finally, he has been a key member in helping with stakeholder meetings and presenting the project to the community to bring awareness to it as well as helping put together the team by introducing several members to one another and the existence of the project.

4.4.6.3 Strengths & Weaknesses

Hashir's major strengths are commitment and accountability. He is also very good at communicating with the rest of the team, ensuring he engages in consistent dialogue with all members of the team as well as stakeholders. Other strengths are ensuring diagrams and reports are of high quality, conveying the message they are intended to. He takes initiative in taking on tasks and helping other members of the group whenever they need it. Hashir is very detail-oriented and collaborative in the work he does. He is very good at working with the other members of the team to ensure that the team is on the same page.

Hashir's major weaknesses had been delaying tasks to the last minute and not finishing them ahead of time. As previously mentioned this is detrimental to the integration of features. Furthermore, Hashir can improve his technical problem-solving skills by practicing the implementation of difficult programming tasks. Some of the implementations pushed by Hashir were not up to standard in the early parts of the project and he had to spend extra time to refactor and fix his parts of the code base.

Finally, Hashir has been said by all members of the team to very good at keeping the team on track and serving a leadership role when needed. He goes above and beyond to ensure that the team is on track but may be tough on others and expect too much.

4.4.6.4 Peer Review & Reflection

Upon reviewing his feedback he has reflected that he has a tendency to rate himself with slight discrepancies versus the rest of the team across all measures of performance. In the early stages of the project he had over-ranked himself with perfect ratings and the team had ratings slightly lower than perfect. In the early stages, he was very optimistic and overestimated his contributions. He realized as time progressed that he was a solid team member but still had room for improvement. As the project progressed he gave himself a four out of five in all categories but the team ranked him much higher than this in all categories. He believes he may be concerned with how others perceive him and ranks himself accordingly in terms of performance metrics.

Hashir understands that his major area of improvement is the conflict management approach. He can be more holistic in this matter. He focuses too much on integrating solutions and using bits and pieces from all team member's ideas to make an overall solution. However, he neglects to use many of the other approaches such as compromising, dominating, avoiding, and accommodating styles of conflict management. This style can be made more holistic so that the results obtained are not biased.

Hashir has also received great feed back in facilitating team work, leading meetings and communicating with sponsors. Despite his direct approach the team recognises his care for others in the team and progress of the project has contributed immensely for what has been accomplished.

4.4.7 James Le

4.4.7.1 Personal Statement

James is an experienced software developer with a moderate amount of experience in software testing and full-stack development. He was very excited to work on this project as he is interested in the space science domain. He has not had the opportunity to work in this domain in the past and wants to apply his software development experience while engaging with this problem domain. James is focused on ensuring the system is developed with continuous integration/delivery principles in mind. He wants to use his past experience with web frameworks such as Java Spring and React.JS to ensure the project is developed at a high standard of compliance with the requirements. Moreover, James wants to ensure that the system's features are frequently and consistently added so that stakeholders are able to provide timely feedback.

4.4.7.2 Contribution Details

James has contributed to the project's quality assurance efforts. He has set up the CI/CD pipeline to automate the process of integrating code changes into a shared repository using Github Actions and Discord webhooks. He has helped to create utility and helper functions to refine API calls for improved full system connectivity, alongside writing end-to-end scripts to ensure overall correctness. Beyond technical implementation, he has helped with user interface design and research into the tools that comprise the technology stack that the solution is employing. In terms of this report, James has specifically worked on the majority of the project management and lessons learned content and served as an editor for the whole report.

4.4.7.3 Strengths & Weaknesses

James was good at frequent communication and being accountable for his shortcomings. He had the desire to contribute more to the project as he had a lot of competency in software development. James' knowledge shines particularly in the testing portion of the project, leveraging his previous experience to ensure quality assurance practices are set to high standards.

James had identified time management and task completion as areas for improvement, acknowledging past challenges with accountability. While he has made attempts to address these issues, there is still room for growth. However, James demonstrated a proactive attitude towards learning from past experiences and is committed to enhancing his performance for future roles.

4.4.7.4 Peer Review & Reflection

James initially faced major challenges with contributing to the technical aspects of the project, experiencing a slow start in the fall semester. However, following peer reviews, his efforts were recognized with higher ratings, indicating commendable progress in addressing his earlier issues. He took proactive steps to reassess his situation in the winter, adopting a more balanced schedule to allocate enough time for the project. Despite improvements since the previous semester, James acknowledges that there is still significant room for effort, as he has yet to contribute as much as he initially intended. This serves as a valuable lesson in the importance of time management, not only in one's career but in all aspects of life.

In terms of conflict resolution, James has a well-balanced approach. He regularly employs the avoiding, accommodating, and integrating paradigms. He has noted that he can improve his use of the dominating and compromising approaches. In terms of actual conflict management James has done a very good job addressing the team's concerns and has reflected on the issues by making a plan of action to turn around the issues.

4.5 Team Reflection

After engaging in individual reflections and peer reviews the team has reflected as a whole on the successes and challenges of teamwork. The process has been mostly positive and the team has really come together producing the final release with the resources available. Furthermore, the team members understand one another very well and are very comfortable working together.

The major successes in working together have been learning together as a group the technologies and systems that comprise this solution. The team found much success showing one another how technical programming tasks should be performed. We have learned very well from each other's methodologies in producing clean code that can be reused and built upon iteration by iteration. As well, we have gotten very comfortable using our industry-grade project management tool, Jira, this has been a very enjoyable platform to use for communicating the work to be done. The team is also very satisfied with the open level of communication that all group members are allowed to have. The group space that has been created is very safe but also one that allows for discussion, occasional disagreement, and constant challenge. This has ensured that everyone's ideas have been valued and their concerns addressed.

Some key challenges that had occurred were staying on schedule and staying on task. There had been issues with not all the backlog items being completed on time and some sprints not going according to plan. Furthermore, some members have not always pulled their weight while others have been very consistent, often going above and beyond what is expected. This discrepancy was gradually eliminated and more consistency was cultivated in the team to ensure no team members burnout. The team however is very understanding of all members. We had an open and honest dialogue about these challenges, developed and followed a plan to address this.

To address our challenges we all performed regular check-ins with one another to ensure we were all up to date with key milestones and project tasks. Furthermore, we ensured that tasks were started in the earlier parts of the sprints rather than left to the last minute so

that any problems that arise can be addressed. Finally, we continued to take very seriously the performance and peer review aspects of the project along with the project management and technical details of the project.

Going forward we will take these lessons and be more consistent and balanced in our own individual efforts while trying to replicate the positive aspects of the space we created in all future projects. Some of these areas include prioritizing clear communication, as well as keeping similar levels of team accountability. We believe that our commitment to constant communication and individual/group accountability has been essential for our success.

Chapter 5

Conclusion

In conclusion, the "Satellite Operations System Optimizer" capstone project has concluded with the final release of the SatOpsMQ product. This has represented a significant undertaking by a dedicated team of engineering students at York University, aiming to address the ever-growing demand for efficient satellite-based earth observations. The solution is a novel open-source system with an event-driven microservices architecture and modern user interface. This architecture leverages the Active Message Queuing Protocol by using the open-source event messaging bus RabbitMQ. Furthermore, the system is containerized using Docker so that it is not constrained by any cloud provider. It can be configured to run on any machine or compute cluster. As well, it runs natively in a desktop environment using the Rust Language based Tauri framework.

This project has been guided by a clear vision and the commitment of a diverse team with a mix of software engineering and space engineering backgrounds. The proposed solution stands as a comprehensive system that leverages an event-driven architecture, microservices, and real-time data processing. With a design that considers scalability, the team has laid a strong foundation for satellite operations optimization for future users of this product. The final release presented stands as a modular solution that can continuously be incremented on in the future.

While the project exhibits several strengths, including its thorough system interfaces, universal design, and potential sustainability impact, some challenges and weaknesses still need to be considered. These include optimization constraints for real-time processing, manufacturability considerations when integrating with mock hardware, and the possibility of unintended consequences due to misuse.

The choice of developing the project using Agile methodology and managing the work-

flow on Atlassian Jira software has been one of the best meta-project decisions. This allows for simple yet reliable project management with robust metrics that are generated for analysis. This ensures that less time is spent managing the project and more time is spent developing key features. As a bonus, for a small team like ours, Jira is available for free. Overall this project is very under budget.

We do note that the team had some issues with precise project management throughout the 2023-2024 academic year but have learned from these mistakes. There has also been very careful attention paid to team dynamics, peer reviews as well as individual and overall team performance analyses. While the team was by no means perfect, it has worked very well and did not have any serious issues. The major takeaway is that everyone enjoyed being on this team and have become good friends as a result. The members of the team always felt safe and valued, which was a sign of a positive work environment. The major areas of improvement were not delaying any work and not to commit to more than can be done. Moreover, the team has developed solid technical competencies in a diverse array of software engineering engineering tasks.

The Satellite Operations System Optimizer project holds great promise for the satellite operations community. It represents a step forward in optimizing satellite resources and promoting sustainability in space technology. As the project is handed off to the customers (CSA sponsor), it has the potential to make a substantial impact on the field of satellite operations, contributing to a more efficient and user-friendly satellite data acquisition and management. Thank you for reading this very long report.

References

[1] A. R. Washburn, "Earth coverage by satellites in circular orbit," *Faculty Notes*, 2009. [Online]. Available: https://faculty.nps.edu/awashburn/Files/Notes/EARTHCOV.pdf

[2] "Earth Explorer," https://earthexplorer.usgs.gov/.

[3] L. Riley, V. Ghildyal, C. Bonesteel, and D. Mishra, "Development and implementation of automated planning in cubesats," *Small Satellite Conference*, Aug 2023.

[4] "Space Mission Design Tools - NASA," https://www.nasa.gov/smallsat-institute/space-mission-design-tools/.

[5] "Agi | delivering mission success," https://www.agi.com/. [Online]. Available: https://www.agi.com/

[6] R. S. Park, "Jpl planetary and lunar ephemerides: Export information," https://ssd.jpl.nasa.gov/planets/eph_export.html, 2020.

[7] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris, "Event-driven programming for robust software," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, ser. EW 10. New York, NY, USA: Association for Computing Machinery, 2002, p. 186–189. [Online]. Available: https://doi.org/10.1145/1133373.1133410

[8] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 243–246. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7958496

[9] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, mar 2014.

[10] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*, 2016, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/7506647

[11] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems*, vol. 49, no. 1, p. 71–79, jan 2015. [Online]. Available: https://doi.org/10.1145/2723872.2723882

[12] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, "Performance evaluation of microservices architectures using containers," in *2015 IEEE 14th International Symposium on Network Computing and Applications*, 2015, pp. 27–34. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7371699

[13] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 2006. [Online]. Available: https://ieeexplore.ieee.org/document/4012603

[14] V. M. Ionescu, "The analysis of the performance of rabbitmq and activemq," in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 2015, pp. 132–137. [Online]. Available: https://ieeexplore.ieee.org/document/7311982

[15] V. Patel, "Analyzing the impact of next.js on site performance and seo," *International Journal of Computer Applications Technology and Research*, vol. 12, pp. 24–27, 10 2023. [Online]. Available: https://www.researchgate.net/publication/376046436_Analyzing_the_Impact_of_NextJS_on_Site_Performance_and_SEO

[16] A. Bandaru, "Amazon web services," 12 2020. [Online]. Available: https://www.researchgate.net/publication/347442916_AMAZON_WEB_SERVICES

[17] "Tauri." [Online]. Available: https://tauri.app/v1/api/config

[18] E. Asaolu, "Tauri vs. electron." [Online]. Available: https://blog.logrocket.com/tauri-electron-comparison-migration-guide/

[19] "Github actions quickstart." [Online]. Available: https://docs.github.com/en/actions/quickstart

[20] S. Beram, "What is a feasibility study?" 2020. [Online]. Available: https://blog.logrocket.com/product-management/feasibility-study-template-examples/

[21] V. authors, "Satellite tool kit (stk) - nasaspaceflight.com," 2023. [Online]. Available: https://forum.nasaspaceflight.com/index.php?topic=26298.0

[22] K. Beck, "Manifesto for agile software development," 2001. [Online]. Available: http://agilemanifesto.org/

[23] C. Drumond, "What is scrum and how to get started." [Online]. Available: https://www.atlassian.com/agile/scrum

[24] M. Rehkopf, "User stories with examples and a template." [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories

[25] D. Radigan, "Story points and estimation." [Online]. Available: https://www.atlassian.com/agile/project-management/estimation

[26] "Themes, epics, stories, and tasks." [Online]. Available: https://www.aha.io/roadmapping/guide/agile/themes-vs-epics-vs-stories-vs-tasks

[27] M. Rehkopf, "Stories, epics, and initiatives." [Online]. Available: https://www.atlassian.com/agile/project-management/epics-stories-themes

[28] M. Cohn, "Epics, features and user stories." [Online]. Available: https://www.mountaingoatsoftware.com/blog/stories-epics-and-themes

[29] SolveXia, "5 types of risk mitigation strategies for business success," https://www.solvexia.com/blog/5-types-of-risk-mitigation-strategies, 2023.

[30] W. Farmer, "Cost-effective payload operations planning software for complex small spacecraft mission operations," *Small Satellite Conference*, Aug 2023.

[31] National Aeronautics and Space Administration, "5.3 product verification," 2023. [Online]. Available: https://www.nasa.gov/reference/5-3-product-verification/

[32] M. Rostanski, K. Grochla, and A. Seman, "Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq," in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 879–884. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6933108

[33] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, ser. DEBS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 227–238. [Online]. Available: https://doi.org/10.1145/3093742.3093908

APPENDICES

Appendix A

Acronym List

Acronym	Definition
CSA	Canadian Space Agency
GS	Ground Station
SAT	Satellite
MVP	Minimum Viable Product
USGS	United States Geological Survey
MASS	Multi-aspect Automated Satellite Scheduler
NASA	National Aeronautics and Space Administration (NASA)
STK	Systems Tool Kit
AGI	Analytical Graphics, Inc.
API	Application Programming Interface
SGP4	Simplified General Perturbations 4
JPL	Jet Propulsion Lab
FOV	Field of View
UML	Universal Modelling Language
AWS	Amazon Web Services
CORS	Cross-Origin Resource Sharing
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
DBMS	Database Management Systems
HTTP	Hyper Text Transfer Protocol
AMQP	Advanded Message Queuing Protocol
MQ	Message Queue
JSON	JavaScript Object Notation
CFD	Cumulative Flow Diagram
UI	User Interface

Table A.1: Acronym List.

