Elysium Documentation

Group Name: Climate Crusaders

Group Number: #2

Github Repository URL:

https://github.com/windyhillville/SE-Project

Team Members:

- Dominick Consiglio
- Benjamin Davidson
- Nathan McTear
- Daniel Moody

Course: CEN3031 - Introduction to Software Engineering

Instructor: Professor Lisha Zhou

Date: April 23, 2025

Table of Contents

Section 1: Project Overview and System Models

1.1 Project Description

The Elysium project was developed to raise awareness about Earth's unique properties and the challenges of planetary habitability by comparing our planet to others beyond our solar system. This project seeks to educate users on what makes our planet habitable, and why preserving its climate is critical. Built using real-world data from a trusted NASA exoplanet database, the system aims to foster environmental awareness by encouraging users to reflect on conditions that sustain life and the potential consequences of climate change. It is intended to be used as an easily accessible learning resource for students, educators, and anyone who is interested in planetary science and climate change.

1.2 How Solution Addressed Challenge Statement

Produce, refine and maintain software and user interface via GUI, that compares our planet's atmosphere to planets outside our solar system based on real world data pulled from a credible database like NASA. This will highlight the unique atmosphere that Earth provides and just how fragile our home is. "Elysium" will highlight the pinpoint differences that make Earth special compared to other exoplanets. We intend to provide the user with valuable information to educate decisions surrounding climate change. While the user is learning about other exoplanets we will provide tips to the user on how to prevent our planet from falling into the conditions that make Earth uninhabitable.

1.3 Features and Functionality

Elysium includes a range of features designed to educate users about planetary habitability and climate awareness through comparative analysis and climate scenarios. Below is an overview of the system's core functionality:

- **Random Exoplanet:**
    - Users can explore a random exoplanet from a curated selection of exoplanets beyond our solar system.
    - Each exoplanet entry include real-world data such as:
        - Mass
        - Radius
        - Gravity
        - Distance From Earth
        - Planetary Type
- **Comparison to Earth**
    - Users can randomly select an exoplanet to compare its data directly with Earth's.
    - Visual and textual comparisons help highlight the unique qualities that make Earth habitable.
- **Climate Scenarios**
    - The system includes three distinct future climate scenarios based on real-world climate models:
        - Green Future: Assumes strong global action on emissions and sustainability.

        - Medium Emissions: Represents moderate environmental policy and partial mitigation.
        - Extreme Emissions: Reflects continued high emissions and global inaction.
    - Each scenario is tied to Earth's climate data to show possible long-term outcomes.
    - The user will also be able to see some climate action tips to see what changes in their lives they can make to put their own contribution into lowering their carbon footprint.
- **Favorite Exoplanets**
    - Users can favorite planets they view in the Random Exoplanet Screen or Comparison to Earth by clicking on the Add Favorite button next to it.
    - In the Favorite Exoplanets menu, they can view the exoplanets they favorited.
    - They are only able to favorite 5 exoplanets.
- **Earth**
    - Users can view Earth without any UI elements to visually explore its characteristics in a minimalistic environment.
- **Solar System**
    - Although it is not directly tied to the solution of the project's challenge statement, it serves as a proof of concept of the capabilities of the Processing-based frontend. Through the simulation of planetary motion in a 3D space, it demonstrates the potential for extending the Elysium platform for more interactive and educational experiences in the future.
- **User Accounts**

    - Users can create and log into personal accounts.
    - Login allows for saving favorite exoplanets.
- **Clean and Educational Interface**
    - Designed for ease of use by students and educators.
    - Focus on navigation and visual appeal to support and an engaging learning environment.

1.4 System Models

For this project, we have chosen to follow a layered architectural approach. The system will be organized into three distinct layers: Presentation Layer, Application Layer, and Data Layer. The Presentation Layer will be composed of all User Interface interactions including but not limited to login, navigation, and visualizations. The Application Layer will handle the processing of planetary data and climate scenarios and implement communication between the UI and underlying data. The Data Layer will deal with storing planetary data points from a reputable archive like NASA, along with ensuring that the planetary data is accurate.

System Decomposition

Here is a breakdown of Elysium into distinct components, which correspond to each layer of our chosen software architectural pattern.

Presentation Layer (Front End – Java)

- *Login System* – Handles user authentication
- *Main Interface* – Provides a clean and intuitive design for users.

- *Interactive Data Visualizations* – Displays climate change scenarios and planetary comparisons.

