Final Project Report

Project Title:
Space Collision Prevention System

ECSE-458D1: Capstone Term 2

Team DP28
Team members:
Daniel Makhlin, daniel.makhlin@mail.mcgill.ca (ID:261 035 284)
Yassine Meliani, yassine.meliani@mail.mcgill.ca (ID:261 035 296)
Shady Guindi, shady.guindi@mail.mcgill.ca (ID:261 026 626)
Radu Petrescu, radu.petrescu@mail.mcgill.ca (ID:261 051 351)

Project advisors:
Nathaniel Cziranka-Crooks (CSA)
Email: nathaniel.cziranka-crooks@asc-csa.gc.ca

Patrick Irvin (CSA)
Email: patrick.irvin@asc-csa.gc.ca

Lili Wei (McGill)
Email: lili.wei@mcgill.ca

Abstract

The increasing congestion in Earth's orbit has made collision avoidance a critical concern for space agencies worldwide. This project aims to develop a comprehensive visualization tool that serves both frontline and technical operators at the CSA, enabling quick risk assessment and informed decision-making for collision avoidance maneuvers. Over the course of the year, we successfully developed and refined a functional prototype for the Space Collision Prevention System. In the first semester, we focused on establishing the system's core architecture, implementing secure authentication with role-based access control, interactive dashboards for both operator types, and advanced visualization components such as timeline-based graphs and 3D spatial representations using CesiumJS. Built using an agile development approach with React and TypeScript on the frontend, Express for the backend, and Supabase for database management, the system was fully containerized with Docker for ease of deployment. In the second semester, we expanded the system's capabilities with several new features: dynamic filtering for CDMs and space events, new visualizations, and real-time alerting with customizable threshold values. We also enabled live data feeding into the filtering system, pagination to display a limited number of events per page, and introduced interaction mechanisms between frontline and technical operators via alert-based communication. These functional upgrades were supported by rigorous backend testing, including unit tests, to ensure reliability and system stability. Throughout the year, regular validation meetings with the Canadian Space Agency helped ensure continuous alignment with stakeholder needs and informed our iterative development process using mock CDM data.

Acknowledgments

We would like to express my sincere gratitude to Professor Lili Wei for her invaluable guidance and support throughout this project. Her deep expertise and insightful feedback have significantly shaped this work. We are equally thankful to our project advisors, Nathaniel and Patrick from the Canadian Space Agency, whose dedication, practical insights, and mentorship were instrumental in bringing this project to fruition. Their combined encouragement and constructive criticism pushed us to explore new perspectives and exceed expectations. This work would not have been possible without their support and mentorship.

Table of contents

Nomenclature

CSA: Canadian Space Agency
NASA: National Aeronautics and Space Administration
CCSDS: Consultative Committee for Space Data Systems
CDM: Conjunction Data Message
CARA: Conjunction Assessment Risk Analysis
DB/db: Database
RSS: Residual Sum of Squares
RBAC: Role-based access control
API: Application Programming Interface

Introduction

The primary focus of our project is to develop a visualization tool tailored for technical and frontline operators at the Canadian Space Agency (CSA). This tool addresses the pressing need for efficient and accurate interpretation of Conjunction Data Messages (CDMs) issued by the US Space Defense Squadrons [1]. CDMs serve as critical warnings about potential collisions between objects in space, such as satellites or debris.

The overarching goal of our project is twofold: first, to enable frontline operators to quickly assess the severity of a potential collision and decide whether it warrants further attention; and second, to provide technical operators with advanced tools and visualizations that assist in planning and executing collision-avoidance maneuvers. By achieving these objectives, we aim to improve decision-making efficiency and reduce the likelihood of catastrophic events in space.

This project holds significant importance due to the increasing congestion in Earth's orbit and the growing reliance on satellite technologies for communication, navigation, and scientific research. A collision in space could trigger a cascade of debris, commonly referred to as the Kessler Syndrome, which would jeopardize the functionality of critical infrastructure and hinder future space exploration [5]. By empowering CSA operators with a robust and intuitive tool, we not only enhance operational efficiency but also contribute to the long-term sustainability of space activities.

The tool is designed with a strong emphasis on usability and functionality. It incorporates statistical and predictive features to aid technical operators in evaluating potential collisions and planning effective mitigation strategies. Frontline operators, on the other hand, benefit from a streamlined interface that allows them to quickly dismiss low-risk CDMs, saving valuable time and resources. Ultimately, this visualization tool bridges the gap between complex technical data and actionable insights, ensuring that the CSA can respond promptly and effectively to potential threats in space.

Background

The development of the visualization tool required us to delve into several theoretical and technical domains to ensure a comprehensive understanding of the problem space and the implementation of effective solutions. This section outlines the critical areas of knowledge we acquired and how they contribute to the project.

Conjunction Data Messages (CDMs)

We had to familiarize ourselves with Conjunction Data Messages (CDMs), which are the primary medium for communicating potential collisions between objects in space. CDMs are generated during Conjunction Assessment (CA) screenings, which identify potential conjunction threats by determining whether objects in space penetrate a volumetric region around a protected asset, such as a NASA scientific satellite [2]. The CDMs include conjunction summary parameters, the states and covariances at the two objects' time of closest approach (TCA), and amplifying information about the orbit determination used to generate these states and covariances. Understanding the structure and significance of CDMs was essential to designing a tool that accurately interprets and visualizes this information [3].

It is important to note that CDMs are not collision warnings. Instead, they are proximity alerts, indicating situations where a serious close approach is possible. They serve as the starting point for collision risk assessment, which combines likelihood and consequence analysis to evaluate whether mitigation actions are necessary [3].

3D Probabilistic Spaces and Covariance Calculations

To effectively model and visualize potential collision scenarios, we studied how 3D probabilistic spaces are calculated. This involved learning about covariance matrices and their role in representing the uncertainty in an object's position. Specifically, we explored how to compute the Root Sum of Squares (RSS) from covariance values to quantify the combined positional uncertainty of two objects. These calculations are critical for accurately determining the likelihood of a collision (probability of collision or Pc), which represents the likelihood that the actual miss distance between two objects will be smaller than their combined size, thus precipitating a collision [4].

We also learned about the thresholds for operational attention and mitigation actions, such as NASA CARA's standard that Pc values greater than 1E-07 merit operational attention, and values greater than 1E-04 require mitigation actions. These principles guided the statistical and predictive tools implemented in our application [3].

Maneuver Modeling and Trade Space Analysis

To support operators in mitigating conjunction risks, we studied how orbital maneuvers are proposed, evaluated, and modeled. When a Conjunction Data Message indicates an increasing probability of collision, technical operators often begin exploring a range of candidate maneuvers. These typically involve applying a small change in velocity (ΔV) to the primary object at various times before the predicted Time of Closest Approach (TCA).

The change in velocity is usually applied in the direction of the satellite's current velocity vector, yielding a new velocity and a slightly altered orbital trajectory. By propagating this new state forward in time, we estimate how the satellite's position will evolve. The resulting position and velocity are then compared to those of the secondary object to compute updated relative motion and a new miss distance.

This process is repeated across a grid of ΔV values and lead times (e.g., 0.01 m/s increments from –0.10 to +0.10 m/s, at 15-minute intervals up to 24 hours before TCA), allowing us to generate a maneuver tradespace. This tradespace visually maps how different maneuvers affect miss distance and, by extension, collision probability. The outputs help operators identify maneuvers that are both effective and fuel-efficient.

Importantly, not all maneuvers that look good on paper are viable in practice. Real-world operational constraints—such as fuel limitations, satellite responsiveness, ground station availability, mission priorities, and notification delays—can restrict when and how maneuvers are performed. Although earlier maneuvers are typically more efficient, they may not be executable due to these constraints. Our modeling approach balances theoretical insight with operational realism, giving CSA operators the information they need to make timely and practical decisions.

3D Visualization with CesiumJS

Creating an intuitive and interactive visualization required us to gain proficiency in 3D web visualization technologies, particularly CesiumJS. This library allowed us to render 3D models of Earth and orbital objects, incorporating real-time data and precise spatial positioning. We learned how to integrate data from CDMs into CesiumJS to create a dynamic and informative visual representation of potential collision scenarios. By visualizing predicted ephemerides and associated uncertainties, our tool provides operators with a clear and actionable view of conjunction events.