Appendix B

Additional Figures

B.1 Final Release User Interface

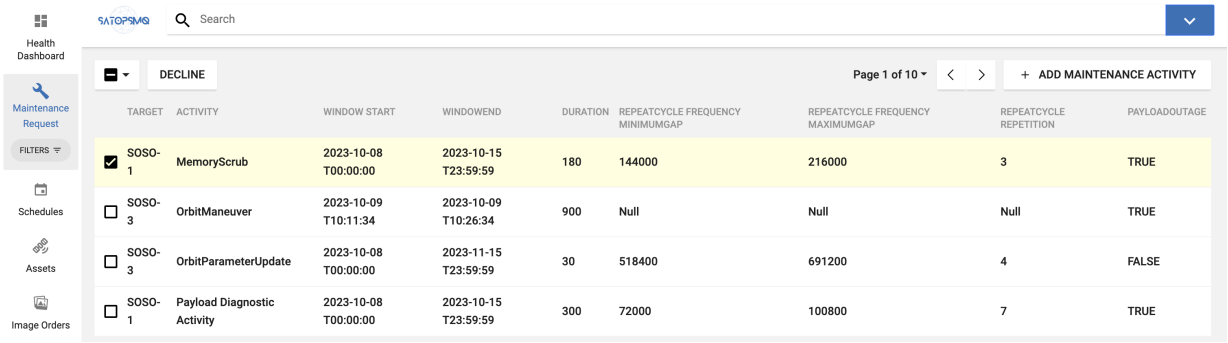


Figure B.1: Maintenance Activities Table

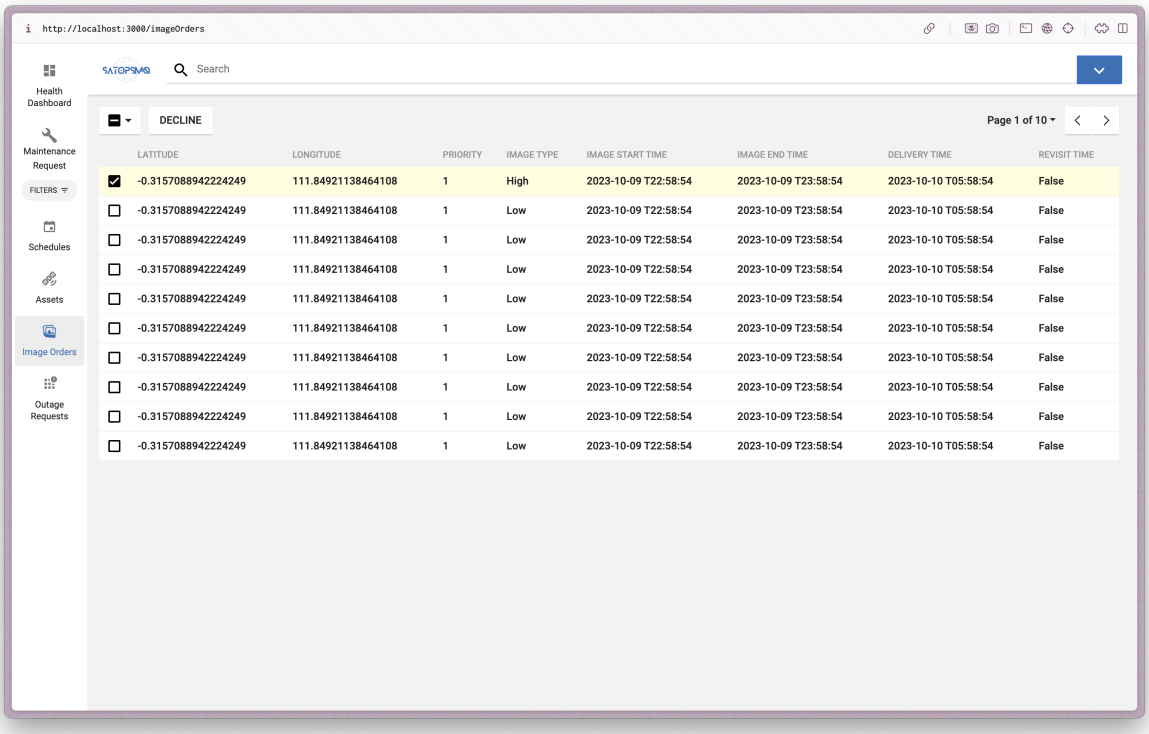


Figure B.2: Image Order Dashboard

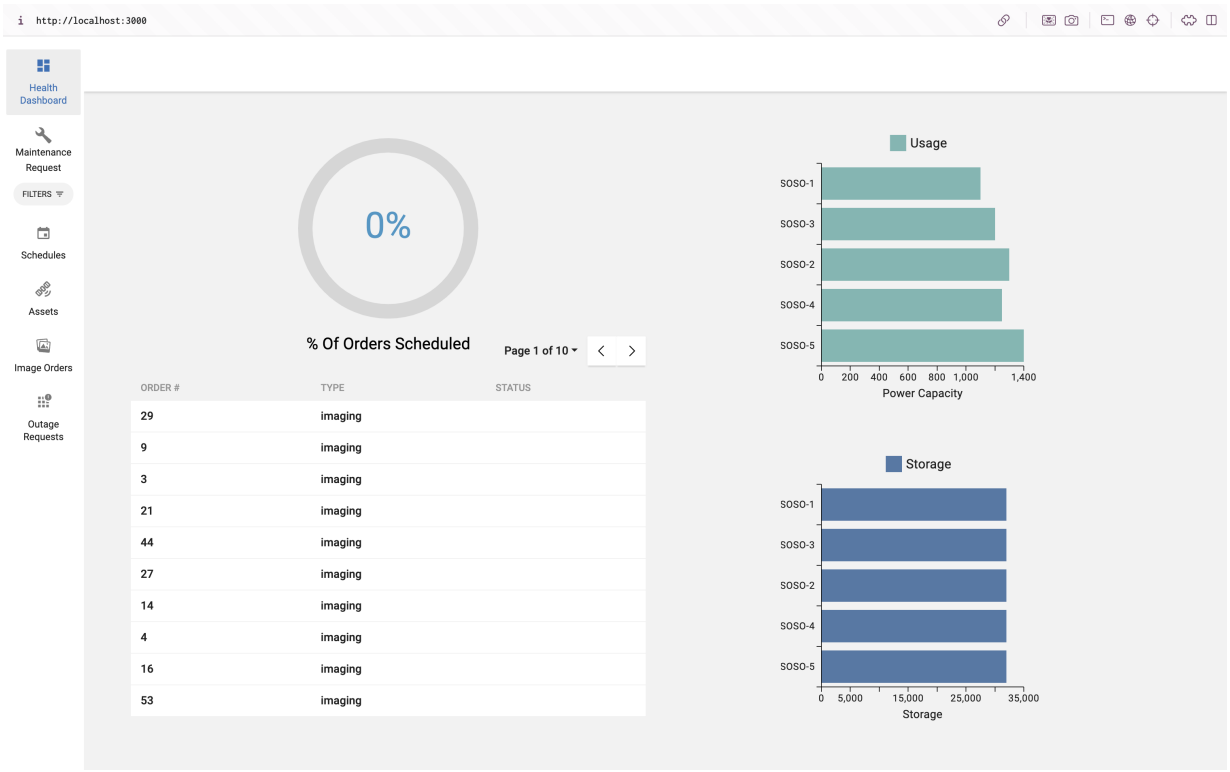


Figure B.3: Asset Health Status Dashboard

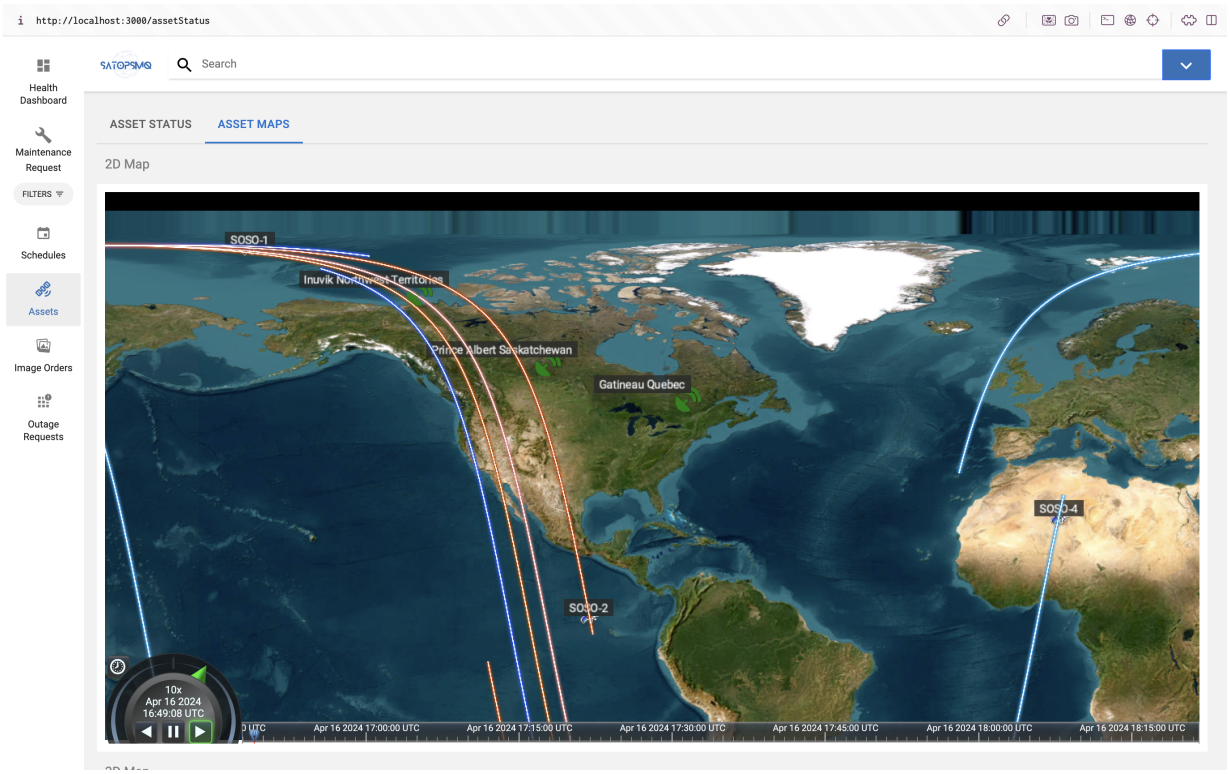


Figure B.4: Live 2D Asset Map

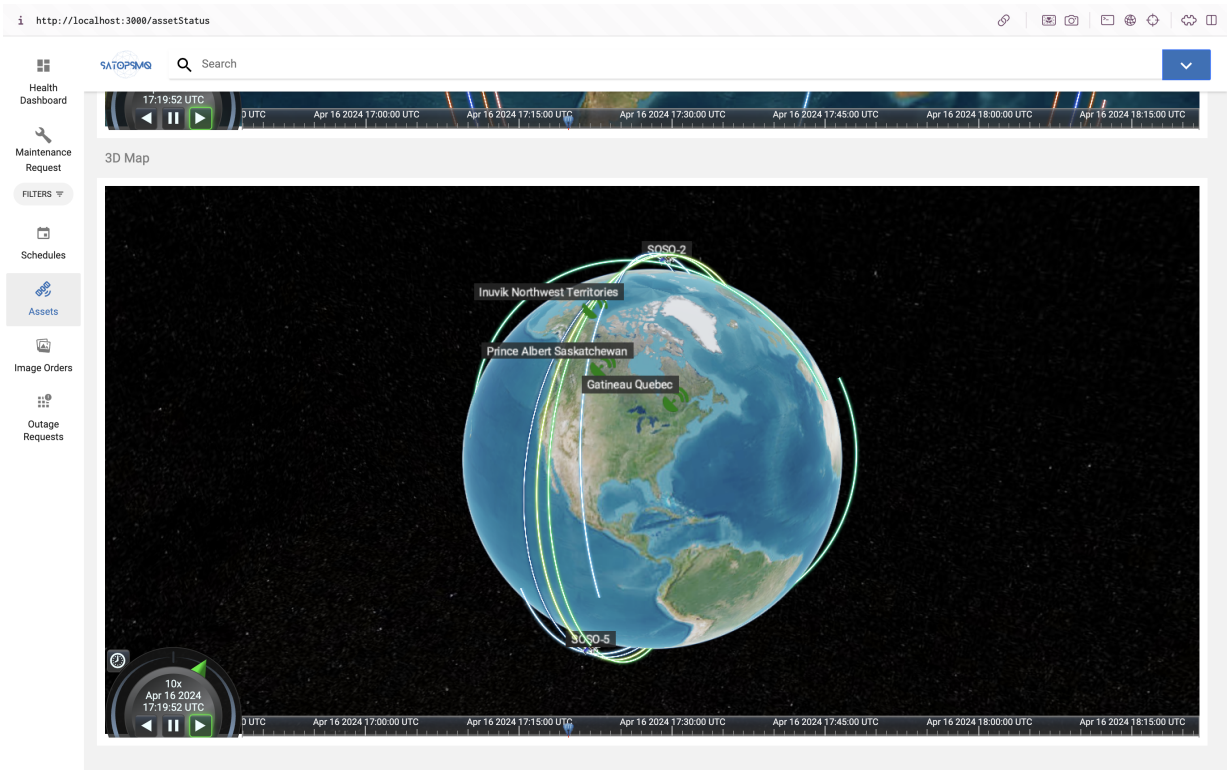


Figure B.5: Live 3D Asset Map

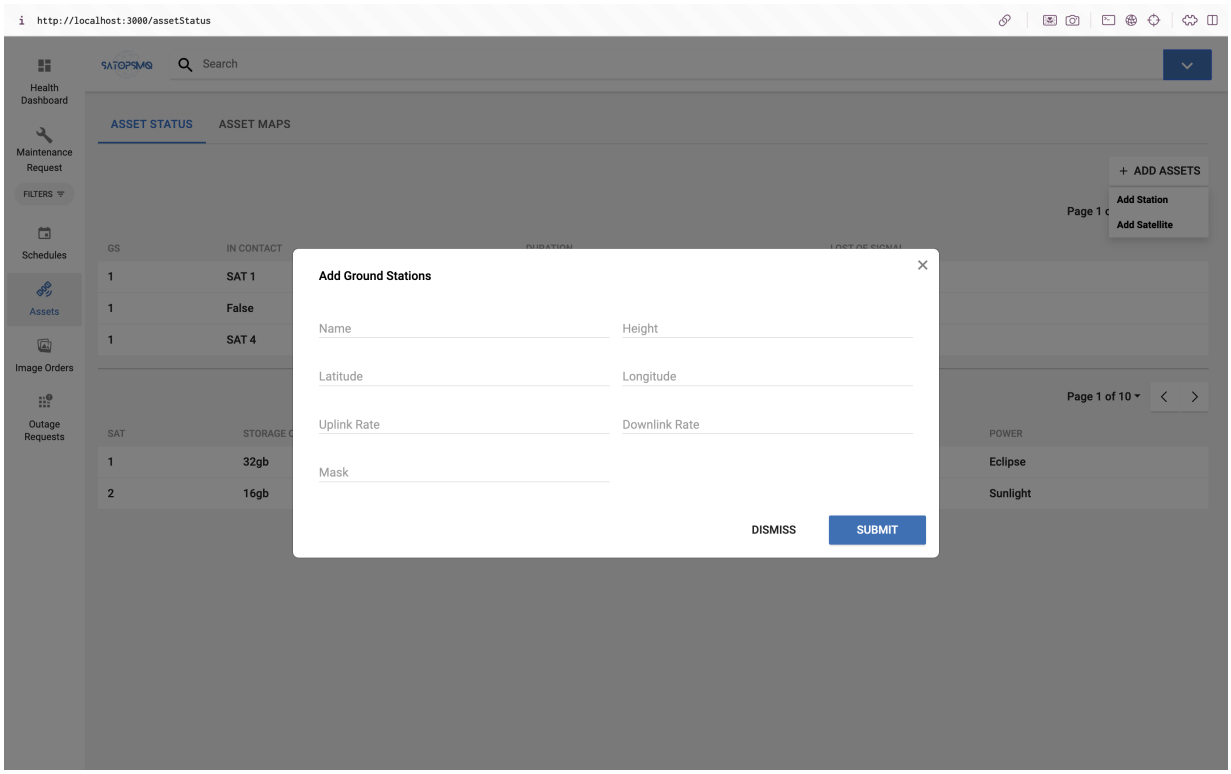


Figure B.6: Add Groundsation Form

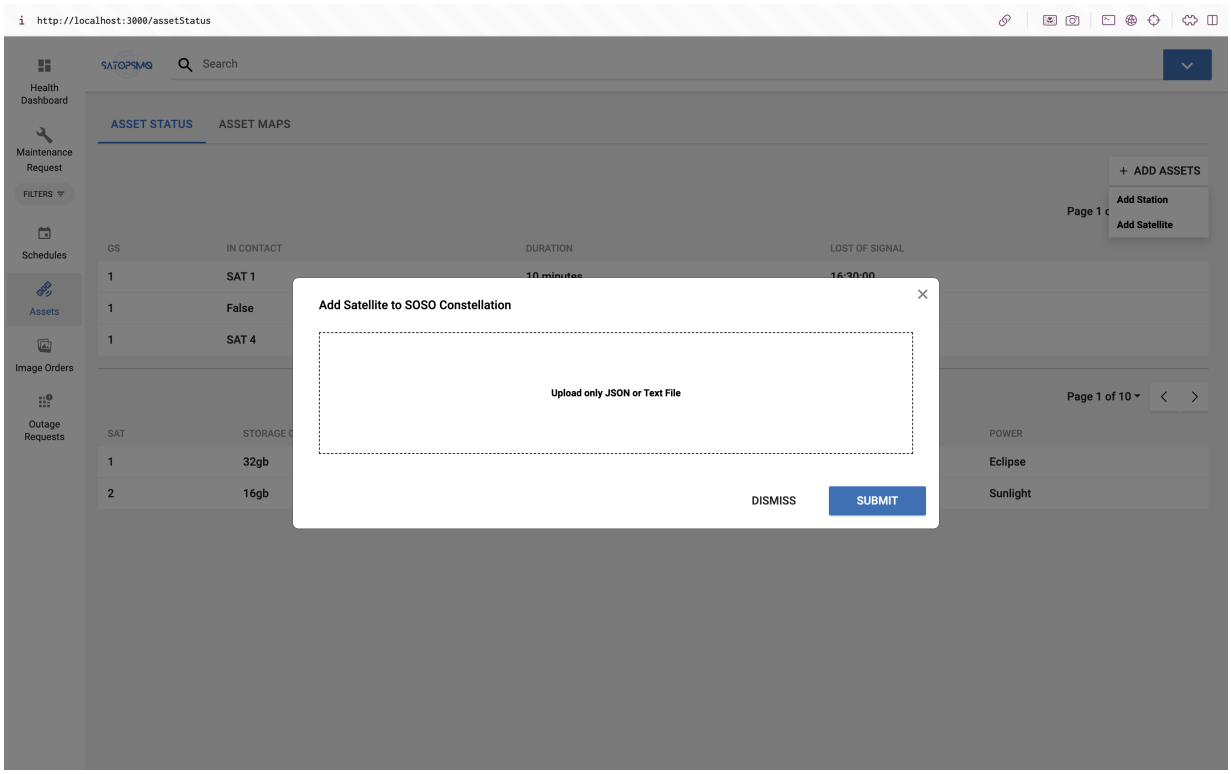


Figure B.7: Add Satellite Submission Box

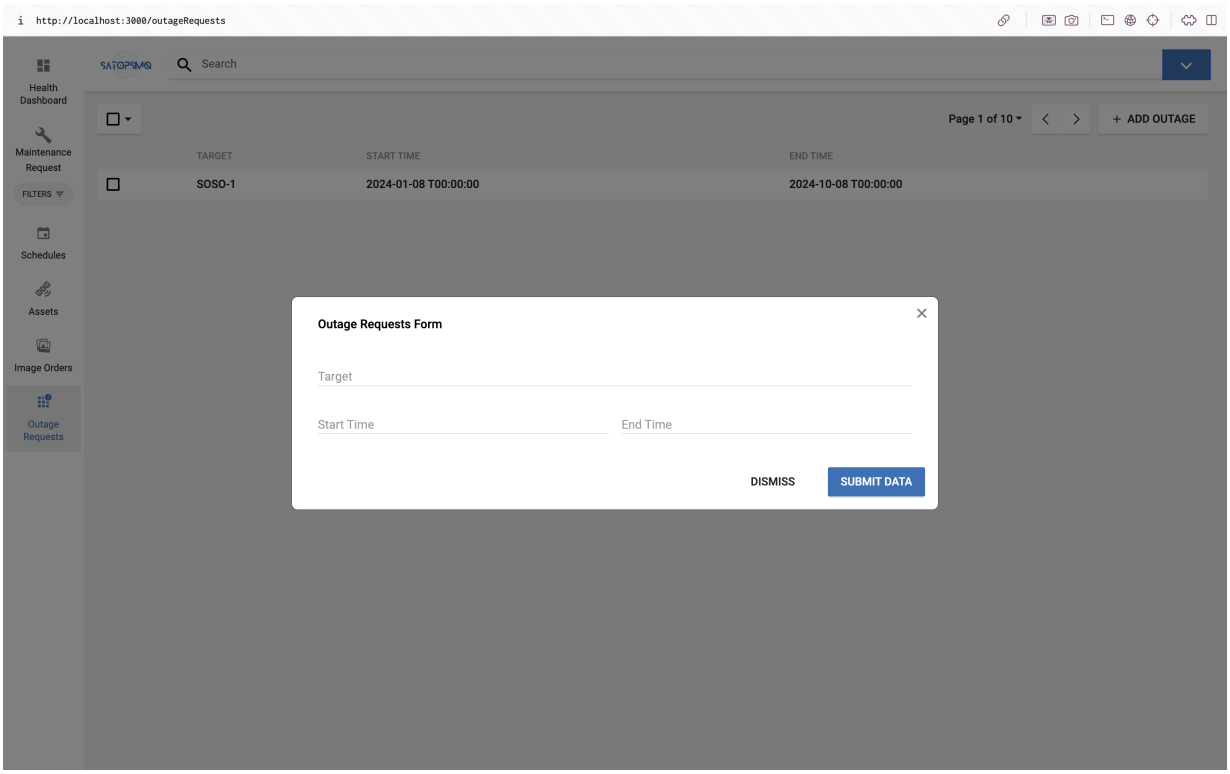


Figure B.8: Outage Request Form

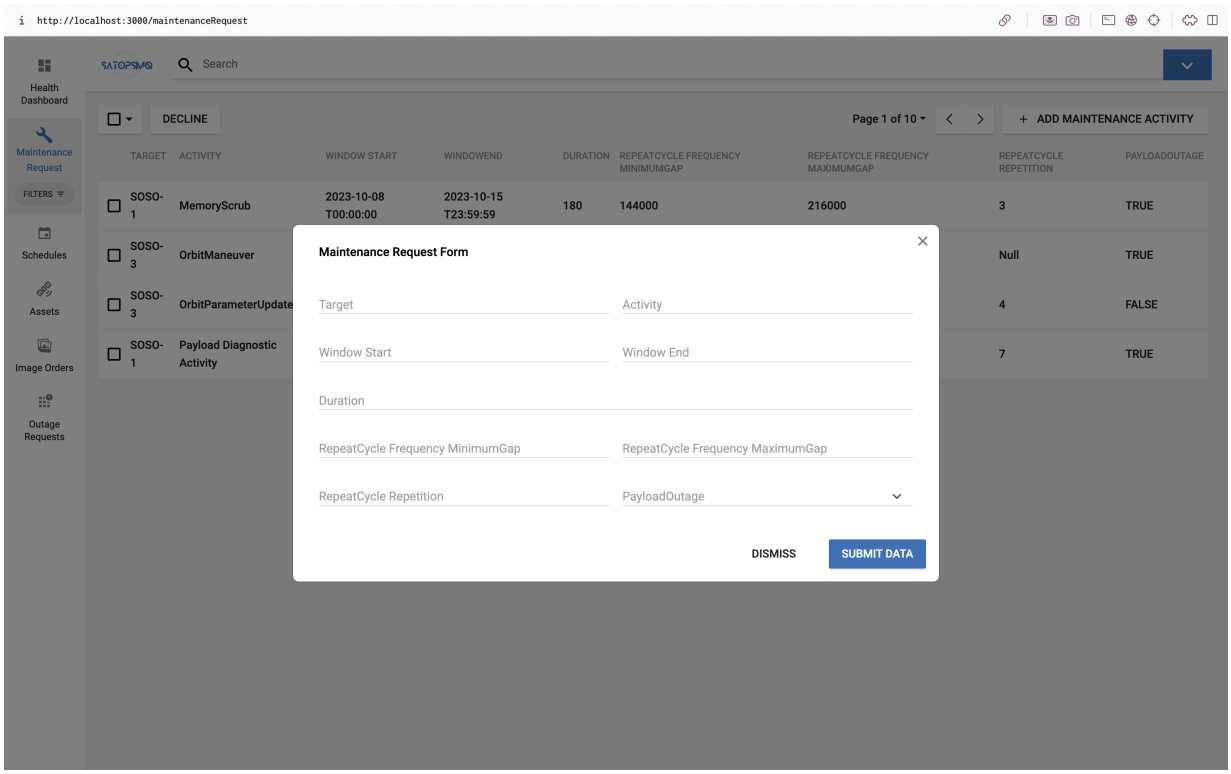


Figure B.9: Maintenance Activity Request Form

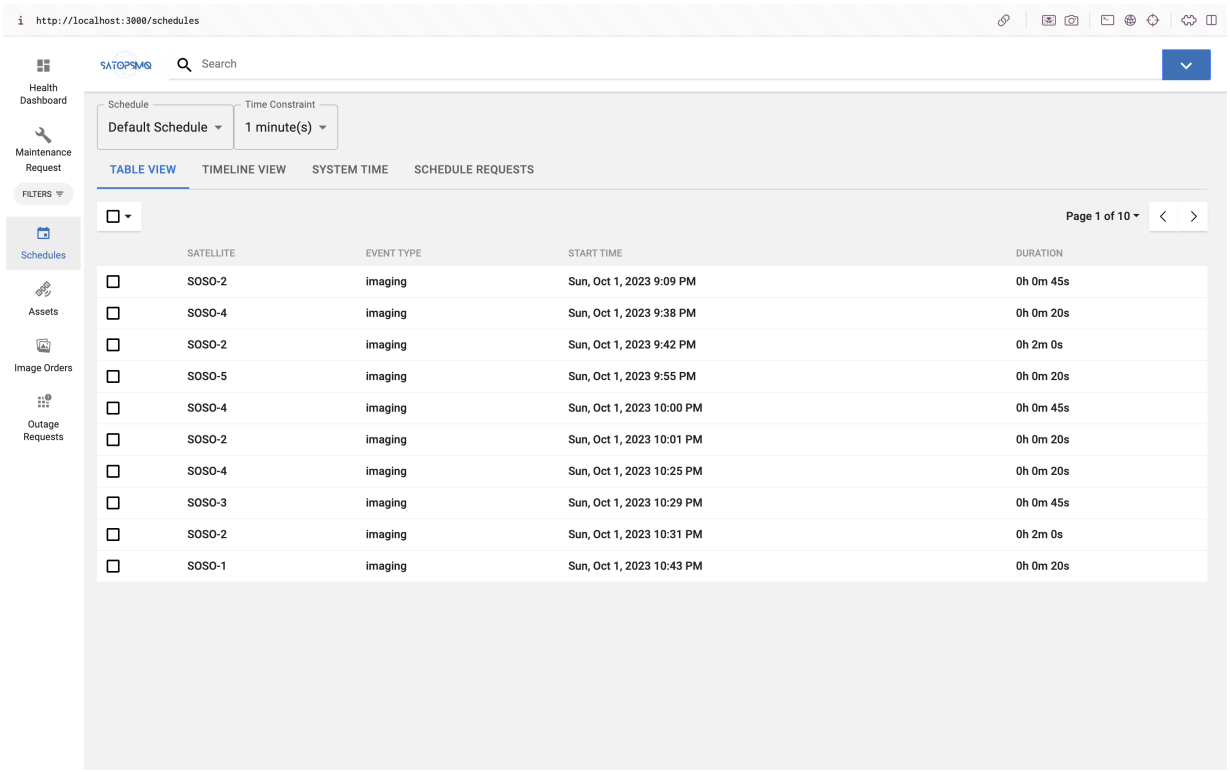


Figure B.10: Schedule Events Table View

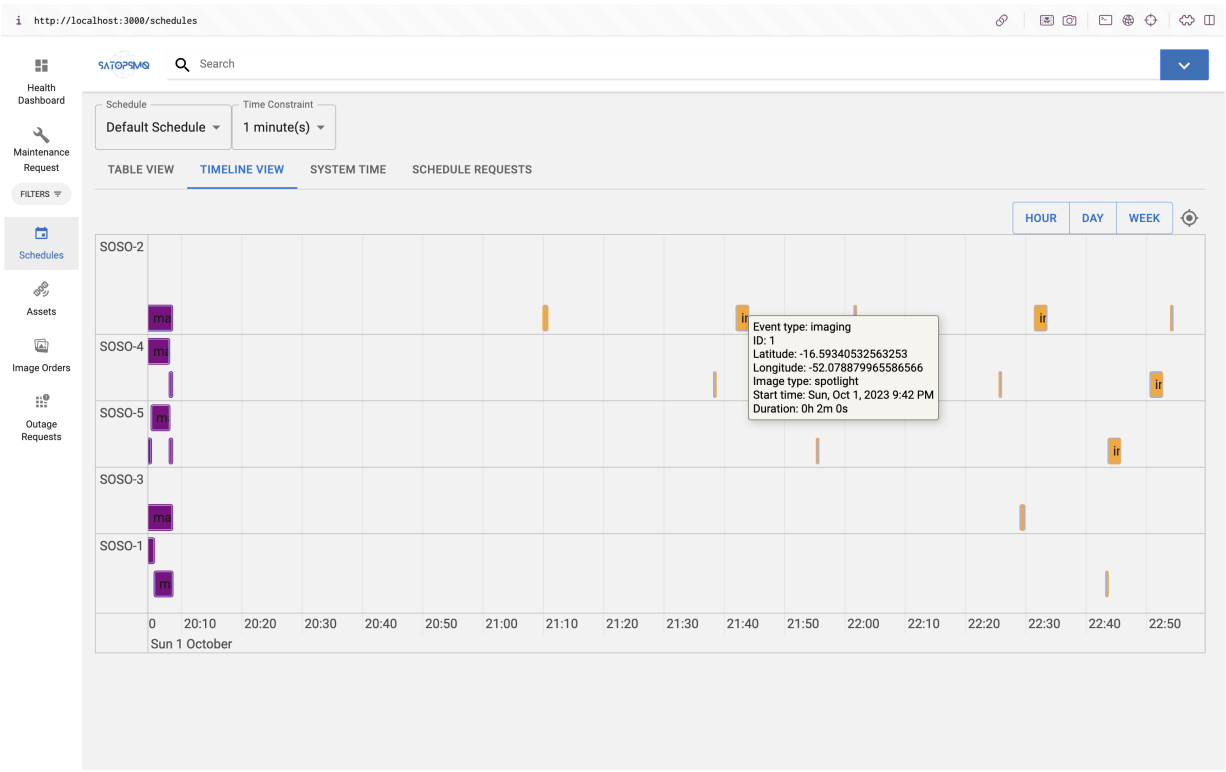


Figure B.11: Schedule Events Timeline View

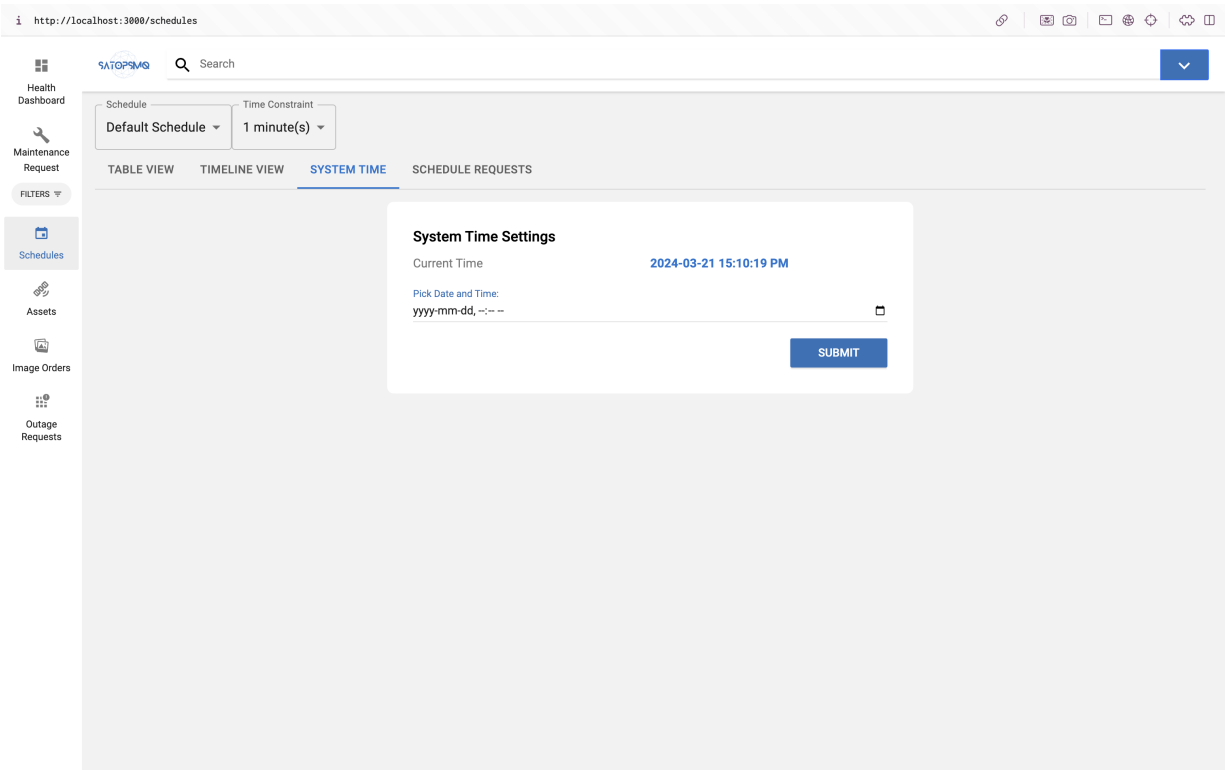


Figure B.12: System time setting page

B.2 MVP User Interface

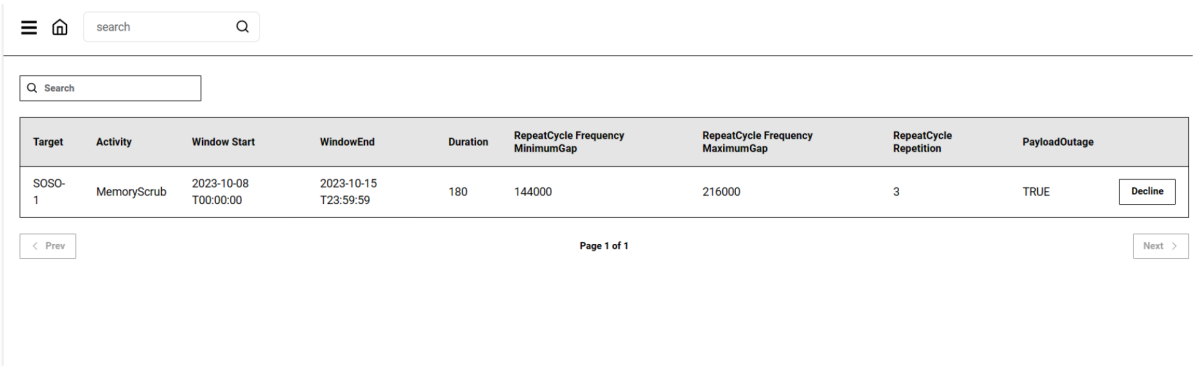


Figure B.13: Maintenance Activities In Progress

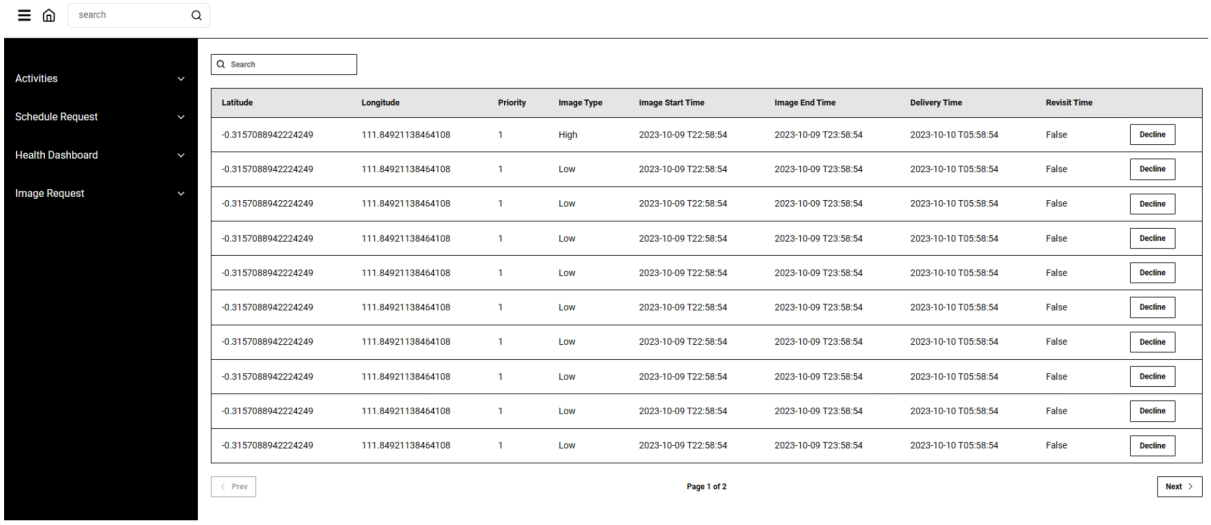


Figure B.14: Image Order Dashboard

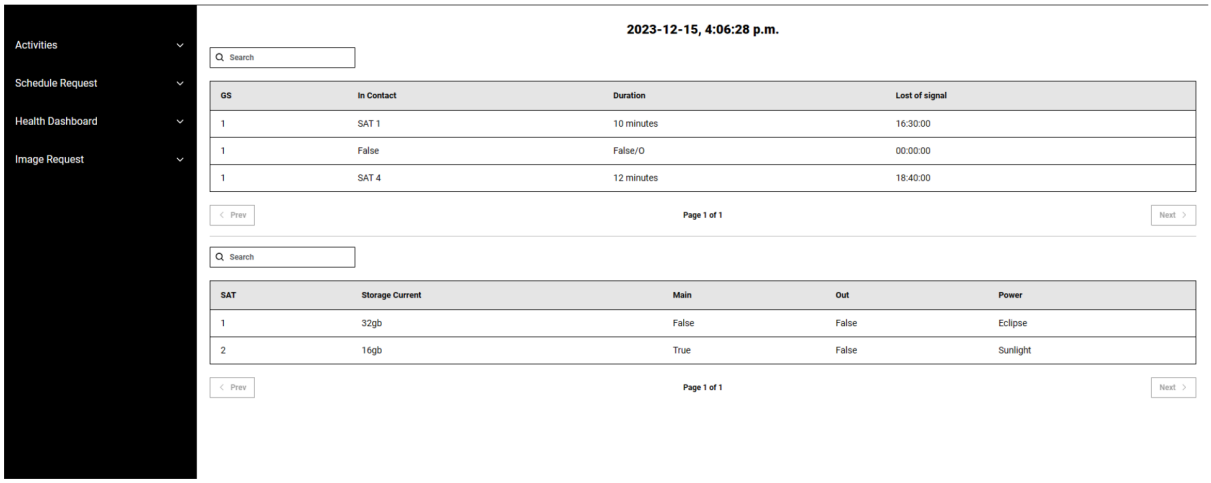


Figure B.15: Asset Health Status Dashboard

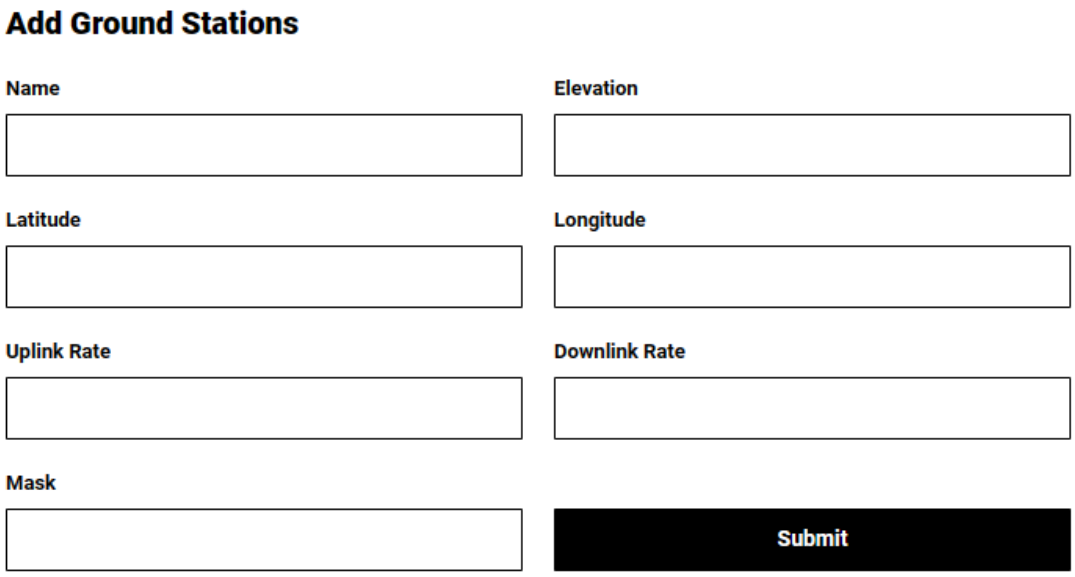


Figure B.16: Add Groundsation Form

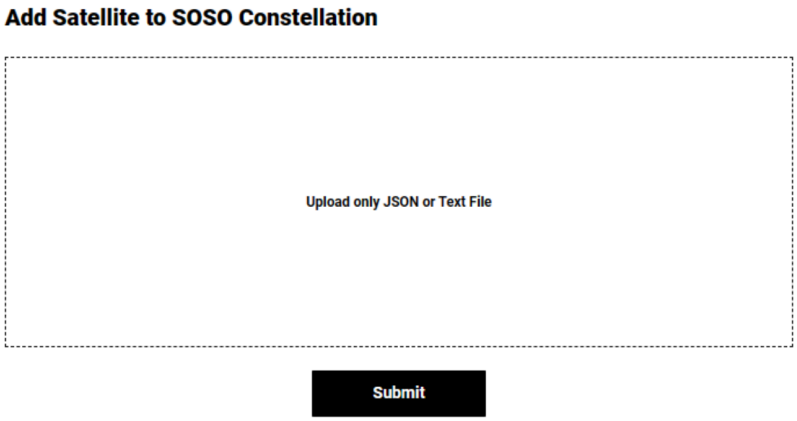


Figure B.17: Add Satellite Submission Box

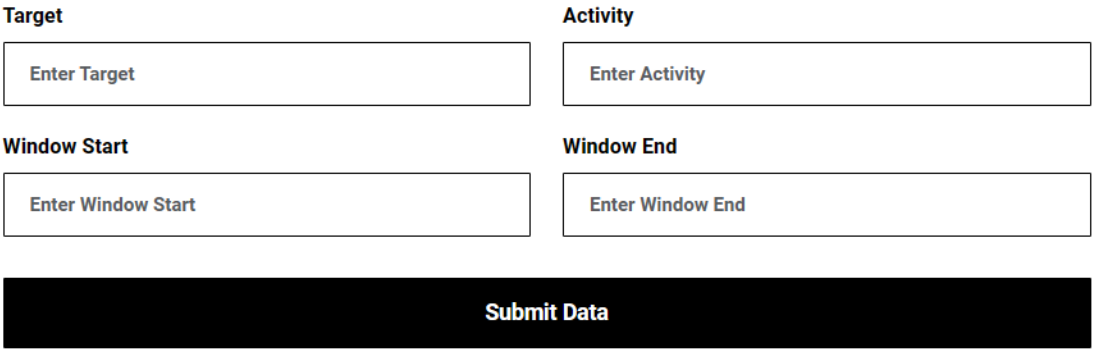


Figure B.18: Outage Request Form

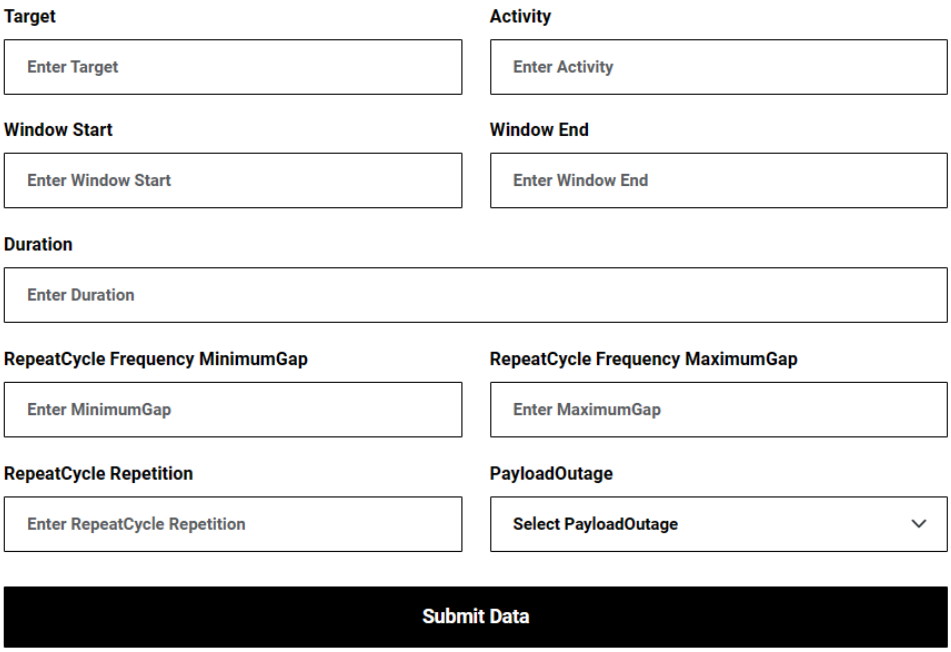


Figure B.19: Maintenance Activity Request Form

B.3 Amazon Web Services Metrics

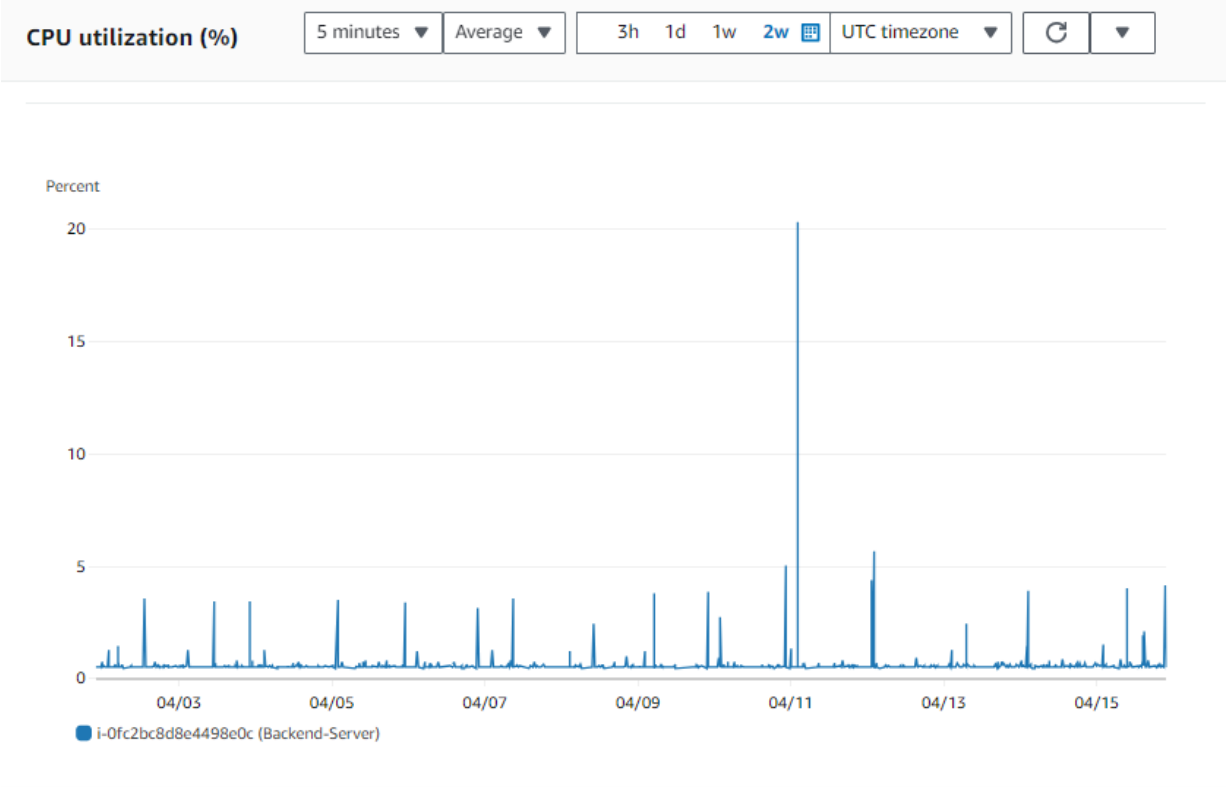


Figure B.20: CPU Utilization by percentage

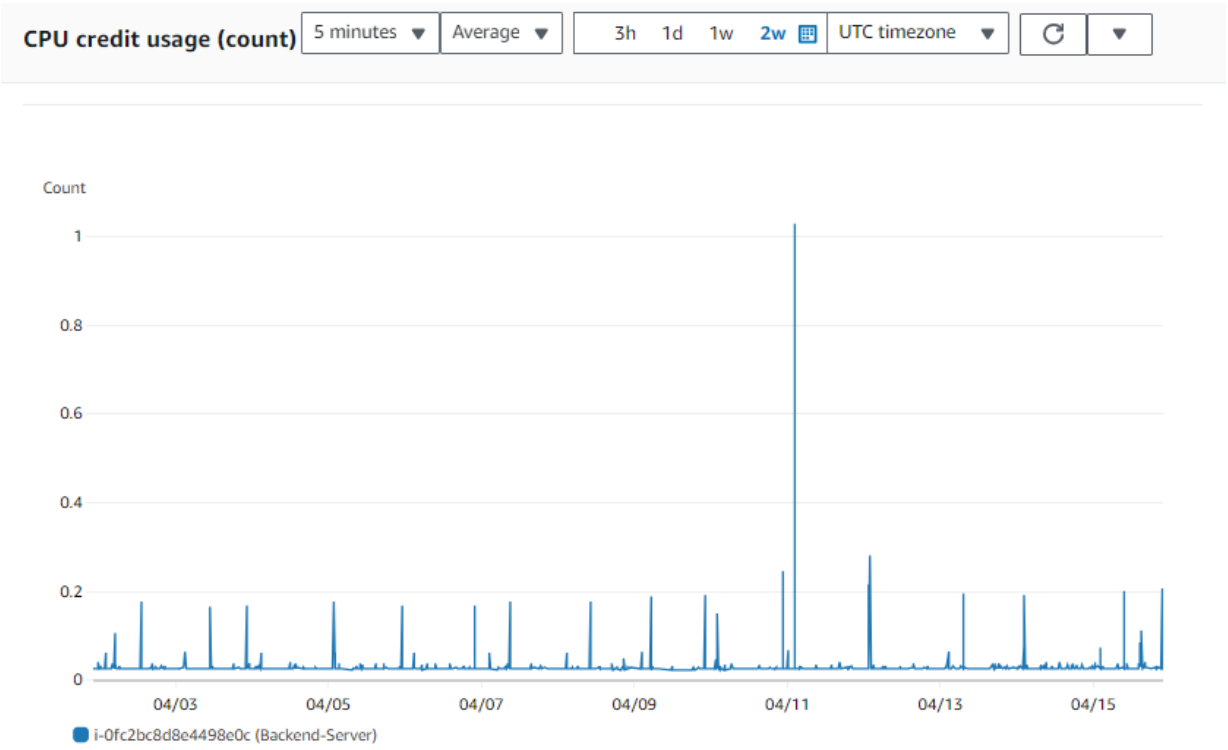


Figure B.21: CPU credits used

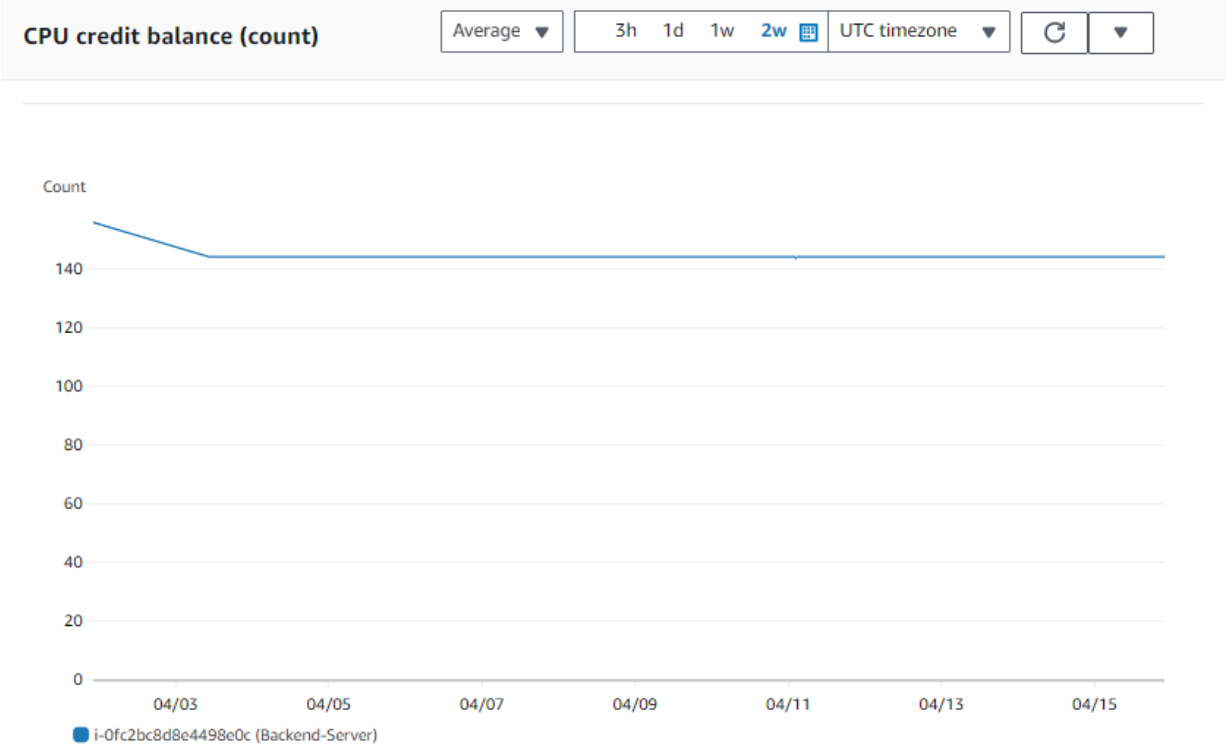


Figure B.22: CPU credit balance remaining

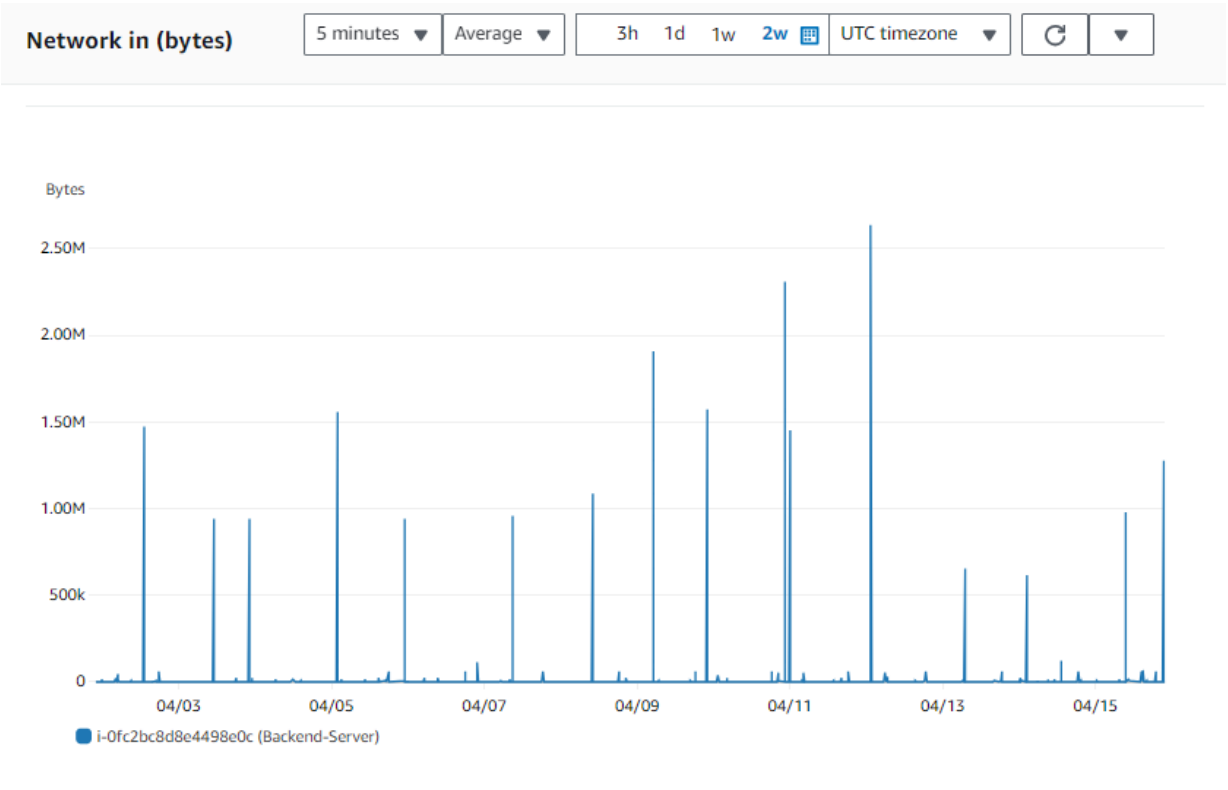


Figure B.23: Inbound network traffic by bytes

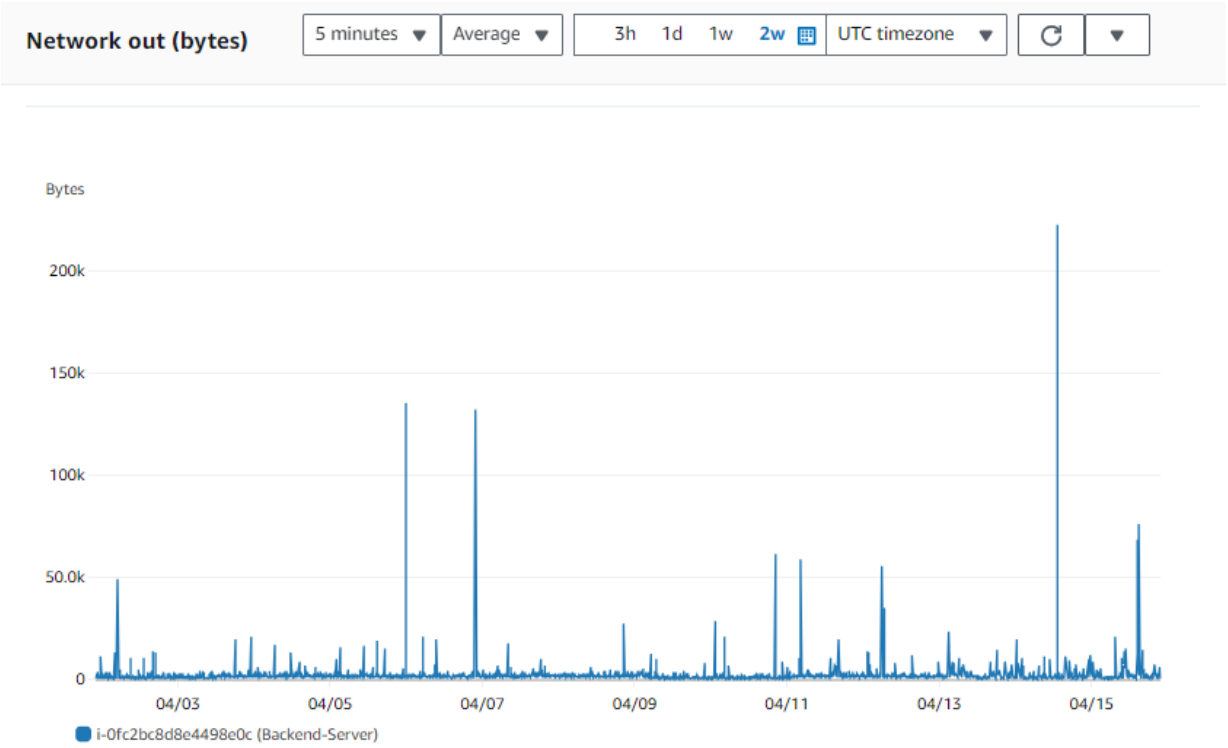


Figure B.24: Outbound network traffic by bytes

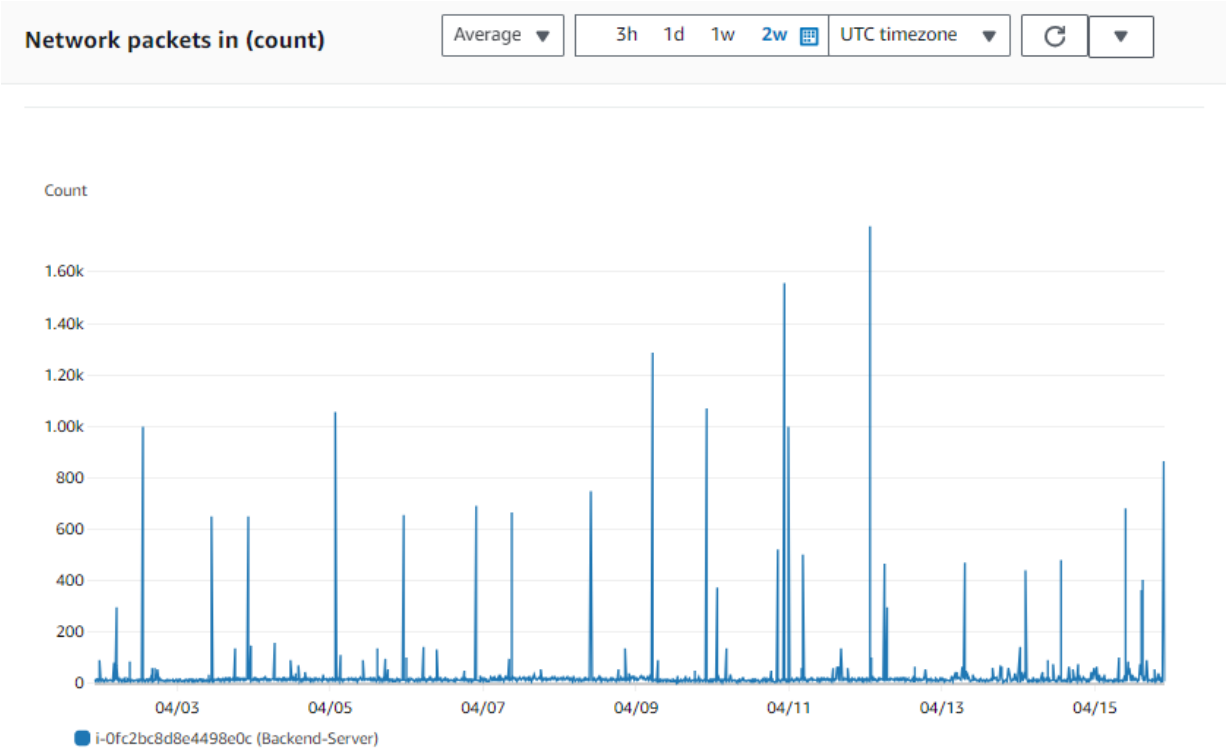


Figure B.25: Inbound network traffic by packets

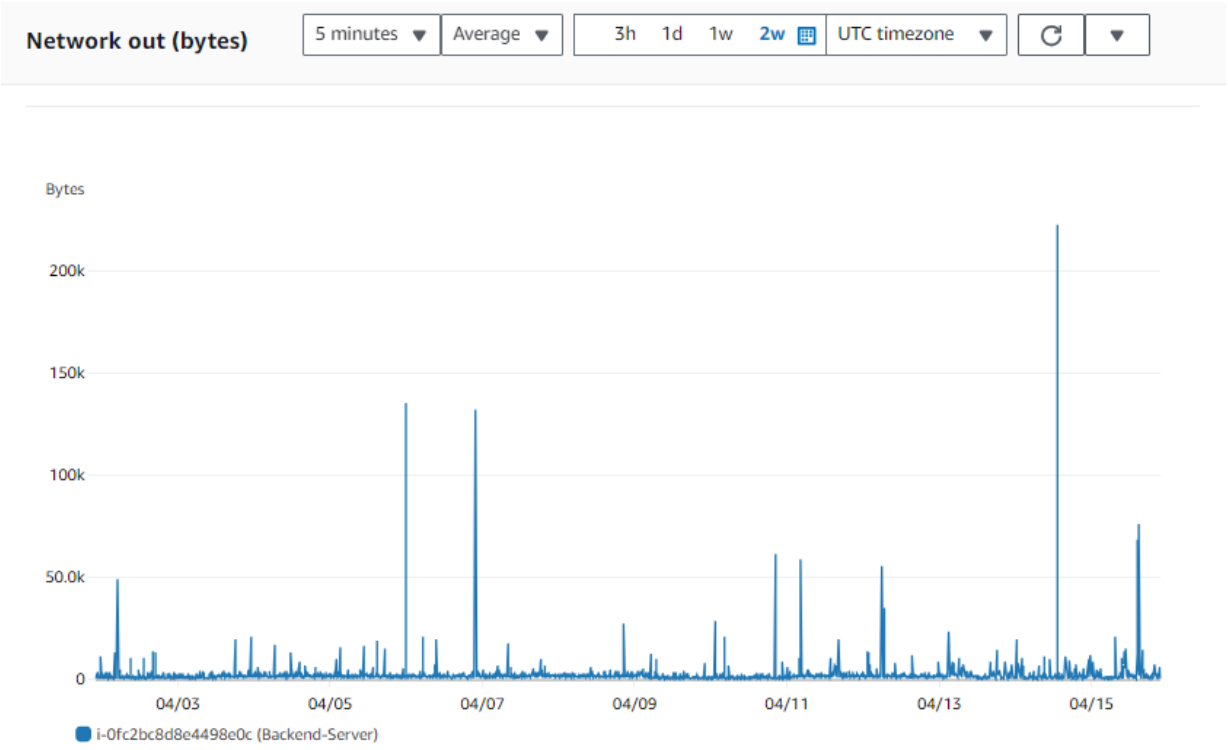


Figure B.26: Outbound network traffic by packets

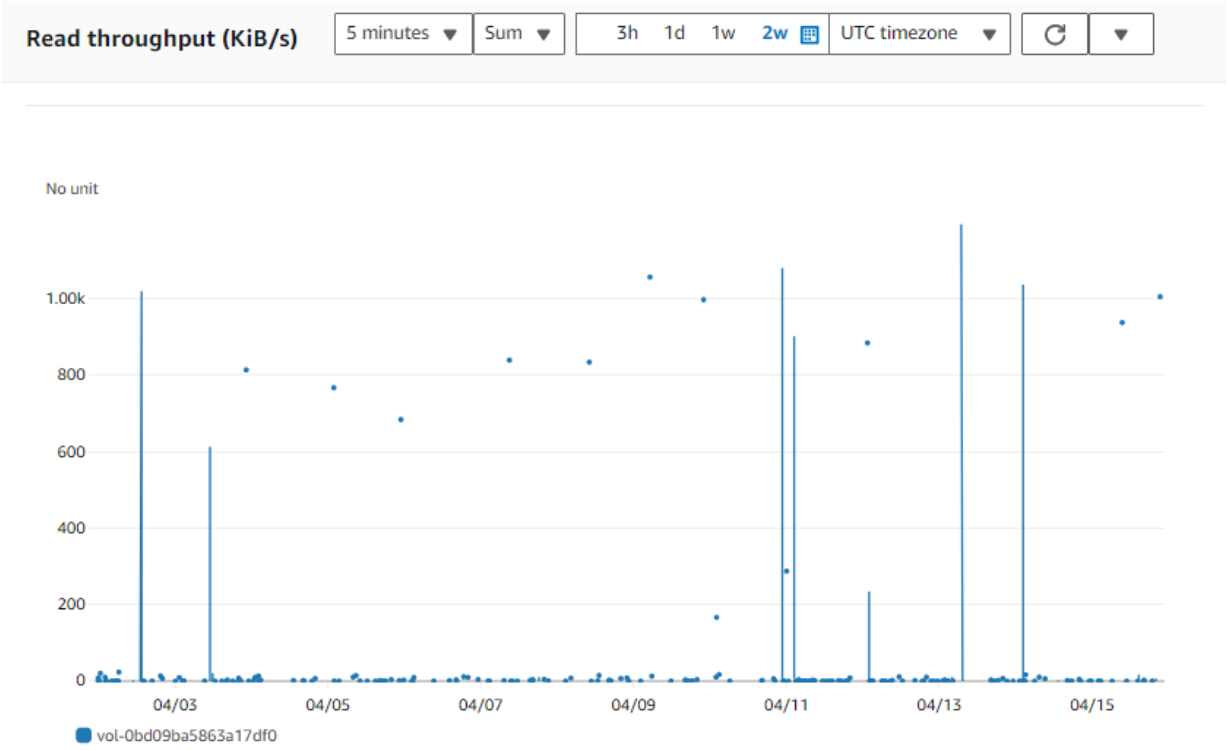