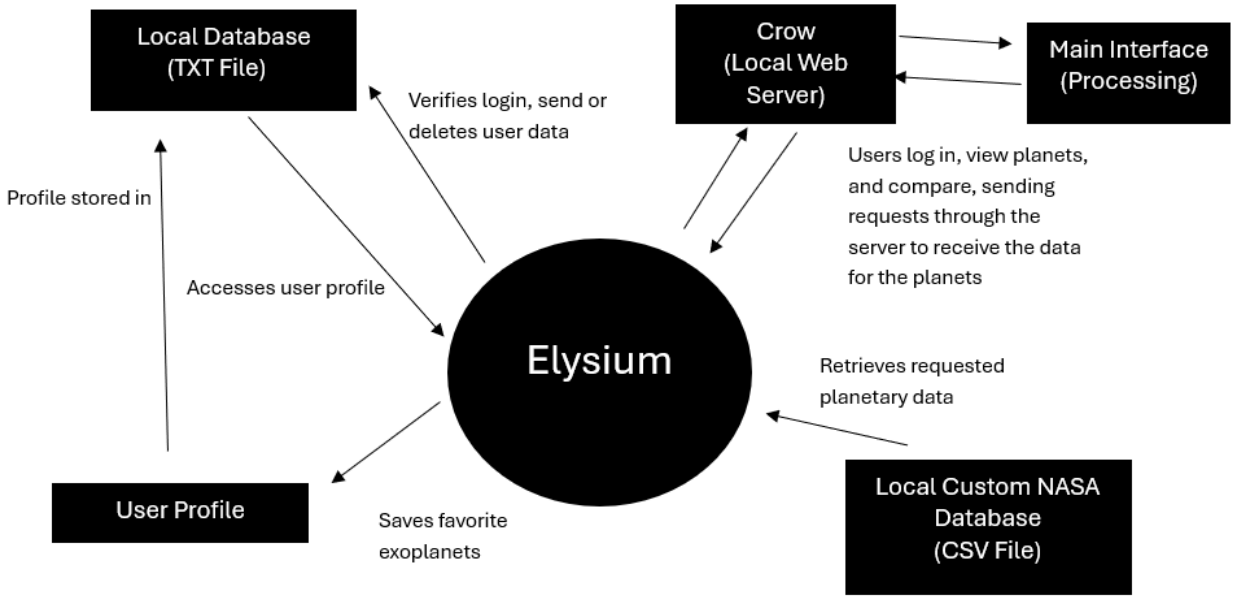
Application Layer (Backend – C++/Crow)

- *Data Processing* – Manages planetary datasets.
- *User data handling* - Manages user data in creation of accounts and favoriting planets

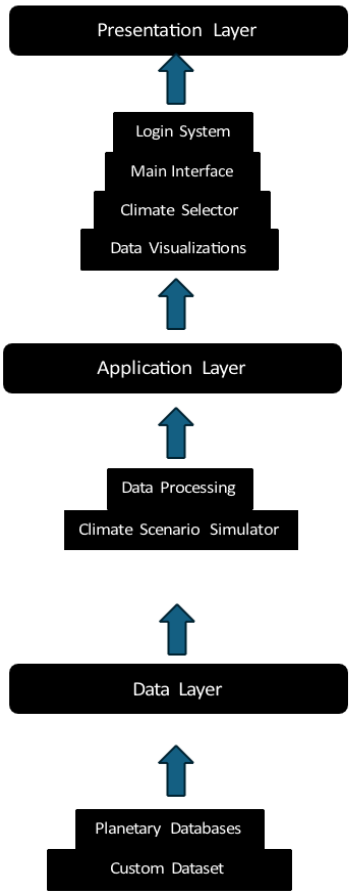
Data Layer (NASA)

- *Planetary Databases (NASA)* – Stores planetary atmospheric data.
- *Custom Dataset* – Contains predefined climate conditions for analysis.