Requirements and Constraints

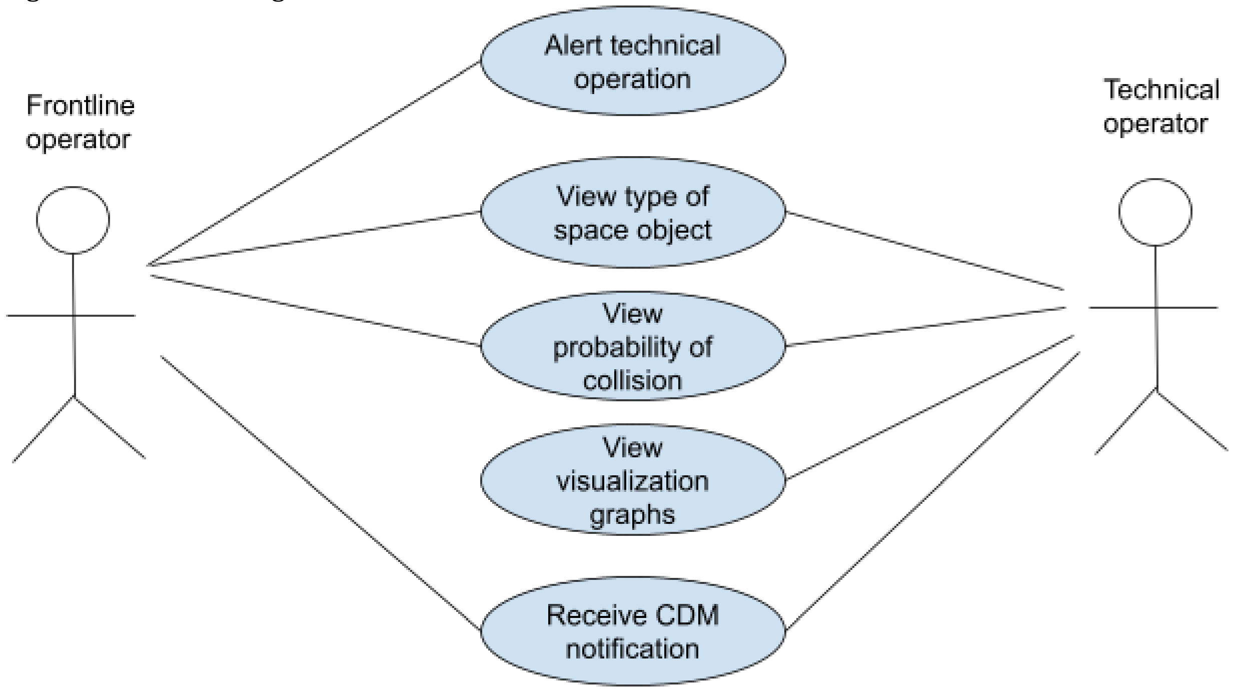
- The system shall allow secure login with corporate email or username/password.
- The system shall allow account creation.
- The system shall have accounts of two types, frontline operators and technical operators.
- The system shall periodically fetch data to update the dashboards.
- The system shall allow frontline operators to view a list of CDMs which are relevant to the specific user.
- The system shall allow frontline operators to select relevant CDMs.
- The system shall allow the frontline operators to filter CDMs and sort by satellite and agency.
- The system shall display an alert message to frontline operators if there are new unviewed CDMs since their most recent login.
- The system shall allow the frontline operator to notify a technical operator of his choosing about a particular CDM.
- The system shall allow real-time and offline alerting of the users.
- The system shall allow frontline operators to view basic information about the CDM on the CDM dashboard.
- The system shall allow frontline operators to see more specific information about the CDM by clicking on the specific CDM card.
- The system shall display frontline operators the important CDM fields (see below) consistently for each CDM card.
- The system shall allow technical operators to view Events, collection of multiple CDMs
- The system shall allow technical operators to click on an event to see more info.
- The system shall allow technical operators to see graphs of type (to be decided) based on the event.
- The system shall allow technical operators to be able to see individual CDM raw data as well as the graphs associated with these specifically.
- The system shall allow technical operators to view a 3d visualization of the probability space through time.
- The system shall allow technical operators to view a timeline that can be moved to see the probable trajectory of the objects through time.
- The system shall be containerized and deployable on the CSA's premise
- The system shall allow technical operators to input or simulate maneuver scenarios and visualize their impact on the probability of collision.
- The system shall expose a secure API for external applications to query CDM data and event details.
- The system shall allow authorized external systems to submit or receive planned maneuver data through the API.
- The system shall support live data feeding into the filtering system, dynamically updating the UI as new CDMs arrive.
- The system shall implement pagination, limiting the number of displayed CDMs/events to 12 per page.
- The system shall allow real-time user-to-user interaction, enabling frontline and technical operators to communicate directly via the alert system.
- The system shall include backend unit testing to ensure stability and correctness of core functionality.
- The system shall be tested for integration between frontend notifications and backend alerting components.

Design and Results

Over the course of the academic year, our team developed a functional and evolving prototype of the Space Collision Prevention System through an iterative, sprint-based approach guided by weekly collaboration with the Canadian Space Agency (CSA). The project was divided into two major phases: the first semester, which focused on establishing a robust foundation through research and system design, and the second semester, which emphasized feature expansion, performance optimization, and enhanced user interactivity. Together, these phases brought the system from a conceptual model to a deployable prototype aligned with CSA's operational needs.

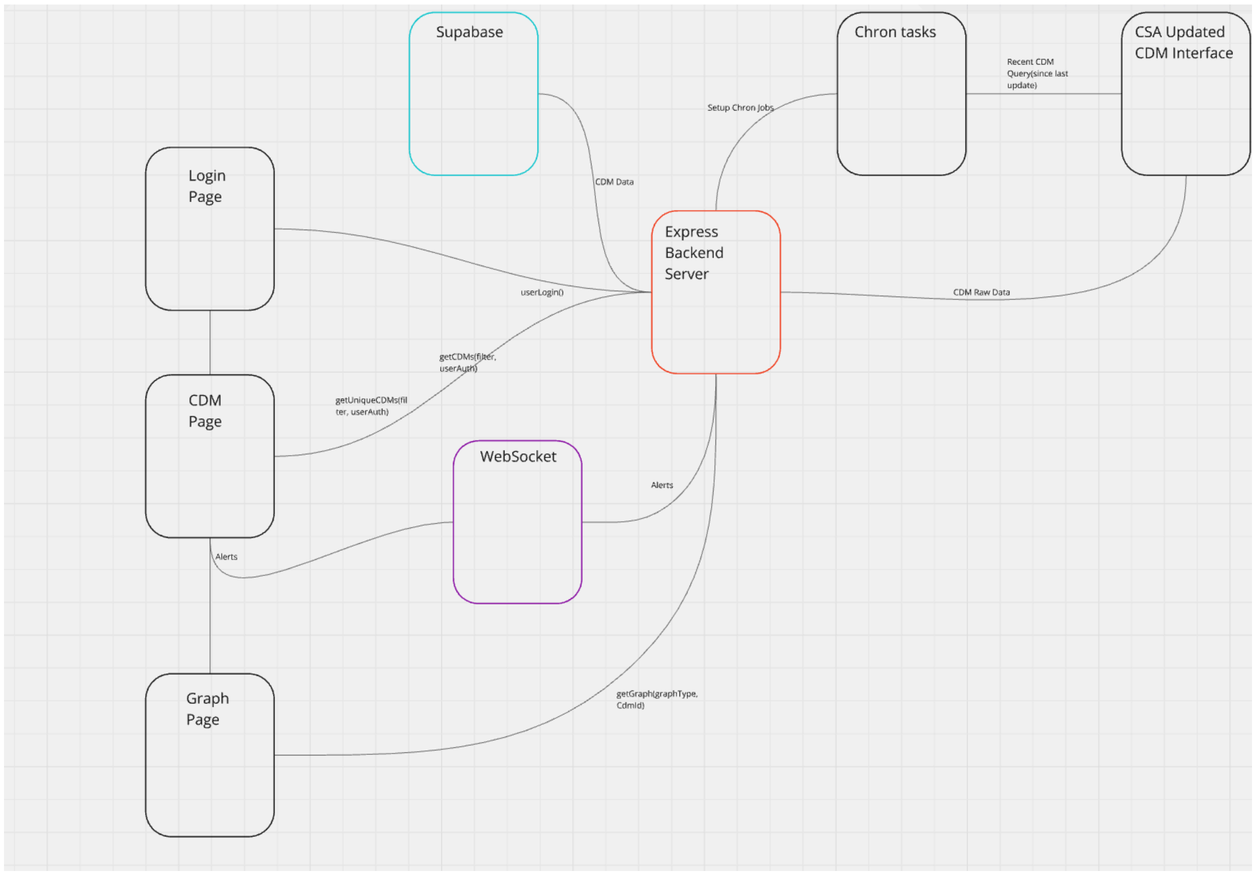
In the initial phase, we concentrated on understanding the domain-specific concepts provided by CSA. Most of the team was unfamiliar with satellite conjunctions, CDMs (Conjunction Data Messages), and the associated risk assessment practices. Guided by readings such as NASA's Collision Avoidance Best Practices Handbook and the CARA program documentation [1, 2, 3, 7], we identified CDMs as the key standardized data object and conducted a thorough review of their structure using CCSDS documentation [6]. We highlighted essential fields for our system, summarized in Table 1, which later informed our data processing and visualization components.