Figure B.27: Read Throughput in Kilobytes per second

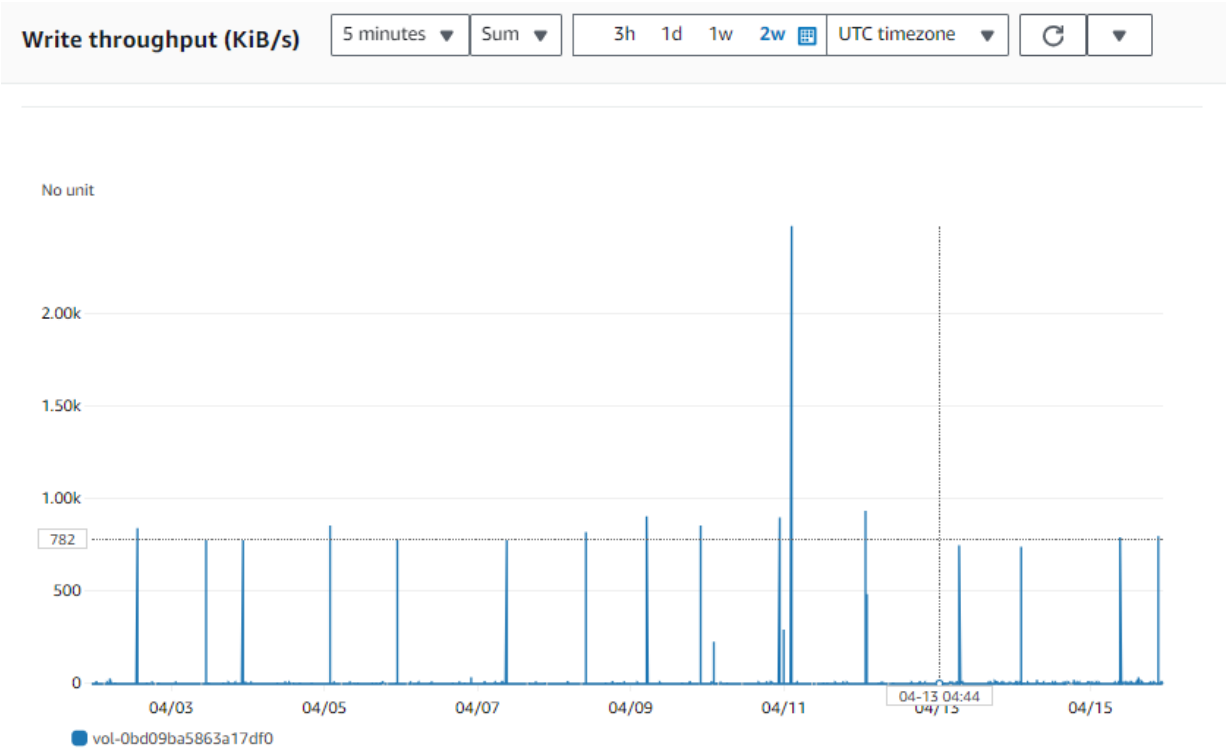


Figure B.28: Write Throughput in Kilobytes per second

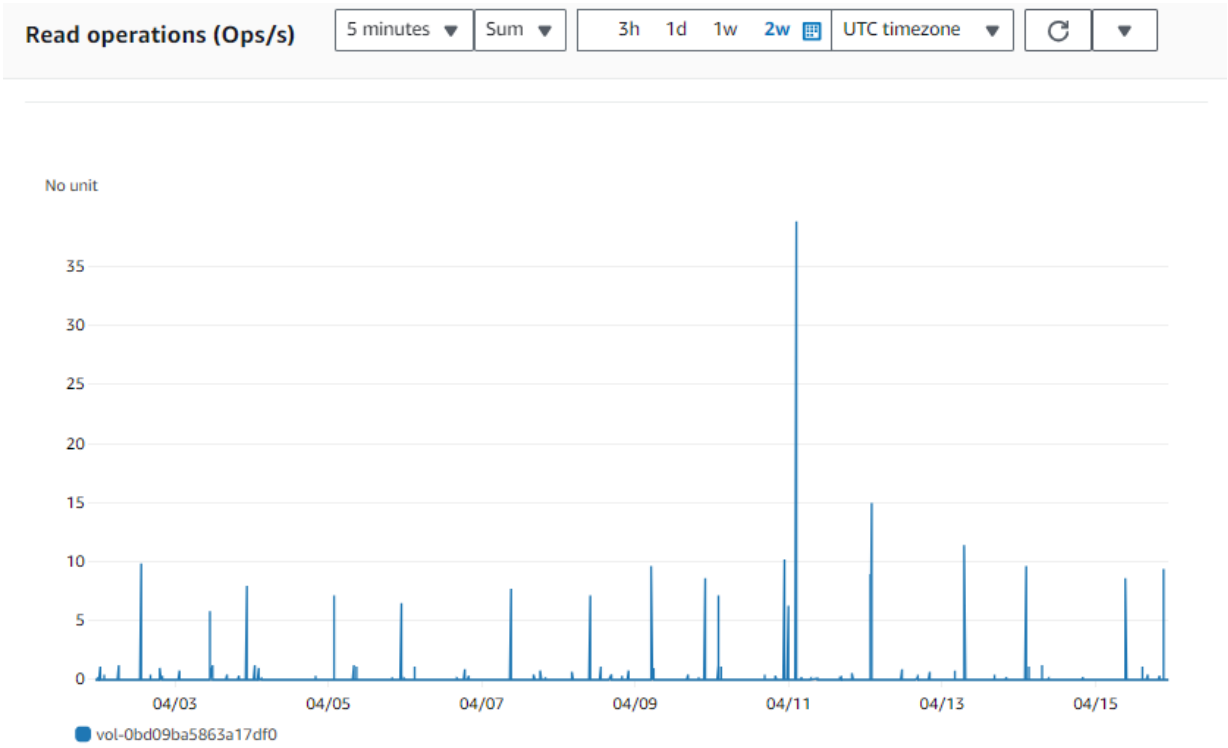


Figure B.29: Read Operations Per Second

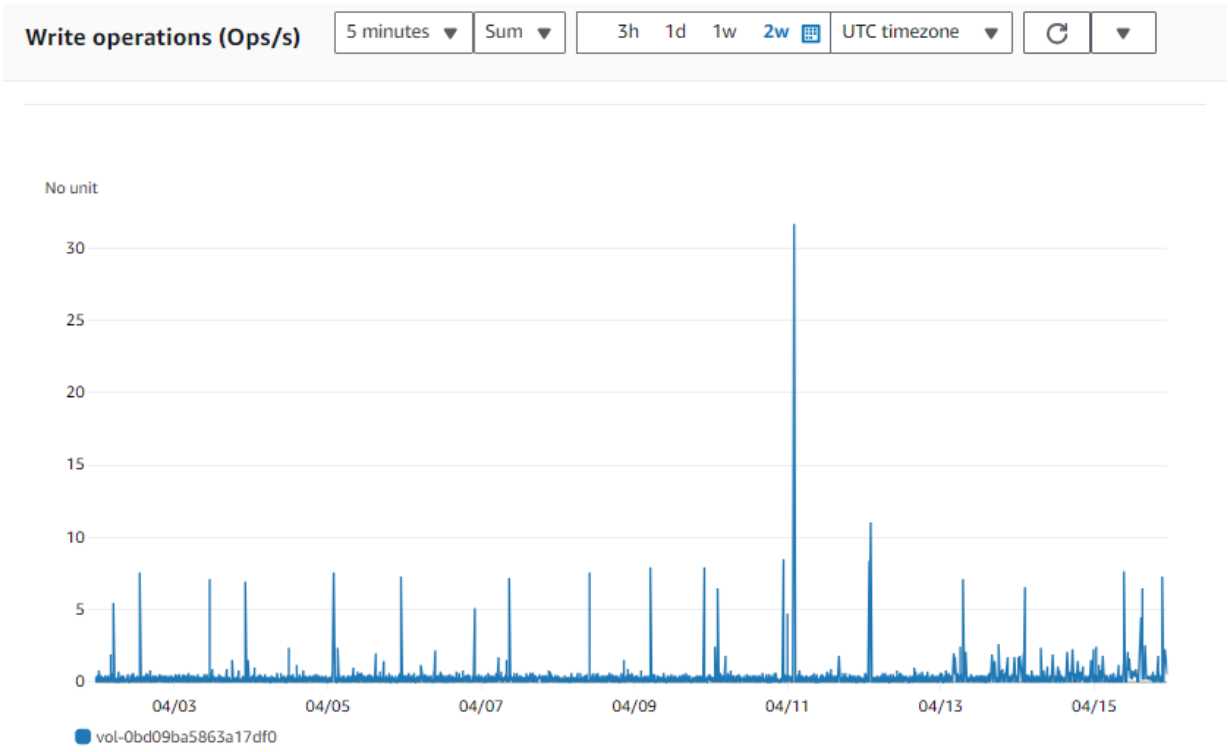


Figure B.30: Write Operations Per Second

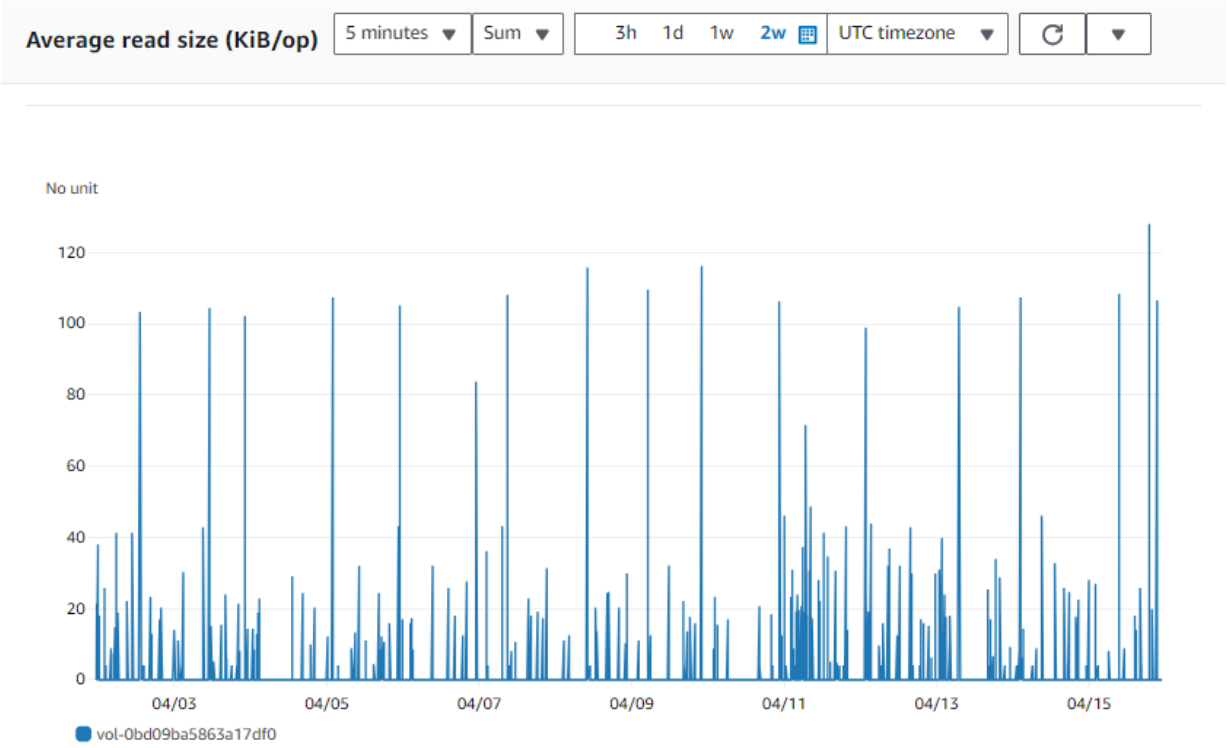


Figure B.31: Average Read Size in Kilobytes per Operation

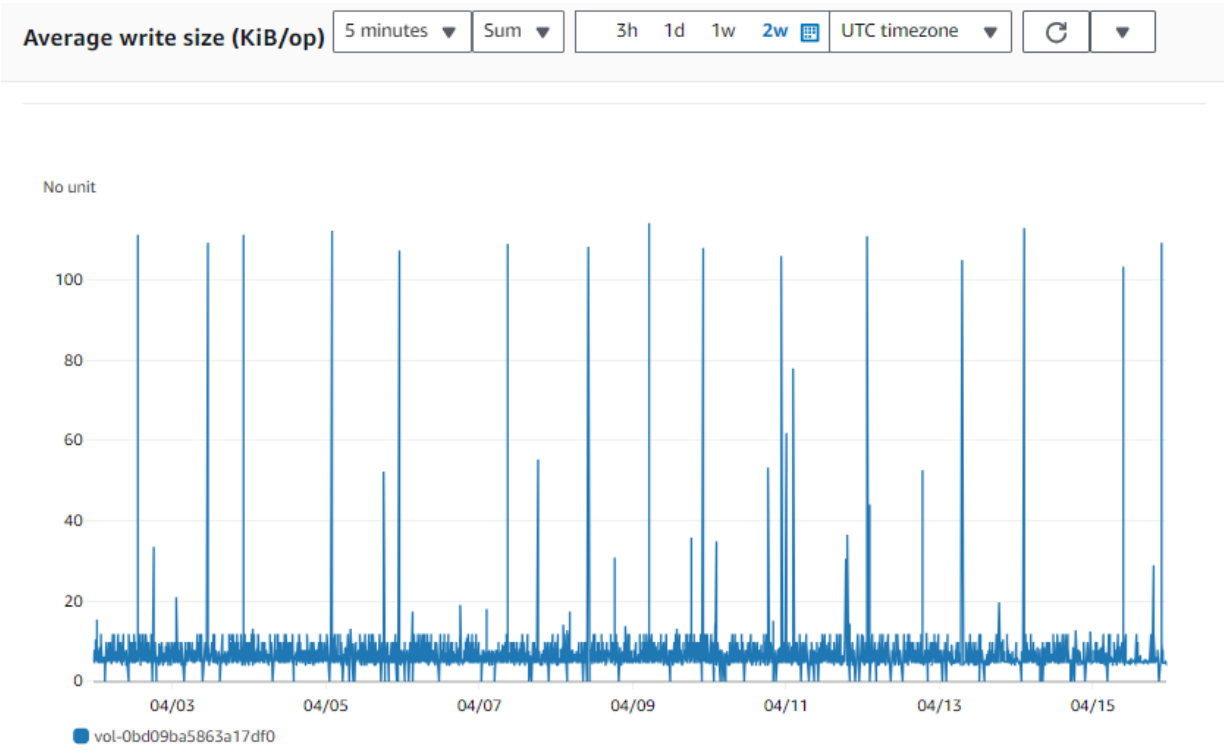


Figure B.32: Average Write Size in Kilobytes per Operation

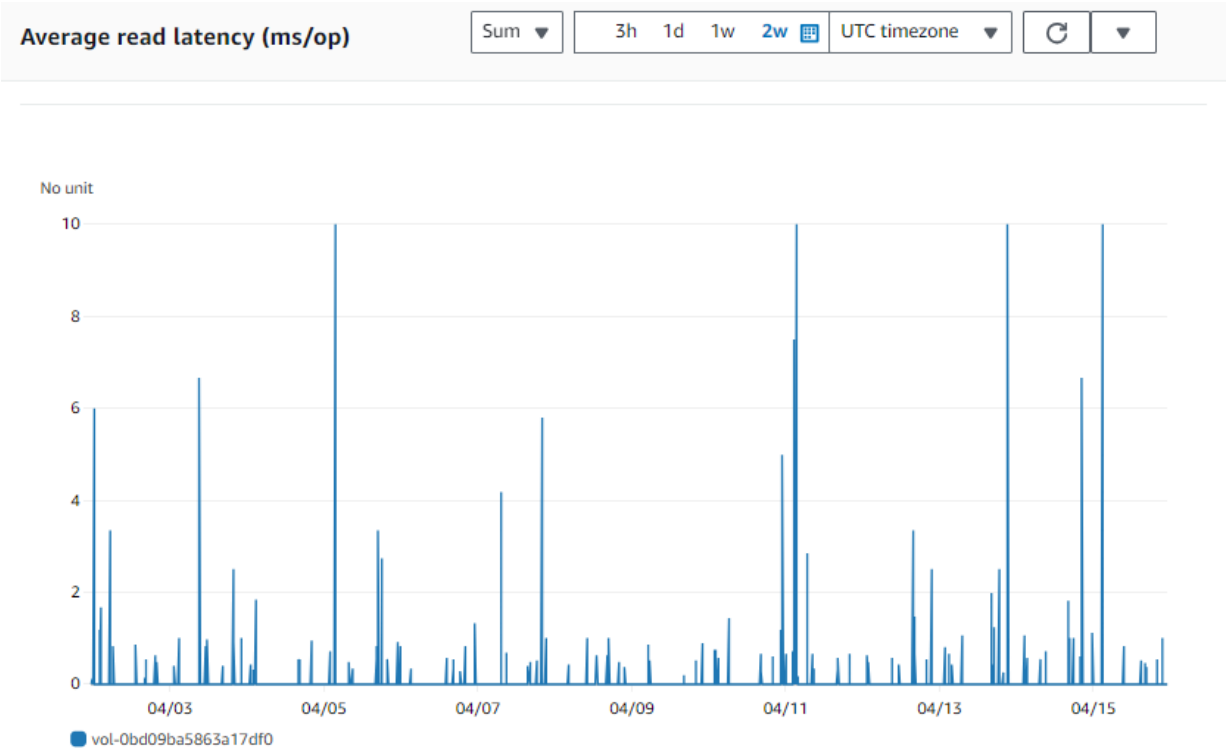


Figure B.33: Average Read latency in Milliseconds per Operation

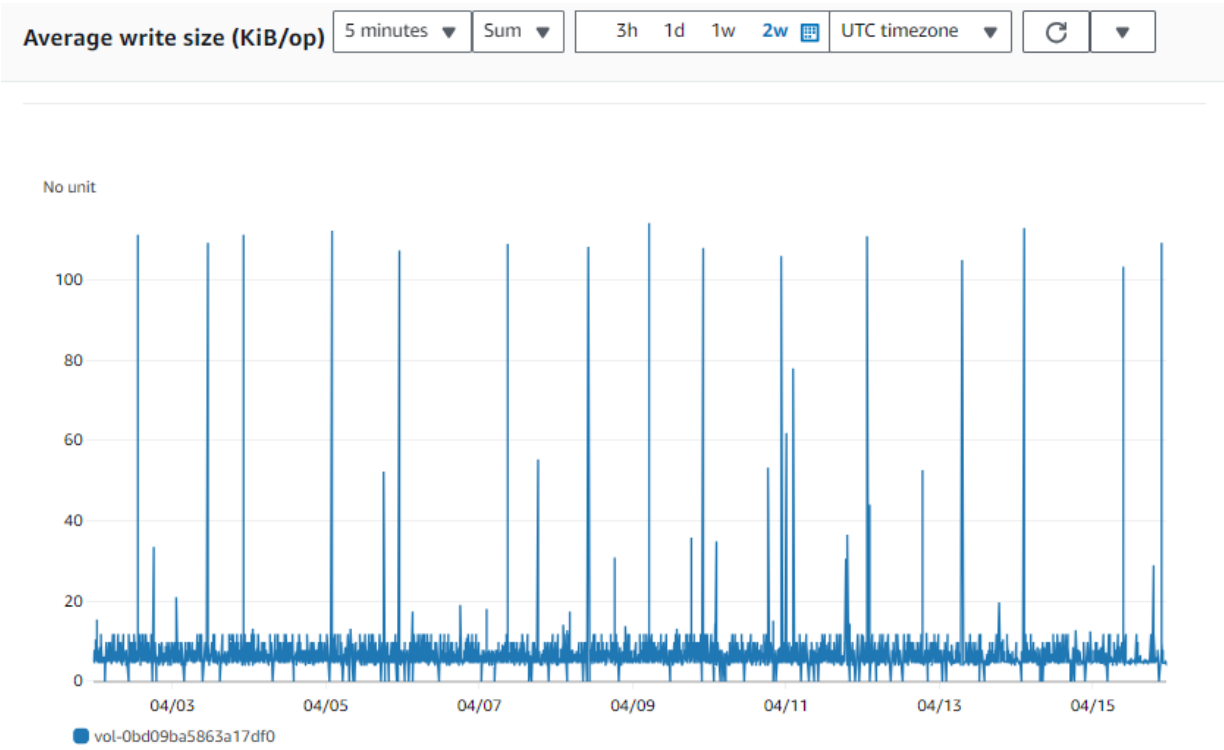


Figure B.34: Average Write latency in Milliseconds per Operation

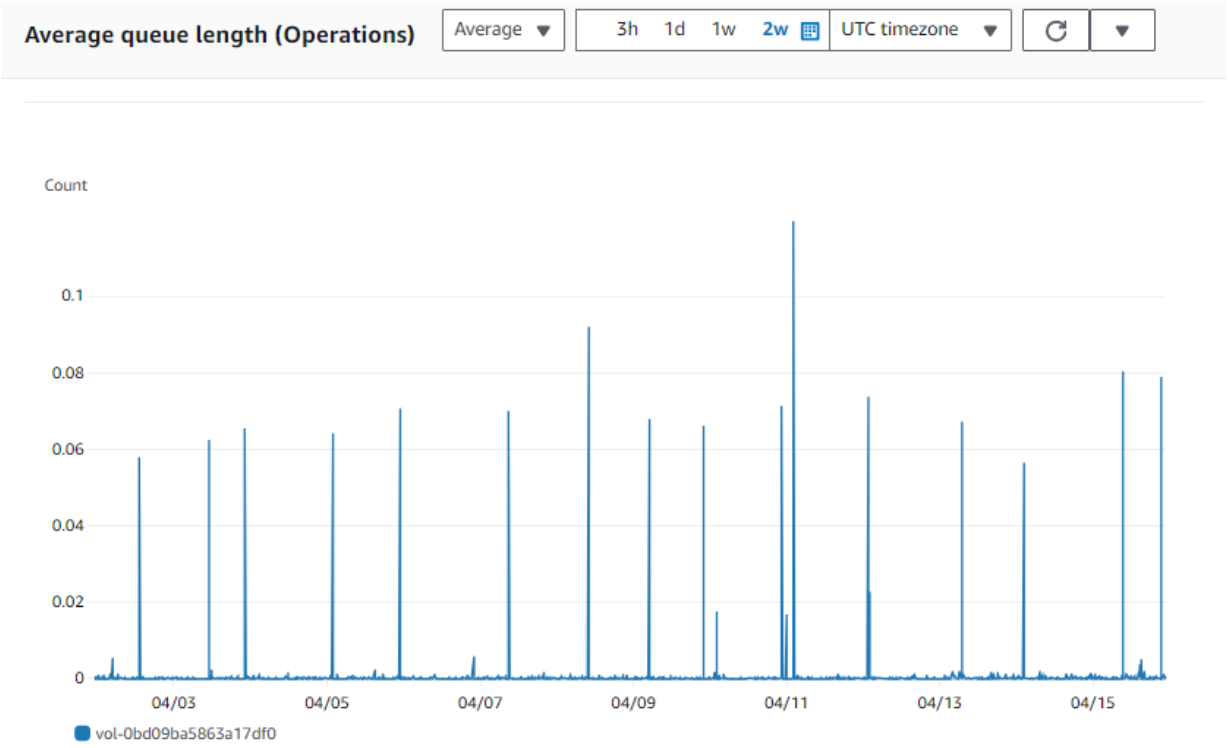


Figure B.35: Average Queue Length by Number of Operations

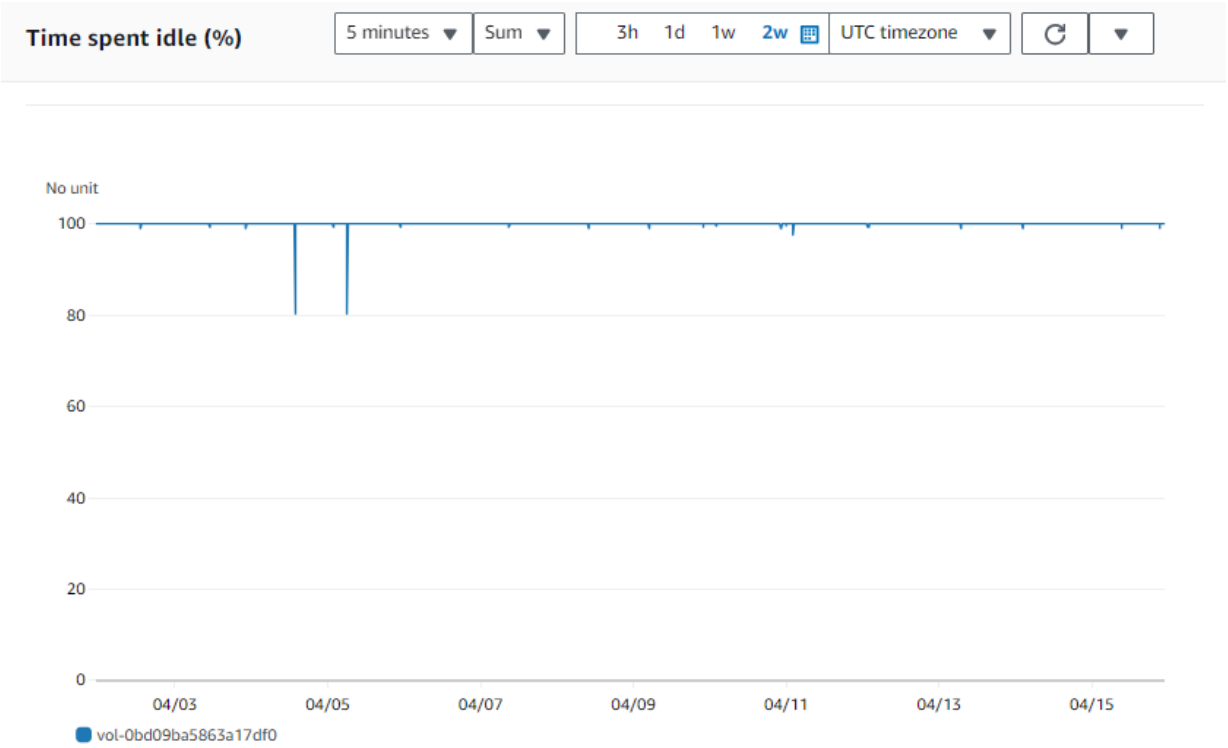


Figure B.36: Percentage of Time Spent Idle

Figure B.37: Percentage Burst Balance

B.4 Deprecated Requirements Tables

Mission Requirements:

Requirement Type	Requirement ID	Description of Requirement	Justification of Requirement	Verification of Requirement	Verification Testing Method	Requirement Validated?
Functional	REQ-M-FUN-0000	Web-app shall be operational on personal computers available to CSA Engineers and Operators. 16 gb RAM, 8 core processor, no discrete GPU requirement.	Customer/Sponsor will need flexibility to operate and test the web-app developed without losing its functionality.	The team will be constantly testing and validating the web-app software by providing subsequent version of the releases.	Demonstration/Inspection	FALSE
Functional	REQ-M-FUN-0010	Given expected inputs, i.e., image transfer order, satellite activities, the web-app will be able to compute the correct schedule.	Customer/Sponsor will need to verify that the output of the web-app meets the requirements given in the project description.	In the UI/Front-End, there will be event logs and other visual depictions to show the state of the system.	Demonstration	FALSE
Functional	REQ-M-FUN-0020	The full tech stack architecture should be dynamic enough to handle any changes to the project description requirements.	The customer/sponsor will need to be updated on the architecture used to develop the web-app to ensure it is compatible with their internal systems.	Design using SOLID (Single-responsibility, Open-Closed, Liskov substitution, Interface segration, dependence inversion) principles to ensure the software is extensible, maintainable, scalable, and modular.	End-product Testing/Analysis	FALSE
