In Your Orbit

Design Document

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 implementation plans of our software solution

1/26/2025

Table of Contents

1. Introduction 3

2. Implementation Overview 4

3. Architectural Overview 6

4. Module and Interface Descriptions 9

5. Implementation Plan 17

6. Conclusion 19

1. 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 and complex SPICE data system, which poses significant accessibility challenges. As missions expand, managing ISD retrieval becomes increasingly difficult, hindering research efficiency.

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 AWS services, 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.

From a software design perspective, our solution must meet several key user-level and functional requirements. It should provide a reliable and efficient method for retrieving ISD, ensuring high availability and scalability via cloud infrastructure. Performance requirements include minimizing latency in data retrieval while maintaining accuracy. Additionally, environmental constraints such as data security, integration with existing USGS tools, and compliance with NASA's data protocols must be addressed.

By successfully implementing this system, we will enhance both USGS and NASA's ability to process planetary imagery at scale, improving workflows in both on-premise and cloud environments. More broadly, this project aims to lower entry barriers for future planetary science researchers by making high-quality ISD retrieval more accessible, fostering innovation and scientific progress in the field.

2. Implementation Overview

The solution proposed for our client's problem is a cloud-based ISD generator which will offer an easy to navigate service which will simplify the process of ISD retrieval for scientists. A RESTful web service which facilitates ISD generation in a more optimized manner will be created using Python with an addition of a caching system to reduce the ISD generation time by retrieving data which has been previously stored. Thus, commonly requested ISDs will be held in the caching server ensuring that more frequently used data is more readily accessible for users. The solution will be hosted on an Amazon Web Services (AWS) server utilizing an Elastic Container Service (ECS) container system and Elastic Compute Cloud (EC2) for autoscaling. This new system will make generating ISDs much more efficient for scientists, in time and user experience.

The approach we will be taking with our system resembles multiple software design patterns. Specifically our system echoes the Cache-Aside pattern and the Producer-Consumer design pattern. Our cache will be available to store recent ISDs and will be communicating with our users to sort and resolve requests.

Frameworks and Packages

FastAPI – A modern web framework that can handle high concurrency efficiently. It offers asynchronous support, automatic data validation, and interactive API documentation, making it both scalable and developer-friendly for our project.

ECS (Elastic Container Service) – A flexible and customizable container orchestration service that supports both **EC2** and **Elasticache**, allowing our application to scale dynamically while still ensuring our resources are allocated appropriately.

ElastiCache – A fully managed, in-memory caching service that enhances application speed by reducing database load times. It supports auto-scaling, handles complex caching strategies, and improves responsiveness to ensure ISDs can be generated and queried efficiently and effectively.

Combination of JSON minimization, Gzip, and Memoization – These techniques collectively improve performance and scalability. JSON minimization reduces payload size, Gzip compresses data for faster transmission, and memoization optimizes computation by caching results ensuring that our system is optimized for any user and not only high performance systems.

Docker – Provides containerization, enabling applications and services (such as caching systems, web services, and databases) to run in isolated, portable environments. With Docker we can test in a safe environment while containing all pieces of the system.

3. Architectural Overview

To understand how the system operates, we'll examine its architecture, key components, and how they interact. This section outlines the responsibilities of each module and the flow of information, ensuring a seamless user experience while maintaining scalability and flexibility.

Web Service

- Hosted on Amazon AWS using EC2 instances within ECS containers.
- Serves as the entry point for user queries related to ISD retrieval and generation.
- Queries the Caching Server for stored ISDs before resorting to new generation.
- Uses FastAPI to handle user requests in a RESTful manner.
- If the ISD is not found in the cache, it initiates data retrieval and ISD generation.

Caching Server

- Stores previously requested ISDs to optimize retrieval time and reduce redundant ISD generation.
- Implements a custom ID system to track different ISD versions.
- Automatically updates stored ISDs when changes occur in the source SPICE database.
- Communicates with the Web Service using a REST API over a TCP connection.