Figure 1: Use case diagram


Frontline operator
Alert technical operation
View type of space object
View probability of collision
View visualization graphs
Receive CDM notification
Technical operator

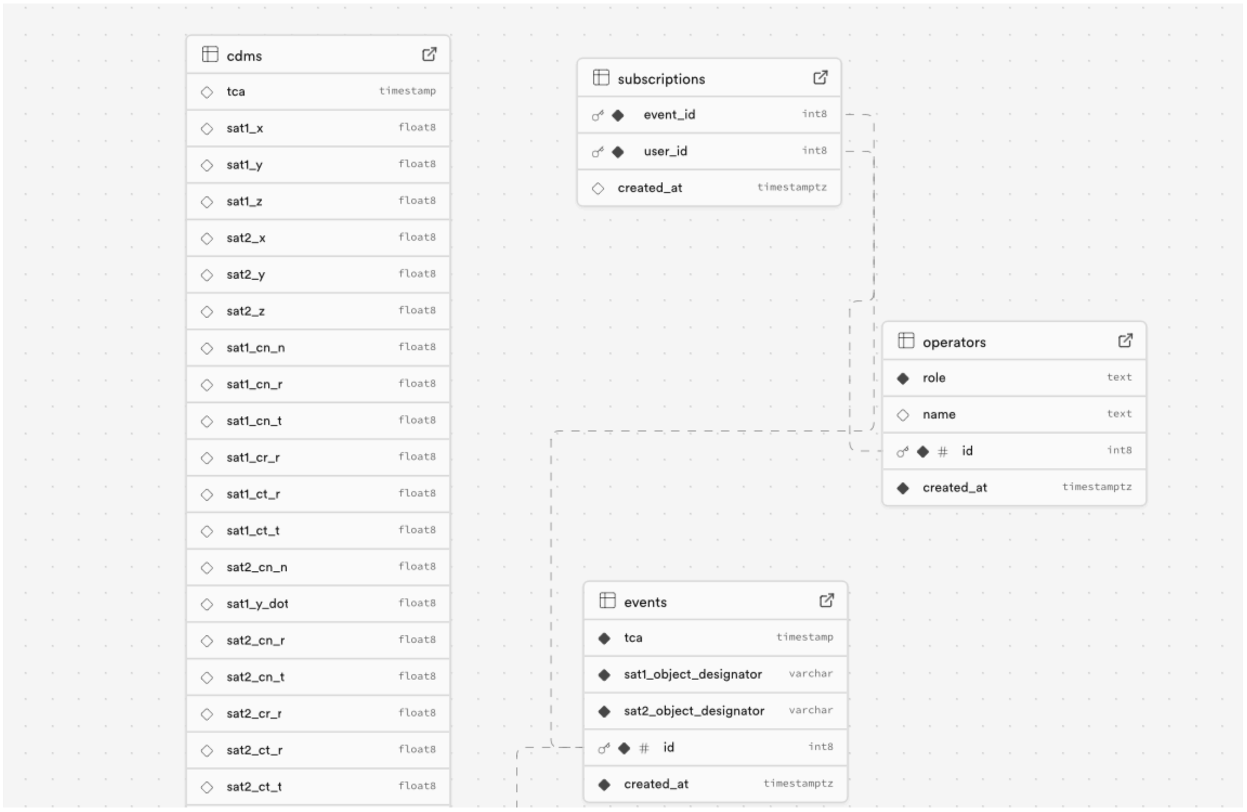
We then created a set of design artifacts to guide implementation. The use case diagram (Figure 1) defined key user roles—frontline and technical operators—and their required interactions. The data flow and architecture diagrams (Figure 2) modeled how CDM data would move through the system, from ingestion to alert generation. Our class diagram (Figure 3) defined core entities like User, CDM, and Event, while Figma mockups (Figures 4 and 5) helped visualize the user interface and inform frontend development.

Figure 2: Architecture Diagram


Supabase
Chron tasks
Recent CDM Query(since last update)
CSA Updated CDM Interface
Setup Chron Jobs
Login Page
CDM Data
Express Backend Server
userLogin()
CDM Raw Data
getCDMs(filter, userAuth)
CDM Page
getUniqueCDMs(filter, userAuth)
WebSocket
Alerts
Alerts
Graph Page
getGraph(graphType, CdmId)

Development was split between frontend and backend teams. The backend, built with Express and Supabase, handled user authentication, data storage, and API services. The frontend, built with React and TypeScript, focused on role-specific dashboards, secure login, and real-time data visualization using CesiumJS. GitHub was used to manage source control and sprint planning, and weekly meetings with CSA allowed us to continuously validate our progress against evolving requirements. For instance, after CSA informed us that live CDM feeds would not be available, we adapted by generating mock data sets—this change ultimately allowed us to test the system more thoroughly by simulating a wide range of scenarios.

Figure 3: Database class diagram



cdms	
tca	timestamp
sat1_x	float8
sat1_y	float8
sat1_z	float8
sat2_x	float8
sat2_y	float8
sat2_z	float8
sat1_cn_n	float8
sat1_cn_r	float8
sat1_cn_t	float8
sat1_cr_r	float8
sat1_ct_r	float8
sat1_ct_t	float8
sat2_cn_n	float8
sat1_y_dot	float8
sat2_cn_r	float8
sat2_cn_t	float8
sat2_cr_r	float8
sat2_ct_r	float8
sat2_ct_t	float8

subscriptions	
event_id	int8
user_id	int8
created_at	timestamptz

operators	
role	text
name	text
# id	int8
created_at	timestamptz

events	
tca	timestamp
sat1_object_designator	varchar
sat2_object_designator	varchar
# id	int8
created_at	timestamptz

Figure 4: Figma design of Technical Dashboard

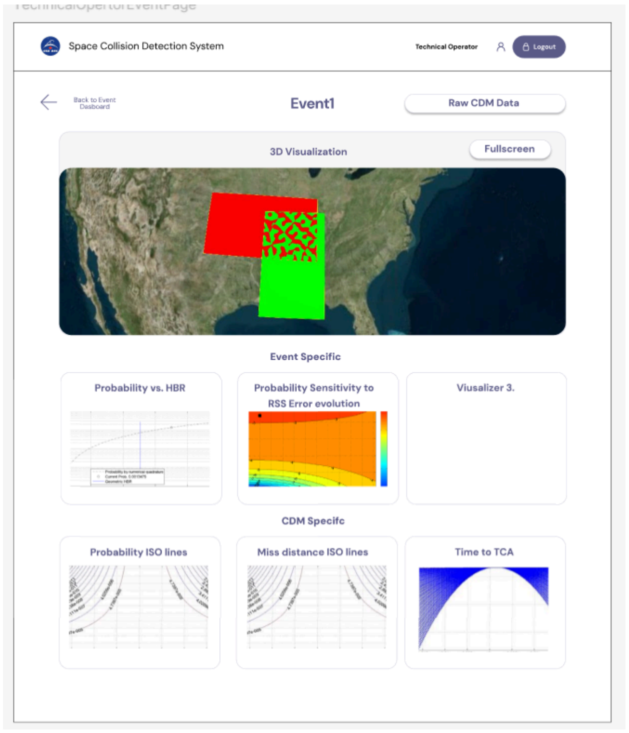
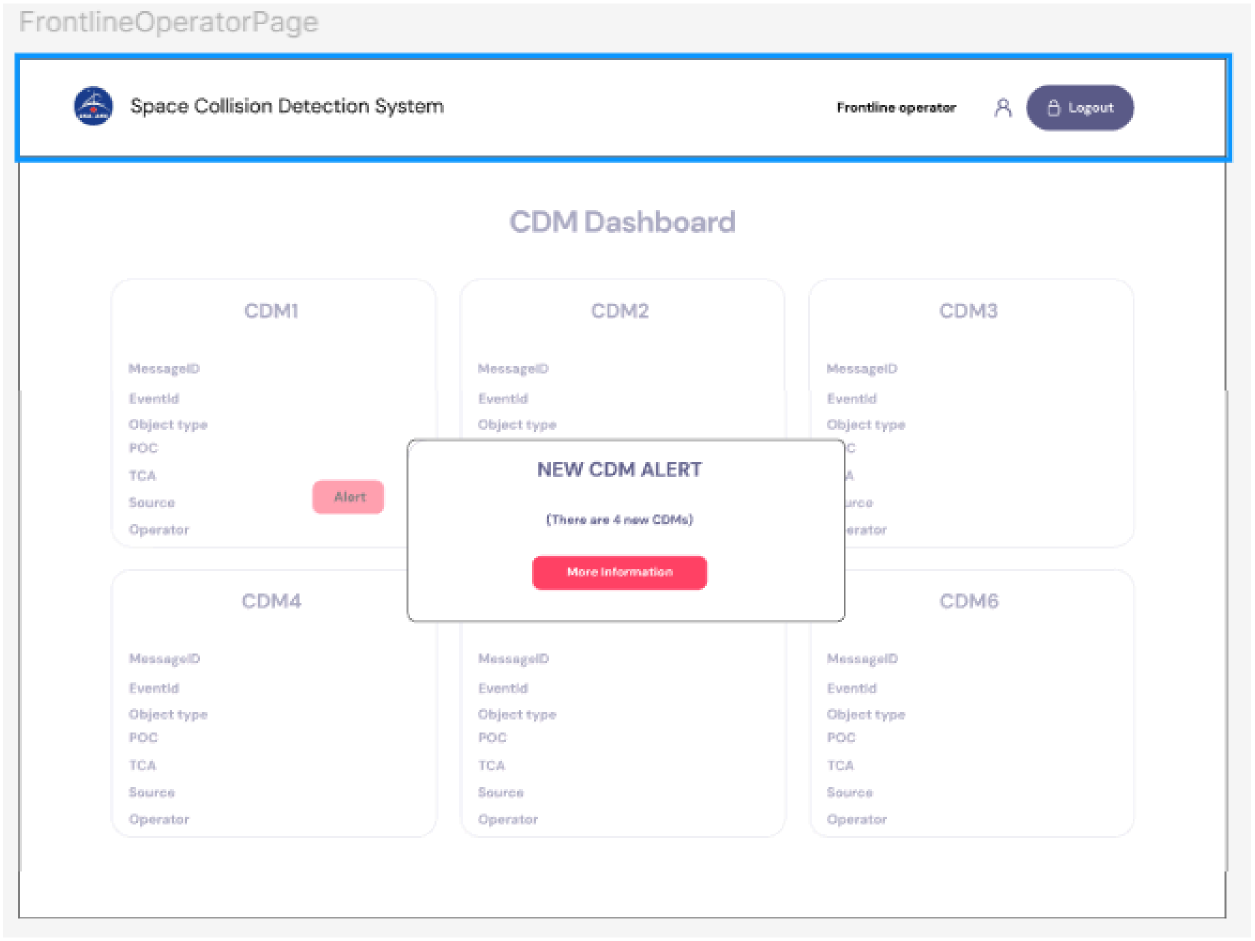