Functional	REQ-M-FUN-0030	The system must be automated but users should still be allowed to monitor the state of the system and the state of submitted orders.	Utilized by CSA engineers and analysts in the workplace through submitted requests and be able to view the state of the system.	Create event logs and state descriptions as core features of the system. Such logs and states will be relayed to the user interface.	Demonstration/Analysis	FALSE

Figure B.38: Mission Requirements

Requirement Type	Requirement ID	Description of Requirement	Justification of Requirement	Verification of Requirement	Verification Testing Method	Requirement Validated?
Ground Station	REQ-M-GS-0010	The system shall take into account that once the satellite is in view of the ground station, it shall require 5 minutes to preconfigure before uplinking and downlinking activities.	To simulate the required real-time for a simulated ground station to prepare before communciating with the satellite/s.	Delay between access timestamp and commencement of downlinking and uplinking activities.	Inspection/Analysis	FALSE
Ground Station	REQ-M-GS-0020	The system should only schedule uplinking or downlinking when ground stations can receive and send data when at an elevation angle of greater than 5 degrees.	To simulate the required constraint on the ground station and its communicate capabilities.	Estimate relative distance between ground station and satellite to adapt constraint into communications.	Inspection/Analysis	FALSE
Ground Station	REQ-M-GS-0030	The system should not send schedules that will require rates higher than what ground stations can operate on - a downlink rate of 100 Mbps & an uplink rate of 40 kbps.	Given downlink and uplink rate constraints from the CSA for ground stations.	Simulate data rate exchanges when accessing satellite to uplink images to the ground station.	End-product Testing/Inspection	FALSE
Ground Station	REQ-M-GS-0040	The system should only schedule uplinking or downlinking to one satellite that the ground station communicates with at a given access interval.	CSA given constraint to be simulated on the ground station and satellite.	System will first identify satellite it needs to communicate with based on schedule and access earliest time interval available.	End-product Testing	FALSE