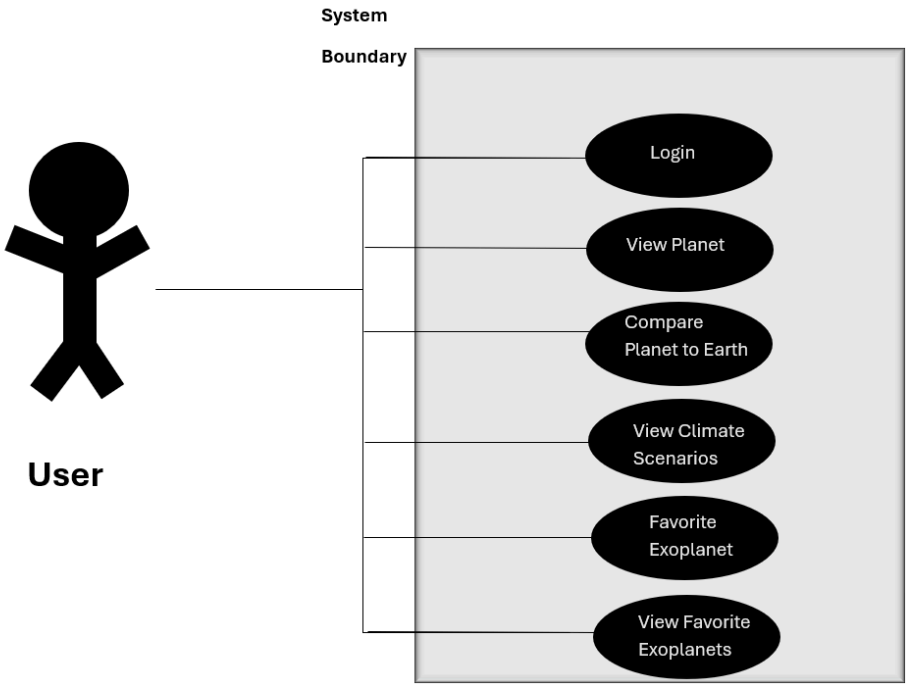
System Context Model:



Identifying the Design Structures, Layers, and Components:



Use Case Model:



Section 2: Code Management, Testing, and Static Analysis

2.1 Code Management

Our team used Git for version control and maintained our project repository on Github to ensure collaboration, change tracking, and backup throughout development.

The project was organized into clearly separated folders to maintain clarity:

frontend:

- SystemUI/ - Source code for the Processing application

backend:

- src/ - Source code for the core application
- external/ - dependencies required for application to work properly
- tests/ - Catch2 framework and unit tests
- users/ - Contains user-created data
- data/ - Contains custom NASA dataset

We worked directly on the main branch and ensured all commits maintained a buildable and working state. Before pushing any changes, each team member tested their contributions locally to confirm stability and avoid breaking main on the repository. Once the changes were confirmed to be stable, the team member would commit their contribution in a clear commit message to state what they changed.

2.2 Test Plan

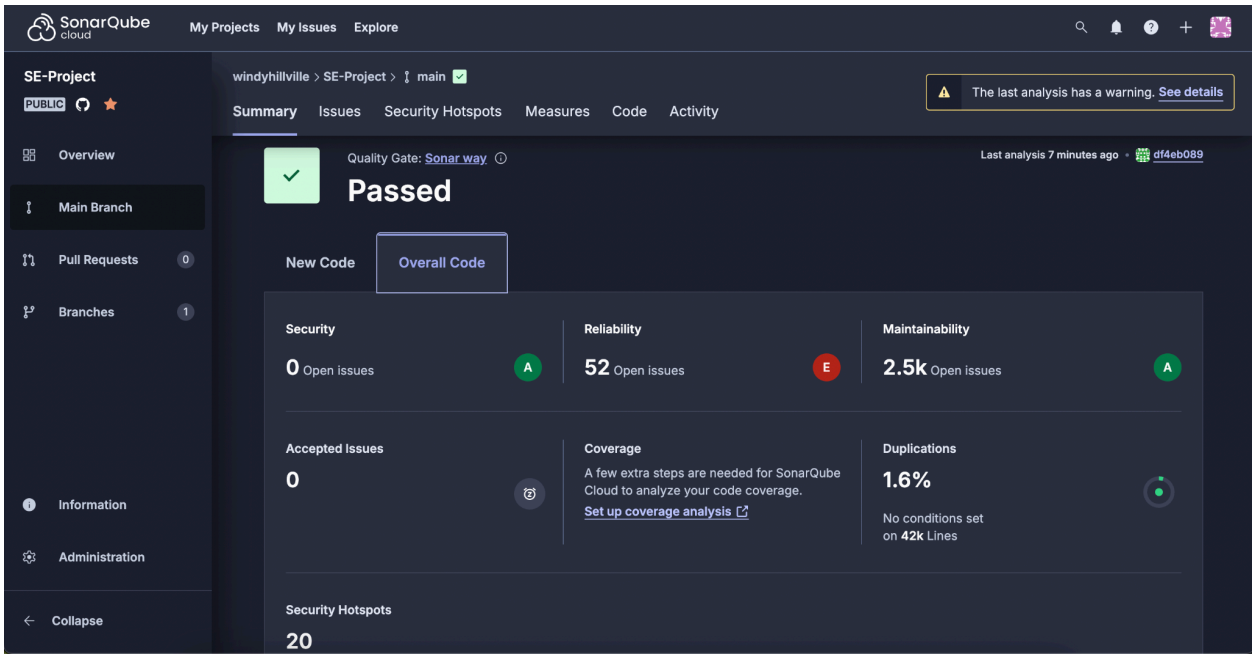
The goal of our testing strategy was to validate the correctness, reliability, and stability of the Elysium system. We focused primarily on unit testing and manual verification of outputs. Tests were created to ensure the core functionality of the system such as data parsing, planet comparison, and user-related features performed correctly.