ISD Generation Engine

- Utilizes NASA's ALE tool to generate ISD files based on user parameters.
- Compresses ISD files in JSON format for efficient storage and transmission.
- Uses SPICE kernel data to derive planetary spatial reference information.

- Updates the Caching Server with newly generated ISDs.

SPICE Database

- Serves as the authoritative source for planetary mission data.
- Provides raw data for ISD generation.
- The Caching Server periodically checks for updates in SPICE kernels.

Load Balancing and Scaling

- EC2 instances scale dynamically based on system demand.
- Ensures the system handles up to 200,000 simultaneous requests.
- Optimizes cost and performance by adjusting resource allocation.

User Interface

- Provides a user-friendly queryable interface for scientists.
- Sends requests to the Web Service for ISD retrieval.
- Displays ISD metadata and retrieval status.

Communication Mechanisms and Information Flow

User Query Processing:

1. A user submits an ISD request via the web portal or API.
2. The Web Service checks the Caching Server for a stored ISD.
3. If found, the ISD is retrieved and returned to the user.
4. If not found, the Web Service requests ISD generation.

ISD Generation Workflow:

1. The Web Service extracts necessary parameters and queries SPICE data.
2. ALE generates the ISD, which is compressed and stored.
3. The ISD is saved in the Caching Server for future retrieval.
4. The generated ISD is returned to the user.

Cache Update and Synchronization:

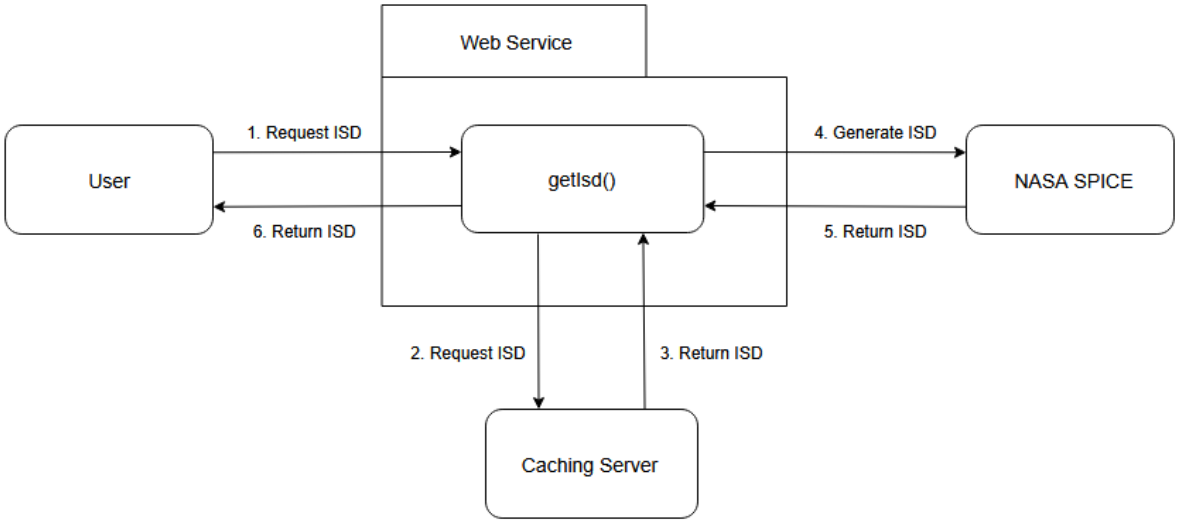
1. The Caching Server monitors SPICE kernel updates.
2. If a change is detected, the ISD is regenerated.
3. The old ISD version is retained for historical reference.

Architectural Influences

The system follows multiple architectural styles to ensure robustness:

- Microservices Architecture:
    - The Web Service and Caching Server are loosely coupled components.
    - Each service scales independently based on demand.
- Client-Server Architecture:
    - The Web Service acts as the central interface, handling client queries and interfacing with backend services.
- Event-Driven Architecture:
    - The Caching Server detects changes in SPICE data and triggers ISD updates.