Image Orders	REQ-M-GS-0050	The system should have a priority system that ground stations shall uplink schedules based on.	Priority system will ensure that the most important image orders are received as requested by the user.	Integration of priority system from 1 being the most important and 3 being the least important.	End-product Testing	FALSE

Figure B.39: Ground Station Requirements

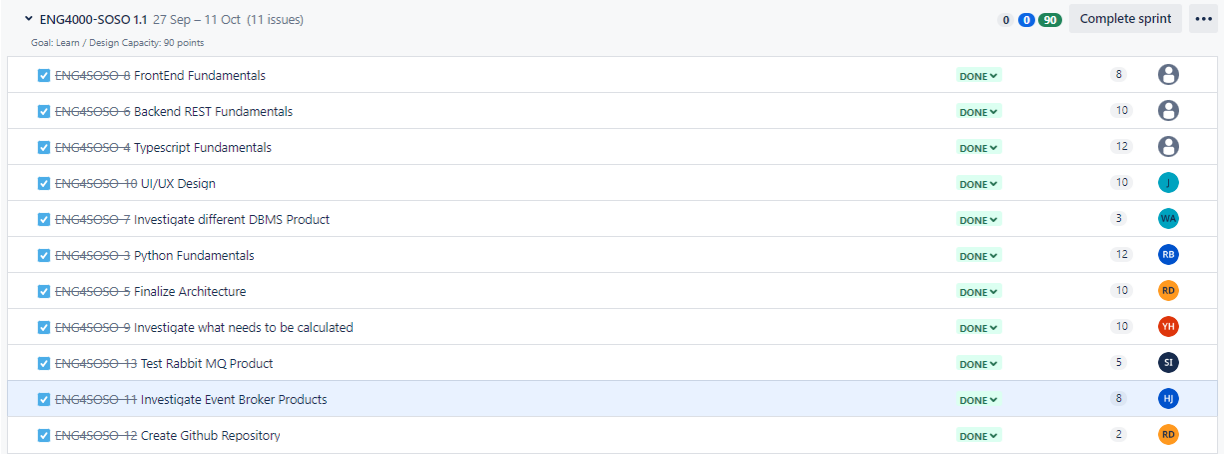
Requirement Type	Requirement ID	Description of Requirement	Justification of Requirement	Verification of Requirement	Verification Testing Method	Requirement Validated?
Satellite	REQ-M-SAT-0010	The system should be able to receive updates on the onboard storage of images based on schedule.	To ensure that the satellite doesn't overload the 32 GB storage capacity.	Based on schedule, continously update the satellite's load for images.	Demonstration/End-product Testing	FALSE
Satellite	REQ-M-SAT-0020	The system should consider satellites changing power generation under eclipse conditions.	To simulate realistic condition where the satellite will be illuminated and in eclipse during regular activities.	Simulate the predefined power limit when in sunlight and the power degradation during eclipse and its impact on the activities.	End-product Testing/Analysis	TRUE
Image Orders	REQ-M-SAT-0030	The system shall be able to estimate field of view relative to ground track and satellite when capturing images.	To categorize acceptable and unacceptable images when capturing images for the user.	Calculation between the ground track and satellite to estimate field of view and validate if image is acceptable or not.	End-product Testing/Analysis	TRUE

Figure B.40: Satellite Requirements

Requirement Type	Requirement ID	Description of Requirement	Justification of Requirement	Verification of Requirement	Verification Testing Method	Requirement Validated?
Functional	REQ-M-FUN-0110	The system should be able to receive image requests, maintenance requests and outage requests and all the constraints that are provided along with them.	Satellite operators should be able to input the requests so they can be scheduled by the system in the web-app.	Testing inputs by developers and/or users.	End-product Testing	FALSE