Figure 5: Figma design of Frontline operator



In the second phase, we expanded the system's functionality significantly. We introduced dynamic filtering that updates in real time as new CDMs are ingested, and implemented a customizable alerting system, where users can define thresholds for collision risk. A new communication mechanism between operator roles allows frontline users to notify technical operators directly within the system. We also added pagination to improve performance when handling large data sets, and extended the visualization layer with additional graphs that show the evolution of conjunction risk over time.

To ensure reliability, we introduced rigorous testing practices, including unit and integration testing, as well as simulated end-to-end interaction testing between frontend and backend components. These efforts helped validate core features like filtering, alert generation, and secure role-based access, and laid the groundwork for further development and potential deployment.

Frontend

Frontend Architecture and State Management

The application's frontend is built using React with TypeScript, selected for its lightweight nature, scalability, and strong typing capabilities. React's component-based architecture ensures modularity, enabling reusable UI elements and a clear separation of concerns. This approach simplifies development and ensures the application is maintainable and easy to scale as it evolves.

State management is handled using Redux, with Redux Toolkit simplifying its configuration. Redux is used to manage critical application state, particularly login functionality and continuous authentication. Supabase is integrated as the backend authentication provider, with access tokens securely stored in the Redux store upon login. This ensures session continuity and reliable authentication. Additionally, middleware is implemented in Redux to enforce role-based access control (RBAC). This restricts technical operators to specific pages, such as TechDashboard, EventDetail, and RawDetail, while frontline operators can only access the FrontDashboard. These measures provide a secure and tailored user experience.

To enhance maintainability and ensure a structured approach, the project follows a modular folder structure. Each page or component resides in its folder, which includes a .tsx file for logic and rendering, an index.ts file for streamlined imports, and a .module.css file for scoped styling. CSS Modules were chosen to prevent style conflicts and maintain consistency throughout the application. This modular structure supports the separation of concerns, allowing each feature to be developed and updated independently.

The application includes several key pages. The Login page provides user authentication functionality, ensuring secure access to the system. The TechDashboard presents technical operators with a list of events, while the FrontDashboard offers a tailored view of recent events for frontline operators, with a planned alerting system to enhance their workflow. EventDetail displays CDM-specific graphs for detailed event analysis, GraphDetail offers detailed graph views by type and event, and RawDetail presents raw CDM data in a tab-based layout for easier navigation. These pages, combined with the RBAC system, ensure the application meets the needs of different user roles while maintaining security and usability.

Routing and Secure Navigation

The application's routing system, implemented with react-router-dom, ensures secure and seamless navigation between pages. Public routes, such as the Login page, are accessible to unauthenticated users, while private routes dynamically route authenticated users to role-specific dashboards based on their permissions. This routing logic enforces strict access control and provides users with a responsive and intuitive experience.

Key routes include /login for unauthenticated users and role-specific routes such as /front-dashboard for frontline operators and /tech-dashboard for technical operators. Dynamic routes like /event/:eventId and /graph/:type/:eventId load event-specific and graph-specific data, respectively, based on URL parameters. To handle unauthorized or undefined paths, a catch-all route redirects users to the appropriate dashboard or Login page, ensuring smooth navigation and preventing access to restricted areas.

Real-Time Data Integration with Hasura Subscriptions

A key addition to our application for the final project is real-time data feeding using Hasura subscriptions. This implementation provides continuous data updates without requiring manual refreshes. Technical operators now see new conjunction events appear on their dashboard instantly as they are detected. The subscription model eliminates the need for polling, which reduces server load and ensures minimal delay between data generation and visualization within the application.

The frontline operators benefit from this real-time capability through an enhanced alerting system on their dashboard. When critical events that meet predefined thresholds are detected, alerts are immediately pushed to the frontline dashboard, enabling faster response times to potential collision risks. This instant notification system significantly improves the operational efficiency by reducing the time between detection and action.

The Hasura subscription implementation is tightly integrated with Redux, with subscription data flowing through dedicated reducers that update the application state. This integration ensures that all components using the data remain synchronized and consistent throughout the application, regardless of which view is active.

Enhanced Filtering System

For the final project, we have implemented a comprehensive filtering system that allows operators to narrow down the events and CDMs displayed based on specific criteria. Operators can filter by Satellite ID, entering specific satellite designations to focus on particular assets of interest. The system includes the ability to manage satellite subscriptions through a personalized list of satellites they're monitoring, with the ability to add or remove items from their watchlist.

Technical parameters like Probability of Collision (POC) and Time of Closest Approach (TCA) now have custom threshold input fields, allowing operators to define their alert sensitivity. These filters work seamlessly with our visualization tools, with dashboard cards showing events dynamically updating based on filter selections. The filtered data feeds directly into the D3.js graphs and Cesium 3D visualizer, ensuring consistent data representation across all views.

The "My Preferences" section allows each user to save their filter settings, creating a personalized experience that persists across sessions. This customization further enhances the role-based access control by not only restricting access based on role but also tailoring the interface to individual user preferences within those roles.

Graphs

As part of the visualization tool, we designed and implemented three key graphs using D3.js: Probability of Collision (Pc) Over Time, Miss Distance Over Time, and Root Sum of Squares (RSS) Over Time. These graphs were carefully chosen for their ability to convey essential information while being relatively straightforward to implement. Together, they provide operators with a clear and comprehensive view of the critical parameters needed to assess and respond to potential conjunction events.

D3.js was selected as the visualization library for its flexibility and power in creating dynamic, interactive, and visually appealing data representations. Its support for custom visualizations allowed us to tailor each graph to meet the specific requirements of CSA operators. Additionally, D3.js integrates seamlessly into web-based platforms, making it an ideal choice for our tool, which prioritizes accessibility and usability.

The three graphs were chosen because they effectively balance complexity with informativeness. The Probability of Collision (Pc) Over Time graph illustrates the evolving risk level as the time to closest approach (TCA) approaches, helping operators identify critical conjunctions. The Miss Distance Over Time graph provides a straightforward visual representation of the spatial separation between objects, offering immediate insights into proximity trends. Lastly, the RSS Over Time graph quantifies the combined positional uncertainty, allowing operators to gauge the confidence level of the predictions. These graphs collectively enable operators to make quick, informed decisions while also serving as a foundational layer for more detailed analysis if needed.

In the final phase of the project, we introduced two advanced visualizations—the Isoline and Heatmap Tradespace graphs—which model how varying maneuver strategies (ΔV and timing) affect the resulting miss distance. These advanced visualizations have been fully implemented with toggle functionality allowing users to switch between visualization types.

The Heatmap View presents a color-coded grid where the x-axis represents the ΔV (change in velocity) values in m/s and the y-axis represents the time before TCA (in hours) when a maneuver might be executed. Colors range from red (indicating smaller miss distances and higher risk) to green (indicating larger, safer miss distances). This comprehensive visualization allows operators to quickly identify optimal maneuver regions without having to manually calculate numerous scenarios.

The complementary Isoline View displays the same data using contour lines, where each line represents a specific miss distance value (marked in km). This representation helps operators identify thresholds and boundaries between different safety levels. For example, they can easily trace the 1.0 km safety contour to determine the minimum ΔV required at different execution times. Both visualizations maintain consistent data, showing the initial miss distance (e.g., 0.47 km) as a reference point.

These advanced visualization tools represent a major step forward in actionable risk assessment and decision support. By providing technical operators with an intuitive interface to explore thousands of maneuver possibilities simultaneously, they can identify the most fuel-efficient and effective collision avoidance strategies with greater confidence and speed.

Cesium 3D Visualizer

A critical feature of the technical operator's dashboard is the 3D Cesium animation, which serves as the centerpiece of the visualization tool. Cesium is a powerful platform for rendering objects on a three-dimensional representation of the Earth, making it especially suited for visualizing objects in space or the atmosphere. The Canadian Space Agency (CSA) has previously utilized Cesium within its operations, so adopting this tool aligns with their existing expertise. This approach not only facilitates the seamless integration of our project into their workflows but also ensures that users can easily navigate and adopt our visualization tool.

