In Your Orbit

Final Report

Version 1.0

Cloud-Based Planetary Ephemerides

United States Geological Survey

Mentor: Scott Larocca

Austin Carlile
Noah Schwartz
Minuka Trikawalagoda
Nicholas Gonzalez

The purpose of this design document is to explicitly detail the entire development process of our software solution

5/8/2025

Table of Contents

Introduction

Planetary science, particularly NASA missions, is a rapidly growing multi-billion-dollar industry reliant on processing planetary images. Sensor models convert coordinates into spatial reference systems, enabling scientists to reconstruct topography and geolocate imagery, supporting missions like the upcoming Europa Clipper. However, retrieving Image Support Data (ISD), essential for these models, requires navigating NASA's vast, exceeding a terabyte, and complex SPICE data system, which poses significant accessibility challenges. Additionally, the current system in place for accessing and processing ISD costs over $10,000 per month. These inefficiencies create significant obstacles for scientists, slowing down research and limiting accessibility. By addressing these issues, our project aims to improve the efficiency and accessibility of planetary image processing, ultimately, accelerating scientific progress.

To address this challenge, our capstone team is developing a cloud-based web service that streamlines ISD retrieval from the SPICE database. Built on existing USGS software and leveraging the services of FastAPI, this solution will enable users to access ISD without requiring full downloads of the SPICE dataset, which can span multiple terabytes. Our goal is to create a more user-friendly, scalable service that allows scientists, especially those new to the field, to shift their focus from data management to scientific discovery.

Process Overview

For our capstone project, we followed a modified SCRUM methodology. While we did not hold daily stand-ups, we conducted three weekly check-ins: one internal team meeting, one with our faculty mentor, and, in the Spring semester, one with the client. These meetings helped us maintain regular progress updates, clarify expectations, and receive timely feedback.

We structured our work into sprints aligned with project deliverables. Tasks were assigned and discussed during team meetings, and we presented our progress through regular demos at mentor and client sessions. This iterative process ensured transparency and continuous improvement.

Our team primarily used **Visual Studio Code** and **Notepad++** as development environments, with **WSL (Windows Subsystem for Linux)** to run the project environment consistently across machines. The codebase was collaboratively managed using **GitHub**, allowing all team members to contribute, review, and merge code.

Bug tracking was handled informally through direct communication on **Discord** and during in-person discussions in our shared Mobile Development course. For UML diagrams and system modeling, we used **Draw.io**, a tool we were all familiar with from previous coursework.

Our team operated with defined roles to streamline collaboration:

- **Austin** – *Team Lead & Client Liaison*: Managed scheduling, task delegation, and client communication.
- **Nicholas** – *Recorder*: Took and distributed meeting notes for team, mentor, and client sessions.
- **Noah** – *Release Manager*: Oversaw GitHub branching, commits, and version releases.
- **Minuka** – *Architect*: Ensured adherence to project architecture and technical standards.
- **All Members** – *Developers*: Contributed code across the stack, each leveraging their areas of strength.

Requirements

Requirements Acquisition Process

The Requirements were acquired through an iterative process involving collaboration with the United States Geological Survey (USGS), regular mentorship, and feedback loop driven by sprint reviews. The team began with initial discovery sessions to understand user pain points in retrieving Image Support Data (ISD) using NASA's ALE toolkit. Over time, requirements evolved through weekly meetings, prototype demonstrations, and continuous validation from domain experts, leading to a refined, user-centered specification.

Function Requirements
For the Web Service:

- Provide RESTful endpoints for ISD requests using mission-specific parameters.
- Validate inputs and return JSON-formatted ISDs via OpenAPI-compliant responses.
- Handle requests asynchronously to maximize responsiveness.

For the Caching Server:

- Implement a cache-aside pattern to store and retrieve recently generated ISDs.
- Track cache versions and perform periodic invalidation to ensure freshness.
- On the cache miss, notify the ISD engine to generate and cache new ISDs.

For the ISD Generation Engine:

- Use NASA's ALE tool process SPICE and generate Image Support Data.
- Compress and minimize JSON output for efficient transmission.
- Cache previous results to avoid redundant processing.