Functional	REQ-M-FUN-0111	The system should verify valid values for input fields	Invalid inputs might cause faulty schedules and system errors	Testing inputs inputs with type and/or logically invalid values	End-product Testing	FALSE
Functional	REQ-M-FUN-0112	The system should store all requests, scheduled or not.	All processing, access and modification uses depend on the data being persisted	write status checked in the code for each request	End-product Testing	FALSE
Functional	REQ-M-FUN-0113	The system should display requests as well as schedules upon user request	Satellite operators should be able to check what's been requested and scheduled	Can be seen on the user interface.	Demonstration	FALSE
Functional	REQ-M-FUN-0114	The system should display schedule status upon user request	Satellite operators should be able to see what has or hasn't been scheduled as well as sent to the ground station or satellite.	Can be seen on the user interface	Demonstration	FALSE
Functional	REQ-M-FUN-0115	The system should display satellite status upon user request	Satellite operators should be able to see location and satellite other satellite status	Can be seen on the user interface	Demonstration	FALSE
Functional	REQ-M-FUN-0116	The system should be able determine satellite location	scheduling depends on the satellite location in orbit	calculate satellite location and give result, can also be compared with other tools	End-product Testing	FALSE
Functional	REQ-M-FUN-0117	The system should be able to determine when a satellite comes in and out of contact with a ground station	Uplinking schedules and dowlinking images depend on proximity of satellite to ground station	Proximity checked using satellite and ground station locations	End-product Testing	FALSE
Functional	REQ-M-FUN-0118	The system should be able to add new satellite and ground stations	New assets might be added in the future	Addition of asset functionality available on the user interface.	Demonstration/Analysis	FALSE
Functional	REQ-M-FUN-0119	The system should store all necessary data about the satellites and ground stations provided as well as any new ones	Data such as satellite TLEs, ground station locations etc. are needed for calculations	Visible on database after creation	Demonstration/Analysis	FALSE