To implement this feature, we utilize the frontend library CesiumJS, along with Resium, a React wrapper that simplifies the integration of Cesium into modern web applications like ours. In the initial prototype, the visualization tool includes satellite objects projected onto their real-time positions on Earth's surface. A timeline feature allows users to navigate through time, while the animation highlights the satellite's trajectory, providing a clear view of its past movements.

For the final implementation, we have significantly enhanced the Cesium visualization with maneuvering options that allow technical operators to input and simulate potential collision avoidance maneuvers. The maneuvering interface enables users to select a specific satellite ID from a dropdown menu and input precise maneuver parameters, including the execution time via a datetime picker and velocity adjustments along the X, Y, and Z axes. These parameters allow operators to model both the magnitude and direction of potential burns, giving them complete control over the simulated maneuver.

The covariance matrix visualization has also been implemented, allowing operators to see the uncertainty in position predictions represented as ellipsoids around the satellite objects. This visual representation helps operators better understand the confidence level of the trajectory predictions and assess the risk margins more accurately. The projected trajectory feature complements this by showing the satellite's expected path both with and without the proposed maneuvers, enabling direct visual comparison of different scenarios.

The user interface for the Cesium visualizer has been enhanced with additional controls, including options to save maneuver plans for future reference, export the maneuver data for use in other systems, and toggle the visibility of predictions and various visualization layers. The "Hide Predictions" button allows operators to focus on the current known positions, while the "Maneuvering Option" control opens the maneuver input dialog.

Future iterations will further expand these capabilities by refining the prediction algorithms and adding more sophisticated visualization of uncertainty factors. These enhancements will provide technical operators with an even more comprehensive and dynamic understanding of satellite behavior, enabling better decision-making and situational awareness in potential conjunction scenarios.

Results Achieved

The frontend development has achieved several milestones. Key pages, including Login, TechDashboard, and FrontDashboard, are fully functional and integrated with the RBAC system. The routing system effectively enforces access control, directing users to the appropriate pages based on their roles. Additionally, the use of Redux and Supabase ensures reliable authentication and session management, creating a secure and user-friendly experience.

The implementation of Hasura subscriptions has successfully enabled real-time data updates across the application, with new events and CDMs appearing instantly on user dashboards. The filtering system has been thoroughly tested and provides users with the ability to effectively narrow down the information displayed based on their specific monitoring needs. The Cesium 3D visualization with maneuvering capabilities offers a powerful tool for technical operators to model and assess potential collision avoidance strategies.

The addition of advanced Heatmap and Isoline visualizations for maneuver tradespace analysis has significantly enhanced the decision-making capabilities of technical operators. These visualizations allow operators to quickly identify optimal maneuver parameters that maximize safety while minimizing fuel consumption, representing a substantial improvement in operational efficiency.

These achievements provide a strong foundation for further development and refinement as we continue to enhance the system's capabilities and user experience.

Design Decisions and Process

Several design decisions have guided the frontend development process. React and TypeScript were selected for their scalability, maintainability, and strong community support. Redux was chosen for its robust ecosystem and ability to manage complex state logic, particularly for authentication and

RBAC. CSS Modules were used for styling to prevent conflicts and maintain a consistent design across the application. The decision to implement RBAC via Redux middleware ensures a secure and personalized user experience.

The design process began with research and prototyping to evaluate various frontend libraries and state management tools. An iterative development approach was adopted, starting with a minimal viable product (MVP) and gradually adding features based on client feedback. The decision to incorporate Hasura subscriptions for real-time data and enhance the Cesium visualization with maneuvering options was made based on user feedback highlighting the importance of immediate situational awareness and scenario modeling capabilities.

The implementation of Heatmap and Isoline visualizations was driven by the need to provide operators with more comprehensive decision support tools that could analyze multiple scenarios simultaneously. These visualizations were designed with input from domain experts to ensure they represented the most useful parameters (ΔV and timing) and metrics (resulting miss distance) for collision avoidance planning.

Assessment of Tools and Integration

The tools and technologies used for the frontend have proven suitable for the application's needs. React and TypeScript provide a solid foundation for building scalable and maintainable applications. Redux effectively manages global state, particularly for login and authentication workflows. CSS Modules enhance styling by preventing conflicts and ensuring a consistent visual design.

Integration between tools has been seamless. Supabase integrates well with Redux for authentication and session persistence. Hasura subscriptions work effectively with the Redux store to maintain real-time data synchronization. The use of React's component-based architecture and modular folder structure has enabled clean and organized code. Middleware in Redux was custom-developed to enforce RBAC, demonstrating the flexibility of the chosen tools.

Cesium and D3.js have proven to be powerful visualization libraries that integrate well with the React ecosystem. D3.js in particular has demonstrated its versatility in creating both the standard time-series graphs and the more complex Heatmap and Isoline visualizations. These visualization components provide both technical and frontline operators with the visual tools they need to understand complex conjunction events and make informed decisions.

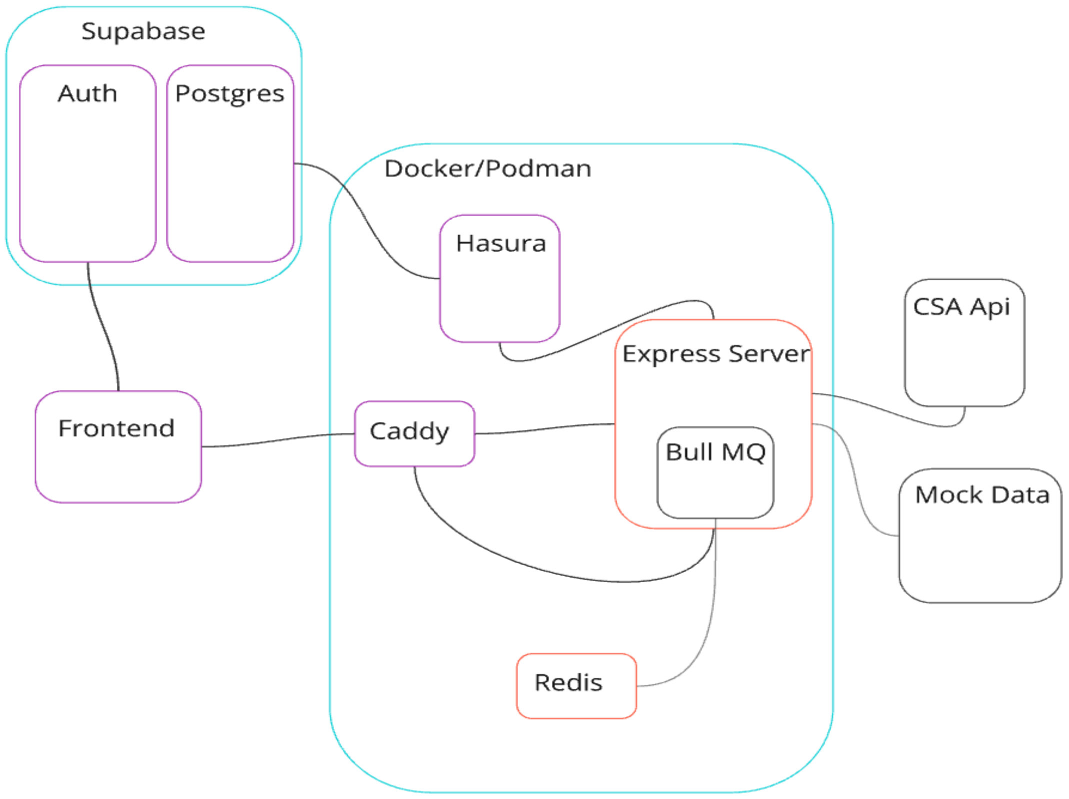
No significant modifications to the tools were required, indicating their suitability for the project's requirements. This successful integration validates our initial technology choices and provides a solid foundation for future enhancements and feature additions.

Backend

Express Api with Typescript

For our backend server, which can be seen in figure 6, we decided to use Express Api with typescript over Javascript. The main reason for using typescript was that we have defined types, mainly CDMS which follow a particular structure. Our database is also an SQL database, which means type adherence is important to avoid issues with our queries. It also makes working with Graph QL substantially easier as query parameters are defined as types.

Figure 6: Updated Backend Diagram



Our choice for express comes primarily from our familiarity with it. Express is also an industry standard as it comes with a very simplistic base without much overhead. It also had a modular structure for responses, which works well with standard authentication flows(especially the supabase one we used). Express also implements well in docker, which integrates well into our self-hosted solution.

Supabase

Supabase has many interesting features and was the optimal choice as a database based on our requirements. One of our requirements was simplicity for users; supabase offers a very intuitive hosted user interface, which means that administrators can easily work with data without needing to install software on their computer(as often needed with standard SQL databases). Another requirement was the simplicity to move between databases. Since Supabase is essentially a postgres wrapper, which means that any implementor can replace Supabase with a postgres, or technically any SQL database of their choice.