Non-Functional Requirements
Environmental:

- System must be fully containerized for platform independence.
- Service must run efficiently to minimize CPU/memory usage and cloud costs.

Scalability:

- The system must support automatic horizontal scaling for high volumes of concurrent ISD requests.

Reliability:

- All failures must require minimal domain knowledge to operate effectively.

Usability:

- API documentation must be auto-generated and accessible to new users.
- Interfaces must require minimal domain knowledge to operate effectively.

Security:

- All API inputs must be sanitized to prevent injection attacks.
- Data access must comply with NASA and USGS governance policies.

Maintainability:

- Code should follow modular design principles and be well-documented.
- Logs and monitoring should be implemented for system visibility.

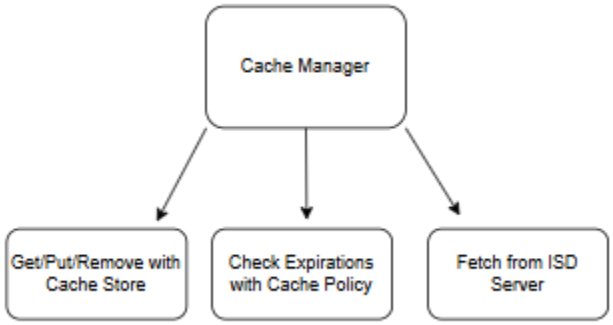
Architecture and Implementation

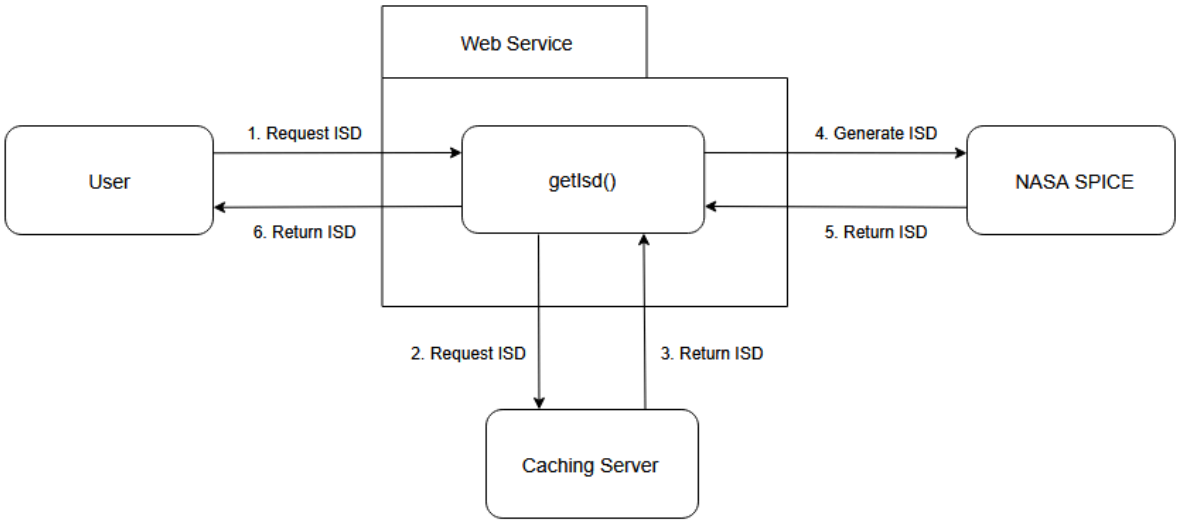
At a high level, the system is designed to support efficient generation, retrieval, and caching of Image Support Data (ISD) files for scientific users. It is structured around a modular service-oriented architecture hosted on AWS, employing FastAPI as the core web framework and leveraging containerized services.

Below is a breakdown of the core components and their interactions:

- **Web Service (FastAPI-based)**
    - **Responsibilities**: Handles client ISD requests via RESTful endpoints. Acts as the gateway to the caching layer and ISD generation engine.
    - **Features**: Automatic data validation, async request handling, and interactive OpenAPI docs.
    - **Typical Use Case**: Receives an ISD request → queries the Caching Server → either returns cached ISD or triggers ISD generation.
- **Caching Server**
    - **Responsibilities**: Stores recently generated ISDs to avoid redundant computation and reduce response times.