Non-Functional	REQ-M-NFUN-0110	The system should be containerized	CSA should be able to run it without the need to go through the set up process required to run locally	Docker containers of all services in the system	Inspection/Analysis	FALSE
Non-Functional	REQ-M-NFUN-0111	The system should be able to handle up to 500 requests per day, i.e., 100 per spacecraft	That many requests may need to be scheduled per day	Successfully receive and store requests	End-product Testing	FALSE

Figure B.41: Lower Level Requirements

B.5 Sprint Logs



ENG4000-SOSO 1.1 27 Sep – 11 Oct (11 issues) — 0 0 90 — Complete sprint

Goal: Learn / Design Capacity: 90 points

~~ENG4SOSO-8~~ FrontEnd Fundamentals	DONE	8	
~~ENG4SOSO-6~~ Backend REST Fundamentals	DONE	10	
~~ENG4SOSO-4~~ Typescript Fundamentals	DONE	12	
~~ENG4SOSO-10~~ UI/UX Design	DONE	10	J
~~ENG4SOSO-7~~ Investigate different DBMS Product	DONE	3	WA
~~ENG4SOSO-3~~ Python Fundamentals	DONE	12	RB
~~ENG4SOSO-5~~ Finalize Architecture	DONE	10	RD
~~ENG4SOSO-9~~ Investigate what needs to be calculated	DONE	10	YH
~~ENG4SOSO-13~~ Test Rabbit MQ Product	DONE	5	SI
~~ENG4SOSO-11~~ Investigate Event Broker Products	DONE	8	HJ
~~ENG4SOSO-12~~ Create Github Repository	DONE	2	RD

Figure B.42: Sprint 1.1 for preliminary research and solution formulation.

ENG4000-SOSO 1.2 12 Oct – 26 Oct (8 issues) — 0 0 62 — Complete sprint

Rafael: 30 Youseff: 20 Ruth: 20 Stanley: 20 Hashir: 20 James: 15 Walid: 10 Total: 115 Goal: Image Request, Scheduling Activities, Scheduler Algorithm, Project Management Tasks

~~ENG4SOSO-20~~ Satellite Calculations TLE	SERVER - SCHEDULE CA...	DONE	10	YH
~~ENG4SOSO-21~~ Ground Stations Positional Data Calculations	SERVER - SCHEDULE CA...	DONE	10	YH
~~ENG4SOSO-23~~ Research current implementation of similar web-apps (Non-engineering)		DONE	2	J
~~ENG4SOSO-18~~ Investigate server needs for NGINX vs. Kubernetes		DONE	5	HJ
~~ENG4SOSO-14~~ Design Database		DONE	15	HJ
~~ENG4SOSO-33~~ Design and Implement Image Request Endpoint(s) in Event-Relay-API	SERVER - IMAGE MANA...	DONE	5	RD
~~ENG4SOSO-34~~ Design and Implement Activity Request Endpoint(s) in Event-Relay-Api	SERVER - SATELLITE AC...	DONE	5	RD
~~ENG4SOSO-16~~ Setup Backend: Server, Directories, Files, and RabbitMQ		DONE	10	RD

Figure B.43: Sprint 1.2 for advanced research and solution refinement.

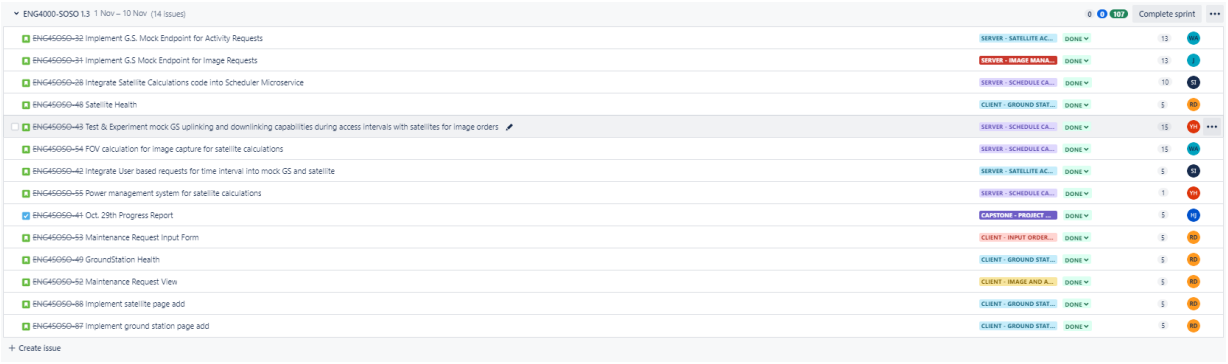


Figure B.44: Sprint 1.3 for implementation of the SatOpsMQ basic framework.