The reason we chose to use an SQL based implementation over a non SQL also mainly stems from the fact that they are very easy to interchange compared to non SQL implementations like Firebase and MongoDB. Another advantage of the SQL structure is it allows us to easily define relationships and forces us to maintain an appropriate structure in our queries. It also allows us to use GraphQl, which makes development substantially more efficient(explained below in the Hasura section). Supabase specifically uses Postgres, which allows us to put a graph QL engine above it(unlike Pocketbase, an alternative we looked into).

Supabase also provides an authentication system, which allows us to manage our RLS by extracting the user's role in the authentication token.

Finally, supabase has a self hosting option, which matches the requirement of our system being self hosted.

Hasura

By utilizing a Graph QL engine such as Hasura, we're able to separate the frontend and backend development almost fully. Hasura provides an automatic Graph QL engine above SQL databases(which works specifically well on Postgres). By using Hasura, our backend only needs to provide one endpoint to the frontend. The backend acts as a middleware between Supabase Authentication and Hasura. It converts the role extracted by Supabase, and passes it into Hasura, where we have predefined RLS (Row level security). This allows only authenticated requests to pass, which are checked on the backend additionally before fetching data from Supabase.

Initially, we were planning on using a custom web socket(such as socketIo), but then decided to use Hasura's built in Graph QL subscriptions. These have the advantage that they allow the user to subscribe with a graph QL query, which like in SQL allows the user to specify any parameters they would like. This allowed us to build a complex filtering system, where the user can specify specific parameters they want, without needing to design a system that handles any form of filter.

Caddy

To make sure our API is secure, we used Caddy, which has automatic HTTPS handling. Caddy allows simple configuration with Cloudflare (which we did not implement due to costs, but recommend the CSA use to proxy the IP of the server).

Containerization (Docker)

One of the main requirements of the system was self hosting. To allow for easy self hosting on any Server, we decided to use Docker, which allows us to build our services as containers. It also allows us to export ports on the same network, which allows communication between our services; this is especially important for the connection between the express server, Redis, supabase, and Hasura. Using docker also allows ease of development, as it allows all of us to have the same environment (independent of local package versions such as node, npm, etc).

Data Design

As shown in Figure 3, the CDM table serves to store all information related to the data of a cdm(conjunction data message). It Contains a field for all data called raw_data (used for storage, in case of missing fields or a change in the names). Also contains an individual field for all data contained in the CDMs.

The events table serves as a central repository for tracking significant occurrences involving two objects, likely satellites. It records the Time of Closest Approach (TCA) between the two objects, along with their identifiers, referred to as sat1_object_designator and sat2_object_designator. Each event is uniquely identified by an id field, ensuring that all records are distinct and traceable. Additionally, the table includes a created_at timestamp to indicate when the event record was first created, providing a chronological context for the data.

The operators table contains information about the individuals or systems responsible for managing or monitoring satellite events. Each operator has a defined role, which could signify their level of access or responsibility, such as an administrator, a user, or a monitoring agent. The name field stores the operator's name, ensuring easy identification. Every operator is uniquely identified by an id field, and the table also includes a created_at timestamp to record when the operator's information was added to the system.

The subscriptions table facilitates the management of user subscriptions to specific satellite events. It associates each subscription with an event_id, linking it to a particular record in the events table, and a user_id, which identifies the subscribing user. This table also maintains a created_at timestamp to log the creation time of each subscription, enabling the system to track when users opted into receiving updates about specific events.

Application Logic

The backend has a singular endpoint, /api/graphQl, which allows for the frontend to send authenticated graphQl queries. It accepts an authentication token in the header, which it processes using supabase's authentication, and the our secret supabase token which it uses to decrypt the token.

On load of the database, the server loads into the database a series of mock CDMS, from which it auto creates Events with the following logic:

Any CDM with a time between 10 minutes of another, having the same SAT1 object designator and SAT2 object designator fall into the same event. Using this logic, we are able to populate our events table with the appropriate events.

Bull MQ and Redis

In order to constantly feed new data into the system, we implemented a queue based chron infrastructure using BullMq and BullArena, which allow constant retrieval of new CDMS. Since we were unable to gain access to the live CDM feed that the CSA uses, we mocked this system by fetching from a list of mock CDMS. This allowed us to load in new CDMS and events every X minutes (where we set X to be 10).

The advantage of BullMQ with redis, is that in the case that the server goes down, jobs are saved in the queue and can be continued once the server is back up. It allows the administrator to manually trigger a "job" (which is just a fetch of new CDMS) whenever needed.

Testing Plan

In order to validate our progress, we set up a weekly meeting with the CSA, in order to validate our progress against their requirements. To avoid going off track, we would compare the initial requirements with what we developed to make sure that any new changes of discoveries were accounted for. It also allowed us to adapt to any changes made by the client (CSA). For example, they realized fairly early on that they would be unable to provide us with direct access to the CDM feed, and would instead give us mock data to test our system.

This adaptation to mock data proved beneficial for our development process, as it allowed us to test our backend directly by simulating various CDM scenarios. We created a comprehensive set of mock

CDMs that represented different types of conjunction events, allowing us to validate our data processing pipeline and visualization components without depending on live data. The mock data approach also enabled us to develop and test our system's error handling capabilities and edge cases that might be difficult to encounter with real data.

As the project matured and new features were integrated, it became increasingly critical to ensure that both the frontend and backend components functioned correctly under a variety of scenarios. To support this, we began implementing systematic unit testing and integration testing strategies across the entire codebase. These testing efforts were aimed not only at verifying individual functions and modules but also at ensuring that complex interactions—such as alert triggering, data filtering, and cross-role communication—behaved as expected under realistic conditions. In the following sections, we will describe how testing was approached in both the frontend and backend, the tools and frameworks we used, and the impact these practices had on the overall stability and maintainability of the system.

Frontend Testing implementation

For the frontend components of our application, we implemented a multi-layered testing strategy that prioritized both quality assurance (QA) testing and targeted unit testing for critical components. The primary focus was on ensuring a seamless user experience while maintaining the integrity of the data visualization and interaction features.

The majority of our testing efforts were concentrated on manual QA testing, which allowed us to verify the complex interactions between components and validate the user workflows from an end-user perspective. This approach was particularly valuable for testing the Cesium 3D visualizer and the D3.js-based graphs, as these components involve intricate visual representations that are challenging to validate through automated tests alone. Manual QA testing sessions were conducted after each significant feature implementation, with particular attention paid to the maneuvering input interface, filter functionality, and real-time data updates via Hasura subscriptions.

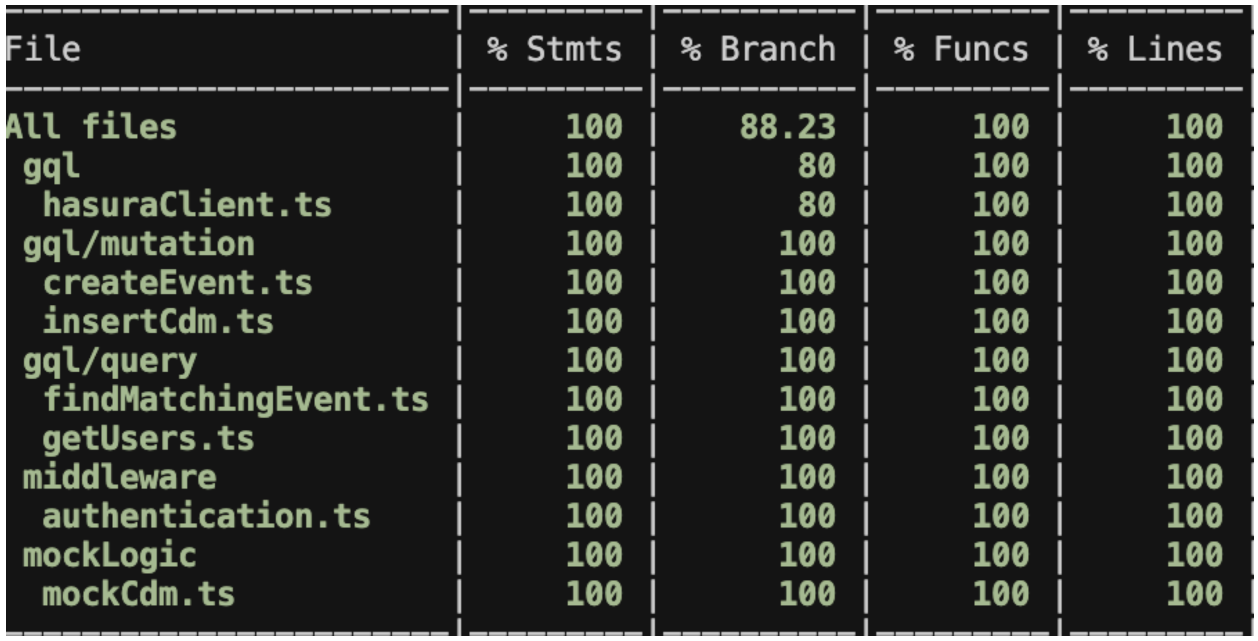
In addition to manual testing, we developed localized unit tests for specific critical components using Jest and React Testing Library. These targeted tests focused on core functionality such as the authentication flow, RBAC enforcement middleware, and the filter application logic. For the visualization components, we implemented unit tests that verified the correct data transformations and calculations before rendering, ensuring that the visualizations accurately represented the underlying data. The Heatmap and Isoline components, for example, included tests to verify that the color mapping and contour generation correctly interpreted the maneuver tradespace data.