The unit tests were written for key functions such as parsing planet data from CSV files, calculating planetary properties, searching planet properties, and managing user data. Manual testing was used to test the GUI to make sure everything was displaying correctly.

The types of test cases were both positive and negative. Positive being valid inputs correctly returning the expected information and negative being invalid inputs handled correctly.

The testing framework that we used was Catch2 to run automated tests in C++ backend. These tests were run frequently throughout the development process, especially after implementing a new feature.

2.3 Static Code Analysis Report



For our static code analysis, we used SonarQube cloud. According to our results, we learned about information surrounding our applications security, reliability, maintainability, duplication, size, complexity, and overall issues. For our security rating, we received an A which indicates that our program is built securely. In terms of our reliability score, we received an E with 36 bugs reported. While this score was not satisfactory, the majority of bugs are in catch.hpp with 16 bugs and our Crow framework with 20 bugs. These bugs were out of our control since we did not write them, but we recognize that there may be better options for designing our backend. For maintainability we scored an A, but with 2,471 code smells. The large majority of these code smells reside in catch.hpp and the Crow framework source code, but we did receive code smells in our DataProcessing.cpp file. This indicates opportunities for refactoring our code like reducing redundancy with our binary search function and improving modularity. Overall, our system passed the quality test, but we acknowledge areas in our design that could use improvements.

Section 3: Technical Setup and Configuration

3.1 Technical Details

Elyisum was developed as a desktop application combining a C++ backend integrated with Crow and a Java-based frontend interface using Processing for data visualization. The project integrates planetary data from CSV files containing real-world values sourced from a custom NASA dataset.

Technical Details:

- Programming Languages:
    - C++ (backend logic, data processing, file handling, Crow)
    - Java (graphical interface, event handling via Processing)
- Frameworks
    - Crow (setting up web server, handling HTTP requests)
- Libraries
    - HTTP Requests for Processing (sends out requests to our server, receives processed request from server)
    - PeasyCam (sets up our camera in Processing)
- Tools
    - CLion (for coding, compiling and debugging our backend)
    - Processing (coding our frontend, graphics libraries, data visualization)
    - Github (version control)

3.2 Installation Instructions

Elysium uses a singular framework for implementing and connecting our backend to our frontend, while using a broad range of libraries for graphics processing. Therefore, we have provided instructions for installing each tool, whether you're using Windows or MacOS.

Crow (C++ Framework) Dependencies

MacOS

1. After cloning our project directory, type the following prompt on the terminal.

   brew install asio

2. In your CMakeLists.txt, comment out the lines specified in the file.

Windows

1. Download Visual Studio Build Tools using the link below or have Visual Studio installed with the C++ environment tools.

   https://visualstudio.microsoft.com/visual-cpp-build-tools/

2. Clone a C++ package manager called vcpkg by typing in the following prompt:

   git clone https://github.com/microsoft/vcpkg.git

3. Go to the directory where you cloned the repo and then use Command Prompt to run these commands in the directory that you choose. (It is easier to put this in the /external folder)

   cd vcpkg

   .\bootstrap-vcpkg.bat

4. Install ASIO in that directory by using the following prompt:

.\vcpkg install asio

In Clion, go to → *Settings* → *Build, Execution, Deployment* → *Cmake*

In cmake options, put this in the field there and click apply.

-DCMAKE_TOOLCHAIN_FILE=external/vcpkg/scripts/buildsystems/vcpkg.cmake

It will take a few moments for Clion to apply these changes.

If it does not find the file, you can use an absolute path to that directory.

Processing

MacOS

1. Install Processing using the link below:

   Getting Started / Processing.org

2. Per the website, "Processing comes as a .dmg disk image. Open the file from your Downloads folder, then drag the Processing icon to your Applications folder."
3. For the libraries, in Processing's menu bar:

   → *Sketch* → *Import Library* → *Manage Libraries*, and then search for and install Peasy Cam, HTTP Requests for Processing, and ControlP5.