- **Features**: Cache-Aside pattern, version tracking, periodic invalidation.
- **Typical Use Case**: Checks for an ISD → if valid, returns it to Web Service → otherwise marks it stale for regeneration.





- **ISD Generation Engine (ALE + Compression Tools)**
  - **Responsibilities**: Generates ISDs from SPICE kernels using NASA's ALE tool, applies compression and memoization.
  - **Features**: JSON minimization, memoization for performance.
  - **Typical Use Case**: On cache miss, takes in parameters → queries SPICE database → returns and stores compressed ISD.

- **SPICE Database Integration**
    - **Responsibilities**: Provides the raw planetary data for ISD generation.
    - **Features**: Periodically monitored for updates that would trigger ISD re-generation.
- **User Interface (Not UI-heavy)**
    - **Responsibilities**: Sends formatted requests and receives JSON responses. Designed with scientists in mind for simplicity.

Testing

To ensure the reliability, functionality, and usability of our cloud-based ISD retrieval system, we employed a multi-layered testing strategy involving unit testing and integration testing. Our approach aimed not only to verify the correctness of individual software components, but also to validate the interactions between them and ensure that real-world users—namely, domain scientists—can effectively use the system.

Our testing plan was designed to reflect the specific architecture and technology stack of the system, as well as the expectations of our client. Because our implementation is based on Python and FastAPI, containerized with Docker, and hosted using AWS services, we prioritized automated and environment-specific testing.

The testing process can be broken into two primary categories:

1. **Unit Testing**:
   Focused on individual components such as the ISD generation module, caching logic, and RESTful API endpoints. These tests ensure that each function behaves as expected in isolation.
    - **Tools Used**: pytest, unittest.mock and FastAPI's built-in test client.
    - **Scope**: High coverage for core logic and error handling paths.

2. **Integration Testing**:
   Assessed the interoperability between the FastAPI Web Service, the DynamoDB-based caching layer, and the ALE-powered ISD generation engine. These tests verified the overall system behavior and data flow across service boundaries.

    - **Tools Used**: Docker containers to replicate the production environment, REST test clients, and scripted integration scenarios.

    - **Scope**: Focused on API-caching interactions, cache misses/hits, and regeneration of ISDs using SPICE data.

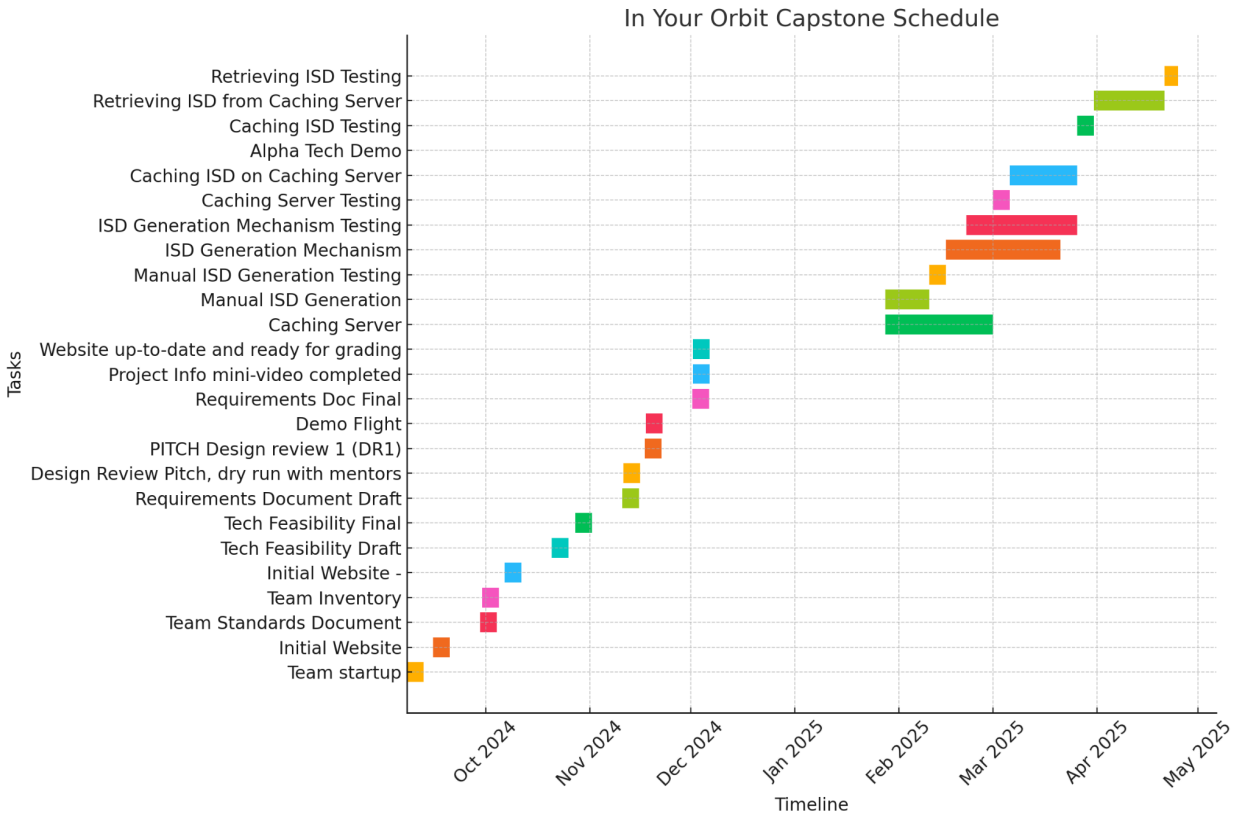
Each layer of testing reflects a need to validate the software from a different perspective:

- **Unit tests** ensured internal correctness and supported maintainability.

- **Integration tests** confirmed that separately developed modules operated as expected when connected.

The results of our testing led to us improving our error handling. Integration tests uncovered edge cases around malformed or incomplete ISD parameters. This prompted the addition of more robust input validation in our FastAPI endpoints.

Project Timeline



In Your Orbit Project Schedule

Phase 1: Project Initiation (September – October 2024)

Team Responsibilities:

- Team startup and role assignment
- Development of the initial website
- Creation of the team inventory and standards documentation

Objective:

Lay the foundation for teamwork, communication tools, and shared expectations.

Phase 2: Research & Planning (October – November 2024)

Team Responsibilities:

- Drafting the Technical Feasibility documents (Draft and Final versions)
- Preparing the Design Review Pitch (DR1) and presenting to mentors
- Converting project information into a mini-video
- Drafting the Requirements Document

Objective:
Establish a clear vision of the project's scope, evaluate feasibility, and plan technical direction.

Phase 3: Requirements & Website Finalization (November – December 2024)

Team Responsibilities:

- Finalizing the Requirements Document
- Completing the public-facing website for evaluation
- Conducting a preliminary Demo Flight

Objective:
Solidify project requirements, confirm understanding of the system, and communicate the vision effectively online.

Phase 4: Core Development & Testing (January – March 2025)

Team Responsibilities:

- Develop and test the following components:
    - Manual ISD Generation

- Caching Server
- ISD Generation Mechanism
- Manual ISD Generation Testing
- Caching Server Testing
- Caching ISD on Server
- Caching ISD Testing
- Retrieving ISD from Caching Server
- Retrieving ISD Testing

Objective:
Build, implement, and verify the core functionalities of the ISD system, focusing on generation, caching, and retrieval.

The project is currently complete and in the wrapping up stages. The team is finishing up the Final Report and User Manual before delivering the final product to our client.

Future Work

While the current system significantly enhances research efficiency by streamlining access to NASA Instrument Support Data (ISDs), there are clear opportunities for future expansion. The United States Geological Survey (USGS) intends to build on this foundation in two major areas:

- **Scalability Enhancements**: Future iterations will focus on scaling the infrastructure to support over 200,000 simultaneous ISD requests, ensuring the system remains responsive under heavy scientific demand.
- **Dataset Expansion**: The current implementation primarily supports ISDs from the Viking mission. Upcoming work will broaden the system's scope to include ISDs from all NASA missions, allowing researchers to access a comprehensive archive of planetary mission data through a unified interface.