The combination of these testing approaches allowed us to maintain high quality throughout the development process while focusing our automated testing efforts where they provided the most value. The final project submission includes the completed test suite that ensures the reliability of critical components such as data processing, user authentication, and visualization systems. The comprehensive testing approach has played a crucial role in delivering a robust and dependable application that meets the Canadian Space Agency's requirements for conjunction event analysis and spacecraft safety management.

Backend Testing implementation

For the backend testing implementation, we developed a comprehensive suite of Jest tests covering all critical components of our system. We focused on verifying the CDM data processing pipeline, which includes reading CDM files, transforming the data, and correctly storing it in our Supabase database through Hasura's GraphQL API. Our tests validate that the worker correctly processes batches of CDMs, creates appropriate event associations, and handles all the complex satellite data fields properly. Through rigorous test development, we've achieved 100% line coverage across our backend codebase, 100% Function Coverage and 88.23% branch coverage (Figure 7).

Figure 7: Test Coverage Backend



```
-----------------------|---------|----------|---------|---------
File                   | % Stmts | % Branch | % Funcs | % Lines
-----------------------|---------|----------|---------|---------
All files              |     100 |    88.23 |     100 |     100
 gql                   |     100 |       80 |     100 |     100
  hasuraClient.ts      |     100 |       80 |     100 |     100
 gql/mutation          |     100 |      100 |     100 |     100
  createEvent.ts       |     100 |      100 |     100 |     100
  insertCdm.ts         |     100 |      100 |     100 |     100
 gql/query             |     100 |      100 |     100 |     100
  findMatchingEvent.ts |     100 |      100 |     100 |     100
  getUsers.ts          |     100 |      100 |     100 |     100
 middleware            |     100 |      100 |     100 |     100
  authentication.ts    |     100 |      100 |     100 |     100
 mockLogic             |     100 |      100 |     100 |     100
  mockCdm.ts           |     100 |      100 |     100 |     100
-----------------------|---------|----------|---------|---------
```

We implemented robust tests for the GraphQL client to ensure proper communication with Hasura, confirming that queries and mutations function correctly while handling authentication and error cases appropriately. These tests verify that our system can retrieve event data, create new events, insert CDM records, and handle various error conditions that might arise from the GraphQL server, including syntax errors and server-side failures.

Integration tests form another critical part of our testing strategy, validating the complete workflow from receiving CDM data to storing and retrieving it through our API endpoints. These tests ensure that our authentication middleware functions as expected, rejecting unauthorized requests while properly processing authenticated ones. They also verify that our API correctly handles complex CDM event workflows, including finding matching events, creating new ones when needed, and associating CDMs with the appropriate events. The test suite also includes extensive error handling scenarios, verifying that the system properly responds to invalid data, network failures, and authentication issues.

Impact on Society and Environment

The environmental and societal impact of our project is significant, particularly in addressing the Kessler syndrome—a scenario where the density of objects in low Earth orbit increases the likelihood of cascading collisions. By assisting operators at the Canadian Space Agency (CSA) in efficiently tracking satellites and predicting potential collisions, our project contributes to mitigating this issue. This has substantial positive implications for both the environment and society. Preventing collisions reduces the generation of space debris, which could otherwise make certain orbital regions unusable for future missions, impacting satellite-based technologies that society heavily relies upon, such as communication, navigation, and weather monitoring.

Environmental Analysis

Use of Non-Renewable Resources: Our project minimizes the use of non-renewable resources at every stage. The application is designed to be hosted on existing CSA servers, eliminating the need for additional hardware and the associated environmental costs of manufacturing, energy consumption, and eventual disposal. This approach aligns with sustainable practices by leveraging existing infrastructure rather than demanding new resource allocation.

Environmental Benefits: By preventing satellite collisions, the project directly reduces the risk of space debris proliferation. Compared to alternative or less effective tracking technologies, our solution represents an environmentally friendlier option. The long-term preservation of low Earth orbit as a viable and sustainable environment is critical for ongoing and future satellite operations, which support Earth observation efforts to monitor climate change and environmental degradation.

Societal Analysis

Safety and Risk: The development of our project poses no safety risks to team members, as it primarily involves software development and data analysis. Furthermore, the application's implementation is designed with robust safeguards to ensure the accuracy of collision predictions, minimizing risks to satellite operators and the broader infrastructure that relies on these systems.

Benefits to Society: The societal benefits of this project are multifaceted. By enabling the CSA to better manage satellite operations, the project ensures continued access to satellite-based technologies that enhance quality of life—from reliable internet access to accurate GPS navigation and disaster management systems. Economically, the prevention of satellite collisions avoids costly damages to spacecraft, reducing financial burdens on operators and fostering sustainable development in the space industry.

Critical Reflection

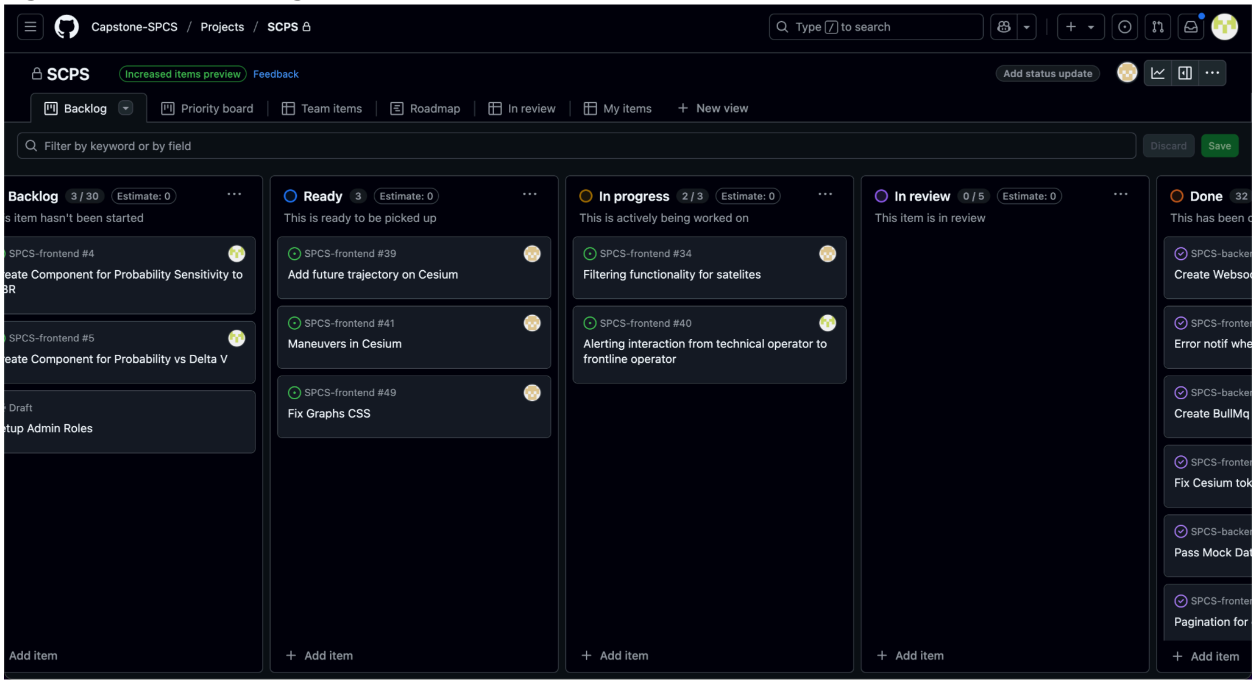
The new technologies introduced in this project also have sociological implications. Enhanced satellite tracking could encourage broader public and private investments in space activities, potentially increasing access to satellite-based services globally. However, this could also exacerbate inequalities if such technologies are not made accessible to developing regions. These factors highlight the need for thoughtful implementation and equitable dissemination of technological benefits.

In conclusion, our project's design solution offers clear environmental and societal benefits, with minimal reliance on non-renewable resources, significant environmental protection through debris mitigation, and notable improvements in societal infrastructure and safety. Its thoughtful design ensures sustainable and equitable impacts, contributing positively to both present and future generations.

Teamwork Report

Our team adopted an agile-inspired workflow, supported by GitHub's project management tools. We used GitHub Issues to break down client requirements into discrete, trackable tasks (figure 8). These were then distributed among team members in a way that ensured fair and balanced workloads, based on individual strengths and availability. We organized our development efforts using GitHub Issues and Pull Requests, which allowed us to maintain a clean, collaborative codebase. Each pull request was linked to a corresponding issue, enabling automatic resolution upon merge and ensuring traceability throughout the development process. Because our team had strong interpersonal familiarity and consistent communication, we were able to coordinate efficiently, adapt quickly to changes, and support one another throughout the project. This made it easy to stay aligned with project goals while maintaining flexibility, a key advantage in an agile development environment.