Windows

1. Install Processing using the link below:

   Getting Started / Processing.org

2. Per the website, "Processing comes as a .msi installer file. Locate the file in your Downloads folder and double-click it to install Processing."

3. Choose a typical install in the installer wizard. That has all the features that are required to run this program.
4. Once it finishes installing Processing, it is normally located in your Program Files on your C drive.
5. Once you find the folder, open up the folder and double click “Processing.exe” to open the program.
6. For the libraries, in Processing’s menu bar:

   → *Sketch* → *Import Library* → *Manage Libraries* and then search for Peasy Cam, HTTP Requests for Processing, and ControlP5.

Running the Application

MacOs

1. In Terminal, go to the project directory and then the cmake-build-debug folder.
2. Once there, type the following command and the server will start running:

   ./server
3. Open the main sketch in Processing titled *CEN3031.pde* it is located in → *frontend* → *SystemUI* directory.
4. Now, run the Processing application.
5. Once you’re finished and exited out of the program, close out of the server by entering the following command prompt:

   Ctrl + *c*

Windows

1. Doubleclick the “run_windows.bat” and that will open up the Crow server backend.
2. Open the main sketch in Processing titled *CEN3031.pde,* it is located in → *frontend* → *SystemUI* directory.
3. Now, run the Processing application.
4. Once you’re finished and exited out of the program, close out of the server by entering the following command prompt:

   Ctrl + *c*

3.3 Login and Access Credentials / API Keys

Elysium includes a basic local login system that allows users to sign in and access personalized data such as saved favorite exoplanets. User account information is stored in a local .txt file in plain text format, as the application is intended for offline educational use only. Upon opening the application up, a login screen will prompt for a username and password. These credentials are checked against entries in the local file; if the user is found and the password matches, the access is granted to the program. Accounts can also be created at this screen if the user does not already have one or wants to create another.

Elysium does not use any third-party APIs or requires any API keys. Everything is handled locally through CSV and TXT files and the application does not connect to the internet.

Section 4: Risk Management and Quality Attributes

4.1 Risk Management Plan

RISK	PROBABILITY	SEVERITY	MITIGATION STRATEGY
Backend fails to launch	Medium	Serious	Provide Error message in frontend. Make sure the file executable is bundled and working before release
Missing required files	High	Tolerable	Use relative paths and include default files in distribution. Throw an error if a file fails to open
Toolchain misconfiguration	Medium	Serious	Document the build process clearly
Frontend does not properly handle failed backend communication	Medium	Serious	Retry logic in Processing to connect backend
Crow server port already in use on user system	Low	Serious	Fallback to another port if unavailable
Poor code readability and maintainability	Medium	Tolerable	Add in-code documentation and modularize the logic to help with future contributions
Runtime crashes from unhandled edge cases	Medium	Serious	Expand unit tests to cover more cases and add safety checks

4.2 Software Quality Attributes

Usability:

Elysium was designed with a clear and intuitive interface using Processing, allowing users to navigate planetary data without the need of a tutorial. Side-by-side comparisons and climate scenarios are presented visually to support learning and ease of use for both students and educators.

Maintainability:

The system is modular, with separate backend and frontend components connected via requests through HTTP. Each part can be updated or debugged independently.

Reliability:

The core functionalities of the system were tested with both valid and invalid inputs to ensure consistent and accurate performance. Any failures such as missing files or connection issues are handled correctly.

Performance:

Due to the fact that all the operations are local and the data files are lightweight, Elysium performs smoothly without any noticeable lag. Backend responses are served quickly through the Crow web server and processed efficiently by the frontend.

Security:

This system is intended for local use anyway, but the system does avoid transmitting any sensitive data over networks. All user information is stored in a txt file with basic hashing of the password to have some sense of security, though it is not encrypted due to the educational use of the product.

References

[1] NASA Exoplanet Archive, *Confirmed Planets Table,* [Online]. Available: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=PSCompPars. [Accessed: Apr. 3, 2025].

[2] Processing Foundation, *Getting Started with Processing,* [Online]. Available: https://processing.org/tutorials/gettingstarted. [Accessed: Apr. 21, 2025].

[3] CrowCpp, *Crow: A C++ microframework for the web,* GitHub Repository, [Online]. Available: https://github.com/CrowCpp/Crow. [Accessed: Apr. 21, 2025].