These efforts will further strengthen the system's impact on scientific accessibility, reliability, and long-term utility.

Conclusion

The In Your Orbit project was initiated to address a critical challenge faced by the United States Geological Survey (USGS): the inefficiency and complexity of retrieving Instrument Support Data (ISDs) for planetary research. Scientists relied on time-consuming manual processes and disjointed data sources, limiting both productivity and accessibility. Motivated by the need to streamline this workflow and enhance research efficiency, our team set out to develop a scalable, cloud-based solution tailored specifically to these scientific and technical needs.

To meet the client's goals, our team built a modular, Python-based system featuring a FastAPI-powered web service, a caching layer, and an ISD generation engine integrated with NASA's ALE tool.

Key features of our system include:

- **Efficient ISD Retrieval**: Rapid access to previously generated ISDs via intelligent caching.
- **Reduced Data Access Costs**: Minimized redundant computations and bandwidth use through memoization and compression strategies.
- **Improved Accessibility**: A streamlined user interface and simplified request process designed specifically for planetary scientists.

The project's impact for USGS has been significant. By automating ISD generation and retrieval, the system drastically reduces the time required for scientists to access essential data. Caching and optimized data flows reduce compute and storage costs, while the scalable infrastructure ensures the system can grow with demand. Scientists now spend less time managing data logistics and more time on research.

Beyond the immediate client, this solution lays the groundwork for broader scientific collaboration and reuse. With future expansions planned to support all NASA missions, this system has the potential to become a central platform for planetary mission data retrieval, benefiting the wider research community.

Reflecting on the project, our team gained valuable experience in full-stack system development, cloud infrastructure, and domain-specific problem-solving. Collaborating with a real-world client deepened our understanding of translating technical requirements into functional solutions. The Capstone experience challenged us to balance performance, scalability, and usability—skills that will serve us well in future professional endeavors.

Glossary

- **ISD** - Image Support Data
- **SPICE** - Spacecraft Planetary Instrument C-Matrix Events
- **ALE** - Abstraction Layer for Ephemerides

Appendix A: Development Environment and Toolchain

Hardware:

- **Development Platforms:** The primary development platform used was Linux, facilitated by WSL (Windows Subsystem for Linux), to ensure a consistent environment across team members' machines.
- **General Hardware Specifications:** The development team used standard personal computers capable of running development environments like Visual Studio Code, running WSL, and handling tasks like code compilation and containerization.
- **Minimum Hardware Requirements:** To develop and run the tools mentioned in the document, a system capable of running a Linux virtualized environment (via WSL), a modern IDE, and Docker would be necessary. Additionally, a relatively modern processor (e.g., Intel Core i5 or equivalent), sufficient RAM (e.g., 8GB or more), and adequate storage.

Toolchain:

- **Visual Studio Code and Notepad++:** These were used as the primary development environments for writing and editing code.
    - VS Code provided a robust and extensible environment for coding.
    - Notepad++ was likely used for quick edits and viewing files.
- **WSL (Windows Subsystem for Linux):** This allowed the team to run a Linux environment directly within Windows, ensuring consistency in the development environment and access to Linux-specific tools.
- **Git and GitHub:** Git was used for version control, and GitHub served as the platform for collaborative code management, including storage, tracking changes, and merging contributions.

- **Draw.io:** This tool was used for creating UML diagrams and system modeling, aiding in the design and documentation of the system's architecture.

Setup:

The steps include:

1. Begin with a Linux-based operating system.
2. Install Miniforge.
3. Install ISIS and set environment variables.
4. Clone the Web Service Git Repository.
5. Install Python-pip and necessary dependencies.
6. Set AWS access keys and region to environment variables.
7. Activate the isis conda environment.
8. Set up AWS Account and IAM User
9. Give IAM User DynamoDB Permission
10. Deploy DynamoDB CloudFormation Stack
11. Run the uvicorn command to start the Web Service.

- For more detailed instructions, refer to the "Installation" section of the User Manual.

Production Cycle:

- The development involved editing Python code (for the Web Service and other components).
- The "uvicorn --reload" flag means the development environment supports hot-reloading, where changes to the code are automatically reflected without a full redeployment.
- The deployment process involves containerization using Docker.