Figure 8: Github backlog



Capstone-SPCS / Projects / SCPS

SCPS Increased items preview Feedback

Backlog | Priority board | Team items | Roadmap | In review | My items | New view

Filter by keyword or by field

Backlog 3 / 30 Estimate: 0
s item hasn't been started
SPCS-frontend #4
eate Component for Probability Sensitivity to
3R
SPCS-frontend #5
eate Component for Probability vs Delta V
Draft
tup Admin Roles
Add item

Ready 3 Estimate: 0
This is ready to be picked up
SPCS-frontend #39
Add future trajectory on Cesium
SPCS-frontend #41
Maneuvers in Cesium
SPCS-frontend #49
Fix Graphs CSS
Add item

In progress 2 / 3 Estimate: 0
This is actively being worked on
SPCS-frontend #34
Filtering functionality for satelites
SPCS-frontend #40
Alerting interaction from technical operator to frontline operator
Add item

In review 0 / 5 Estimate: 0
This item is in review
Add item

Done 32
This has been
SPCS-backe
Create Webso
SPCS-fronte
Error notif whe
SPCS-backe
Create BullMq
SPCS-fronte
Fix Cesium tok
SPCS-backe
Pass Mock Dat
SPCS-fronte
Pagination for
Add item

Radu's collaboration includes most of the project management, the visualization tool with Cesium, testing infrastructure, maneuvering functionality, and other frontend tasks. Some difficulties were encountered in the implementation of the Cesium animation and the interactive maneuver planning features, but these challenges were overcome with support from peers and resources available online that clarified how to effectively use the Cesium library and related tools in context. Additionally, Radu contributed to setting up automated testing workflows using Jest to ensure frontend reliability and played a key role in integrating the canvas-based tradespace engine used for simulating satellite avoidance maneuvers.

Yassine made significant contributions to the project's engineering design phase, including the creation of comprehensive design diagrams, user workflows, and architectural documentation that formed the foundation of our development approach. Additionally, he played a crucial role in implementing the visualization components, particularly the D3.js graphs that provide critical conjunction event analysis capabilities for technical operators. As well as computing, evaluating and displaying the possible maneuvers through the isoline and heatmap tradespace graphs.

Daniel made significant contributions to the system's backend architecture and infrastructure, implementing the core services including Hasura, Docker containerization, and Supabase integration. He established the Express server logic and authentication flows, ensuring secure and efficient communication between all system components. Furthermore, his work on implementing row-level security and GraphQL integration through Hasura created a robust foundation for handling complex data queries while maintaining proper access controls. Daniel also created the backend testing plan and implemented it.

Shady led the frontend team during the initial development phase, taking charge of defining the core architecture and setting up the project. He established the React repository's structure, integrating TypeScript for scalability and strong typing. Additionally, he designed the Redux architecture to manage global state effectively, including user authentication and role-based access control. Shady implemented routing using react-router-dom, set up secure navigation between role-specific dashboards, and integrated API communication with custom hooks for data handling. This foundational work ensured the frontend was modular, maintainable, and aligned with the project's goals.

Conclusion

Over the course of the year, we successfully designed, implemented, and delivered the Space Collision Prevention System as a fully functional, self-hosted solution for the Canadian Space Agency. Building on the solid foundation laid in the first semester, we completed all planned features and went beyond initial expectations in several areas. The system now includes secure authentication, role-based access control, intuitive dashboards for both frontline and technical operators, and comprehensive visualizations of conjunction events—both in 2D using D3.js and in 3D using CesiumJS.

Our backend infrastructure, developed in Express and TypeScript, reliably processes and organizes Conjunction Data Messages, while the frontend, built in React, offers an operator-friendly interface. We integrated Supabase and Hasura to provide seamless real-time data management through a GraphQL API. The system is fully containerized with Docker, enabling consistent deployment across environments, and can be easily hosted on-premise using tools like Podman. We also added a flexible canvas-based tradespace visualization engine that allows technical operators to simulate and assess avoidance maneuvers.

In the second semester, we implemented automated testing using Jest, built an interactive heatmap and isoline engine for maneuver planning, and laid the groundwork for integration with real-time CDM feeds. The visualization tools now support both immediate triage and in-depth analysis, bridging operational and technical workflows. Our weekly consultations with CSA advisors helped us fine-tune the system to meet real-world constraints, including a continued reliance on mock CDM data.

This project has been an invaluable learning experience that pushed us to grow not just as developers, but as engineers. Over the past year, we gained first-hand insight into the challenges of building mission-critical software in a high-stakes, real-world domain like aerospace. From navigating complex client requirements to architecting a scalable, responsive, and secure system, we encountered—and overcame—obstacles that demanded both technical skill and team coordination. The opportunity to work closely with CSA advisors offered a unique window into the operational realities of satellite collision monitoring, grounding our work in real-world needs and constraints. Along the way, we developed a much deeper appreciation for clear communication, modular design, and rigorous testing practices. More than anything, this project taught us how to translate ambiguity into architecture, and ideas into impact. It's incredibly rewarding to see our work evolve into something that could one day contribute to space safety. As we look ahead to our careers, we carry forward the confidence, resilience, and collaboration this project helped us build—qualities we believe will define us as engineers.

References

[1] NASA, "STEP 1: Conjunction Event Prediction," *Conjunction Assessment Risk Analysis (CARA)*. [Online]. Available: https://www.nasa.gov/cara/step-1-conjunction-event-prediction/. [Accessed: Dec. 18, 2024].

[2] NASA, "STEP 2: Close Approach Risk Assessment," *Conjunction Assessment Risk Analysis (CARA).* [Online]. Available: https://www.nasa.gov/cara/step-2-close-approach-risk-assessment/. [Accessed: Dec. 18, 2024].

[3] NASA, "STEP 3: Close Approach Risk Mitigation," *Conjunction Assessment Risk Analysis (CARA)*. [Online]. Available: https://www.nasa.gov/cara/step-3-close-approach-risk-mitigation. [Accessed: Dec. 18, 2024].

[4] V. Abbasi, F. Babiker, M. Doyon, and D. Golla, "Close Encounters of an Advanced Kind: Lessons Learned and New Approaches in Collision Risk Assessment and Mitigation," Satellite Operations, Ground Infrastructure & Applications, Space Utilization, Canadian Space Agency.

[5] NASA, "Micrometeoroids and Orbital Debris (MMOD)," *White Sands Test Facility.* [Online]. Available: https://www.nasa.gov/centers-and-facilities/white-sands/micrometeoroids-and-orbital-debris-mmod/#:~:text=Kessler%20demonstrated%20that%20once%20the,orbit%20is%20no%20longer%20usable.. [Accessed: Dec. 18, 2024].

[6] Consultative Committee for Space Data Systems (CCSDS), "Conjunction Data Message," CCSDS 508.0-B-1, Blue Book, Aug. 2013. [Online]. Available: https://public.ccsds.org/Pubs/508x0b1.pdf. [Accessed: Dec. 18, 2024].

[7] NASA Office of Safety and Mission Assurance**,** "Spacecraft Conjunction Assessment and Collision Avoidance Best Practices Handbook," NASA OCE-51, Nov. 2021. [Online]. Available: https://nasa.gov/sma/oce_51.pdf. [Accessed: Dec. 18, 2024].

Appendices

Table 1: Important CDM field

Keyword	Description	Example of Values or Units	Obligatory
ORIGINATOR	Creating agency or owner/operator	JSPOC, NASA-JPL, SDC, CNES	Yes
MESSAGE_FOR	Spacecraft names(s) for which the CDM is provided	SPOT, ENVISAT, INTELSAT	No
MESSAGE_ID	ID that uniquely identifies a message from a given originator	ABC-12_34	Yes
TCA	The date and time in UTF of the closest approach.	n/a	Yes
RELATIVE_POSITION (RTN)	The R,T,N component of object 2's position in the Radial, Transverse, and Normal (RTN) coordinate frame.	m	No
RELATIVE_VELOCITY (RTN)	The R,T,N component of Object2's velocity relative to Object1's velocity in the RTN coordinate frame	m/s	No
COLISSION_PROBABILITY	The probability (denoted 'p' where 0.0<=p<=1.0), that Object1 and Object2 will collide. Data type = double.	n/a	No
OBJECT_NAME	Spacecraft name for the object.	SPOT, ENVISAT, IRIDIUM, INTELSAT	Yes
OBJET_TYPE	The object type.	PAYLOAD ROCKET BODY DEBRIS UNKNOWN OTHER	No
TIME_LASTOB_START	The start of a time interval (UTC) that contains the time of the last accepted observation. For an exact time, the time interval is of zero duration (i.e., same value as that of TIME_LASTOB_END).	n/a	No
TIME_LASTOB_END	The end of a time interval (UTC)	n/a	No

	that contains the time of the last accepted observation. For an exact time, the time interval is of zero duration (i.e., same value as that of TIME_LASTOB_START).		
AREA_PC	The actual area of the object. Data type = double	m**2	No
MASS	The mass of the object. Data type = double.	kg	No
THRUST ACCELERATION	The object's acceleration due to in-track thrust used to propagate the state vector and covariance to TCA.		
STATE VECTOR	Object Position Vector X, Y, X_DOT, Y_DOT & Z_DOT	km km/s for _DOT	Yes
COVARIANCE MATRIX	(Covariance Matrix 9×9 Lower Triangular Form. All parameters of the 6×6 position/velocity submatrix must be given. All data type=double.)	m**2	Yes