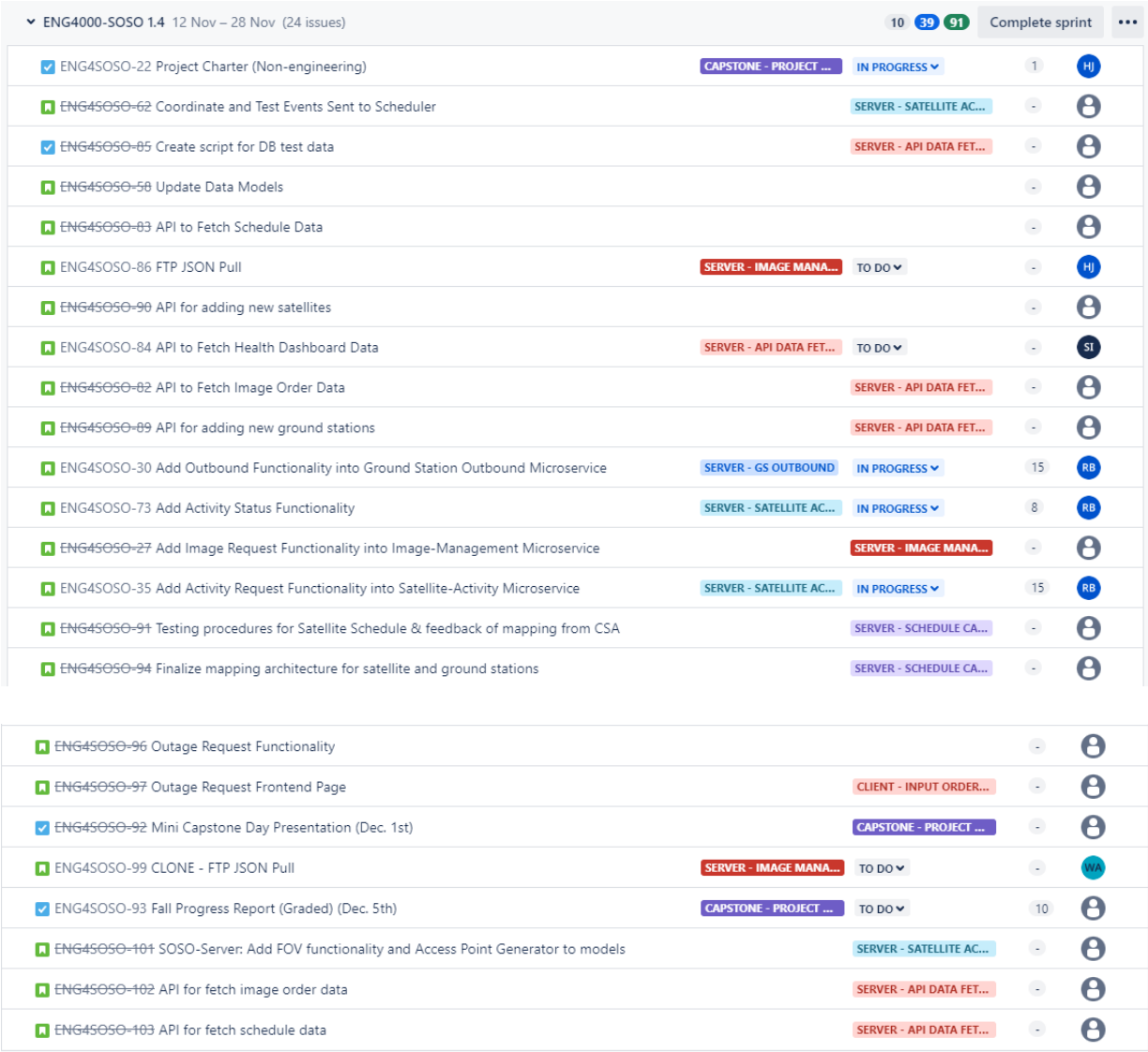


Figure B.45: Sprint 1.4

Sprint 1.5 (Jira)

Project: ENG 4000: Satellite Operations Services Optimizer
: ENG4000-SOSO 1.5
Sorted by: Created descending
1–23 of **23** as at: **03/Mar/24 9:07 PM**

T	Key	Summary	Assignee	Reporter	Status	Sprint
	ENG4SOSO-128	Translate Recurrence Attributes of Img Req to Img Order	Walid AlDari	Walid AlDari	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-127	Research Concepts	Ruth Bezabeh	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-126	Heuristic Algorithm Attempt	Stanley Ihesiulo	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-124	Algorithm Attempt - RL, GA, Graphs	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-123	Integrate Orbit Maps into frontend	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-121	Implement 2D & 3D Orbit Maps	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-114	Design Logo	jamesmql	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-113	Parse image order data for parent and child relationship	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-112	Repositories Polishing	Walid AlDari	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-110	Implementation of Topic Exchanges	Stanley Ihesiulo	Rafael Dolores	DONE	ENG4000-SOSO 1.5
	ENG4SOSO-108	Familiarize with FrontEnd	*Unassigned*	Rafael Dolores	DONE	ENG4000-SOSO 1.5

Figure B.46: Sprint 1.5 part-1

T	Key	Summary	Assignee	Reporter	Status	Sprint
☑	ENG4SOSO-104	UI Improvement	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
▣	ENG4SOSO-99	CLONE - FTP JSON Pull	Walid AlDari	Hashir Jamil	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
☑	ENG4SOSO-93	Fall Progress Report (Graded) (Dec. 5th)	*Unassigned*	Youssef Hany	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
▣	ENG4SOSO-86	FTP JSON Pull	Walid AlDari	Hashir Jamil	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
▣	ENG4SOSO-84	API to Fetch Health Dashboard Data	Stanley Ihesiulo	Rafael Dolores	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
▣	ENG4SOSO-73	Add Activity Status Functionality	Ruth Bezabeh	Ruth Bezabeh	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
▣	ENG4SOSO-72	UI Mockups	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
▣	ENG4SOSO-71	Tool Familiarization	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
▣	ENG4SOSO-70	Orbital Basics Research	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.5
▣	ENG4SOSO-69	Setup Websockets Connection	Stanley Ihesiulo	Rafael Dolores	DONE	ENG4000-SOSO 1.5
▣	ENG4SOSO-35	Add Activity Request Functionality into Satellite-Activity Microservice	Ruth Bezabeh	Rafael Dolores	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5
☑	ENG4SOSO-22	Project Charter (Non-engineering)	Hashir Jamil	Youssef Hany	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.5

Figure B.47: Sprint 1.5 part-2

Sprint 1.6 (Jira)
Project: ENG 4000: Satellite Operations Services Optimizer
: ENG4000-SOSO 1.6
Sorted by: Created descending
1–11 of **11** as at: **03/Mar/24 8:47 PM**

T	Key	Summary	Assignee	Reporter	Status	Sprint
☑	ENG4SOSO-140	Peer Review for Winter Semester 1	*Unassigned*	Youssef Hany	DONE	ENG4000-SOSO 1.6
☑	ENG4SOSO-139	Meeting Notes	*Unassigned*	Youssef Hany	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-137	As a developer, I want to have well defined scoring functions to provide genetic algorithm optimization	Stanley Ihesiulo	Rafael Dolores	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-136	As a satellite operator, I want to updates on satellite eclipses and ground station to satellite contact opportunities	Hashir Jamil	Rafael Dolores	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-135	As an overall user of the system, I want satellite schedules to be automatically generated.	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-133	As a developer, I want to ensure orders are parsed into parent and children accurately	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-132	As a developer I want to use particle swarm optimization to produce schedules	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-129	As a developer, I want to define detailed performance metrics to evaluate the system	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-120	As a developer, I want to have an automated continuous integration & continuous delivery pipeline for the system's version control actions	jamesmql	Rafael Dolores	DONE	ENG4000-SOSO 1.6
🔖	ENG4SOSO-117	As a developer, I want all Individual units in event relay and image management to produce the desired output	Walid AlDari	Rafael Dolores	DONE	ENG4000-SOSO 1.6
☑	ENG4SOSO-107	As a developer, I want to have a more readable codebase for the event-relay API microservice	Hashir Jamil	Rafael Dolores	DONE	ENG4000-SOSO 1.6

Figure B.48: Sprint 1.6

Sprint 1.7 (Jira)

Project: ENG 4000: Satellite Operations Services Optimizer
: ENG4000-SOSO 1.7
Sorted by: Created descending
1–12 of **12** as at: **03/Mar/24 8:46 PM**

T	Key	Summary	Assignee	Reporter	Status	Sprint
	ENG4SOSO-152	As a developer, I want to be able to deploy the client side module as an open source container	jamesmql	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-151	As a developer, I want full integration of the deterministic scheduling algorithm in the server side code	Stanley Ihesiulo	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-150	Meeting Notes	*Unassigned*	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-149	As a developer I want to use particle swarm optimization to produce schedules	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-148	As a satellite operator, I want satellite and ground station schedules to be sent to the ground stations	Ruth Bezabeh	Rafael Dolores	DONE	ENG4000-SOSO 1.4, ENG4000-SOSO 1.7
	ENG4SOSO-146	As a satellite operator, I want to be given hourly updates on satellite eclipses and ground station to satellite contact opportunities	Hashir Jamil	Rafael Dolores	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-145	As an imaging customer, I want to send orders that have recurrences built in to be handled by the systems automated nature	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-144	As a developer, I want to select a hosting platform and purchase a domain based on a trade study	Walid AlDari	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-143	As a member of the SatOpsMQ team, I want user stories to reflect user requirements	Ruth Bezabeh	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-142	As a satellite operator, I want to view the health of ground station and satellite assets	Hashir Jamil	Rafael Dolores	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-138	As a satellite operator, I want detailed information regarding power constraints on satellites	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.7
	ENG4SOSO-131	As a developer, I want to define and document a well formed testing environment	jamesmql	Youssef Hany	DONE	ENG4000-SOSO 1.7

Figure B.49: Sprint 1.7

Sprint 1.8 (Jira)

Project: ENG 4000: Satellite Operations Services Optimizer
: ENG4000-SOSO 1.8
Sorted by: Created descending
1–5 of **5** as at: **03/Mar/24 8:46 PM**

T	Key	Summary	Assignee	Reporter	Status	Sprint
	ENG4SOSO-159	As a developer I want the code to reflect the updated database schema and utilize App-Config	Walid AlDari	Walid AlDari	DONE	ENG4000-SOSO 1.8
	ENG4SOSO-158	As a Developer I want all Individual units in the backend to produce the desired output	Walid AlDari	Youssef Hany	DONE	ENG4000-SOSO 1.8
	ENG4SOSO-156	As the satellite operator I want new activities to affect the existing schedule only with in a limited window	Rafael Dolores	Rafael Dolores	DONE	ENG4000-SOSO 1.8
	ENG4SOSO-155	As the Developer I want to keep track of bugs and Issues	Walid AlDari	Youssef Hany	DONE	ENG4000-SOSO 1.8
	ENG4SOSO-153	As a Developer I want to be able to create orders for testing purposes	Youssef Hany	Youssef Hany	DONE	ENG4000-SOSO 1.8

Figure B.50: Sprint 1.8

Sprint 1.9 (Jira)

Project: ENG 4000: Satellite Operations Services Optimizer
: ENG4000-SOSO 1.9
Sorted by: Created descending
1–**8** of **8** as at: **03/Mar/24 8:41 PM**

T	Key	Summary	Status	Sprint	Assignee
	ENG4SOSO-168	As a satellite operator I want to receive schedules by only uploading Image orders or inputting satellite activity orders	IN PROGRESS	ENG4000-SOSO 1.9	Stanley Ihesiulo
	ENG4SOSO-167	As a satellite operator I want assets to be utilized equally	IN PROGRESS	ENG4000-SOSO 1.9	Rafael Dolores
	ENG4SOSO-166	As a satellite operator I want image captures scheduled only when the area of interest is in the satellite's FOV	IN PROGRESS	ENG4000-SOSO 1.9	Youssef Hany
	ENG4SOSO-163	As a satellite operator I want generated schedules to prioritize optimization metrics	IN PROGRESS	ENG4000-SOSO 1.9	Youssef Hany
	ENG4SOSO-161	As a member of the satopsmq team I want user stories to reflect user requirements	IN PROGRESS	ENG4000-SOSO 1.9	Ruth Bezabeh
	ENG4SOSO-157	As a satellite operator I want the data I see to be updated	IN PROGRESS	ENG4000-SOSO 1.9	Walid AlDari
	ENG4SOSO-118	As a satellite operator I want to receive correct outputs for every input	IN PROGRESS	ENG4000-SOSO 1.9	jamesmql
	ENG4SOSO-116	As a developer I want communication between services and components to be accurate	IN PROGRESS	ENG4000-SOSO 1.9	jamesmql

Figure B.51: Sprint 1.9

Appendix C

Additional Tables

C.1 Input Parameter Data Descriptions

Parameter	Description
Region to Image	A point defined by latitude and longitude on the Earth's surface that must be imaged.
Priority	The relative importance of an image, with higher priority images taking precedence in the event of a conflict.
Image Type	The type of image requested, with options including Spotlight (high-resolution), Medium, and Low resolution, each covering a specific area and requiring a certain amount of time to complete.
Imaging Time	The time range within which the image must be taken.
Delivery Time	When the image should be downlinked to the ground.
Revisit Time	The frequency of retaking the same image if needed.

Table C.1: Image Request Parameters

Parameter	Description
Target	The satellite that requires maintenance.
Window	The time period during which the maintenance must be completed.
Duration	How long the maintenance task will take.
Repeat Cycle	How often the maintenance must be repeated, including frequency and number of repetitions.
Payload Operations	Indicates whether payload operations can continue during maintenance.

Table C.2: Satellite Activities Request Parameters

Activity Type	Description
Payload Diagnostic Activity Maintenance	Regular checks and maintenance tasks performed on the satellite's payload to ensure it functions correctly and efficiently.
Orbit Parameter Update	Adjustments or updates made to the satellite's orbit parameters to maintain or change its trajectory and positioning.
Orbit Maintenance	General maintenance activities to ensure the satellite remains in its intended orbit. This might include maneuvers to correct any deviations or drifts.
Memory Scrub	A process of checking and correcting errors in the satellite's memory systems to prevent data corruption and ensure reliable operation.

Table C.3: Descriptions of Satellite Maintenance Activity Types

Parameter	Description
Target	The asset experiencing an outage.
Start Time	When the outage begins.
End Time	When the outage ends.

Table C.4: Outage Request Parameters

Appendix D

Meeting Minutes & Self Evaluation

D.1 Meeting Minutes

All meeting minutes can be read and reviewed at the following link for the Fall Semester: SatOpsMQ Meeting Minutes (Fall Semester).

All meeting minutes can be read and reviewed at the following link for the Winter Semester: SatOpsMQ Meeting Minutes (Winter Semester).

D.2 Self Evaluation

Below is set tables showing the reported self-evaluation criteria. This matches the rubric in section 2.5 of the Final Year Project Progress, Mid-Project and Final Project Deliverables Document. The tables below state the rankings we feel we have earned and the references report sections that detail the fulfilled obligation(s) as per the rubric. These self evaluations are also present at the end of volume 2 and volume 3.

Criterion	Ranking	Justification
Articulate the Design Problem To Be Solved	Exceeding	Reference1, Reference2
Feasibility of Proposed Approach	Exceeding	Reference1
Formulate a Strategy for Designing an Engineering Solution	Exceeding	Reference1
Decompose a Complex System into Subsystems	Exceeding	Reference1
Criteria to Compare & Contrast	Exceeding	Reference1, Reference2,
Justify Strengths & Limitations, Make Recommendations	Meeting	Reference1, Reference2
Apply Engineering, Fundamentals, Theories, Practices	Exceeding	Reference1
Develop Creative, Innovative Solution	Exceeding	Reference1, Reference2, Reference3,
Review Project Sustainability, Impact, Unintended Consequences	Exceeding	Reference1

Table D.1: Self-Evaluation General Section: Engineering Design.

Criterion	Ranking	Justification
Ensure State Appropriate for Formal Testing	Exceeding	Reference1
Develop Test Plan for System Verification/Validation	Meeting	Reference1
Develop Test Procedures and Ensure Test Equipment Understood	Meeting	Reference1, Reference2, Reference3
Monitor and Address Deviations From the Test Plan	Meeting	Reference1
Ensure System Configuration, Failure Tracking Defined and Implemented	Exceeding	Reference1
Review System Test Results for Validity	Exceeding	Reference1, Reference2, Reference3
Document and Track Test Failures Appropriately	Exceeding	Reference1, Reference2

Table D.2: Self-Evaluation General Section: Test Related Criteria

Criterion	Ranking	Justification
Identify Driving Requirements	Exceeding	Reference1, Reference2
Understand Rationale for Requirements	Exceeding	Reference1, Reference1, Reference2
Analyze Requirements Wording	Meeting	Reference1, Reference1
Analyze Requirements Scoping	Meeting	Reference1, Reference2, Reference3
Analyze Requirements Compliance	Exceeding	Reference1

Table D.3: Self-Evaluation For a Well-Defined Project Section

Criterion	Ranking	Justification
Develop Agile Roadmap	Exceeding	Reference1
Breakdown, Prioritize Product Features Into Increments	Exceeding	Reference1, Reference2, Reference3
Assign, Track, Asses Team Tasks & Performance	Exceeding	Reference1, Reference2, Reference3, Reference4, Reference5, Reference6, Reference7, Reference8
Review Sprint Completion Status	Exceeding	Reference1, Reference2
Review Sprint Effectiveness	Exceeding	Reference1, Reference2, Reference3, Reference4
Ensure Appropriateness of Sprint Plan	Exceeding	Reference1 Reference2, Reference3,
Review Project Costs	Exceeding	Reference1, Reference2
Obtain Stakeholder Feedback on Releases To Inform Next Development	Exceeding	Reference1, Reference2, Reference3

Table D.4: Self-Evaluation For Agile Project Management Section

Criterion	Ranking	Justification
Project Schedule	Exceeding	Reference1
Sprint Plans	Exceeding	Reference1
Final Project Costs	Exceeding	Reference1
Preliminary Business Case	Exceeding	Reference1
Lessons Learned/Failure Report	Exceeding	Reference1, Reference2

Table D.5: Self-Evaluation For Agile Project Management Section Delvierable List List