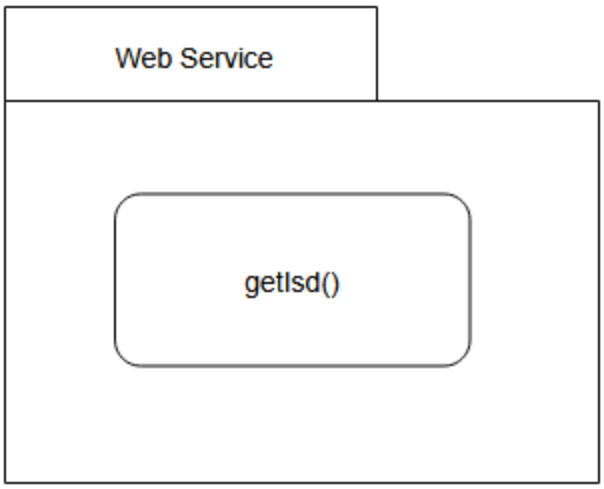
- RESTful API Design:
    - All communication between services follows RESTful HTTP requests, ensuring interoperability.



4. Module and Interface Descriptions

The system consists of a few key modules that comprise and host the Web Service, Caching Server and Testing Mechanisms. This section will break down these modules and describe their internal mechanisms, responsibilities and public interfaces.

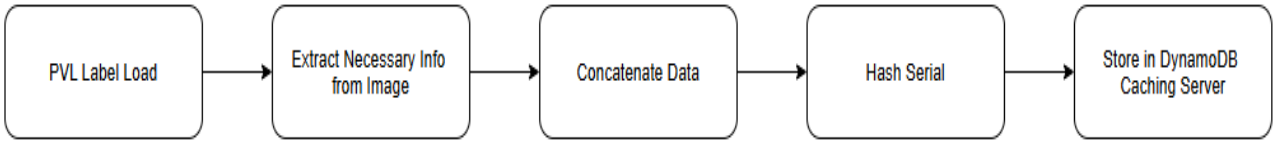
1. **Web Service**
   The Web Service will provide an API for the user to request Imagery Support Data for a given image. The Web Service will generate the ISD or request it from the Caching Server and return it to the user.



The public interface of the Web Service will include a singular function called getIsd(), that will be a GET request to retrieve an ISD either from generation or retrieval from the caching server if the ISD is already cached. The user will call the function from a command line either manually or inside an external application.

2. ISD Generation

The generation of ISD is through the use of the ALE library and SPICE database. The ALE library's usage lies in functions such as **isd_generate()**, which contains code necessary for parsing .lbl files, the format that ISDs are typically in. This is done using Parameter Value Language, hereafter PVL, which is a markup language similar to XML. The ALE library uses PVL to turn the label files into more readable JSON files. The SPICE database is where these ISDs are stored and are accessed through the **spiceinit()** function which accesses the web-based SPICE database.



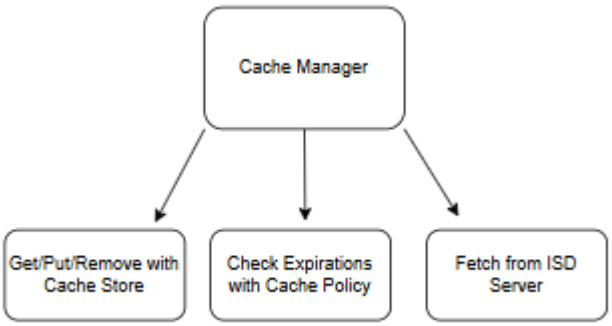
ISD will be generated and loaded to the caching server after various steps are handled in a Python file. First, the ISD file location is found and the commands mentioned above are run (i.e. spiceinit and isd_generate). Then, using PVL, the less important parts of the label are removed to create a new “mini-label”. The mini label is then broken apart and concatenated into one identification string. This string is then inputted to a hashing algorithm that finally spits out a unique hash ID to identify the label when cached.

3. ISD Retrieval

The retrieval of ISD is a major process the capstone project is aiming to solve in a more efficient manner. As seen in the diagram from Section 4.2, the process for generating ISD will result in a uniquely created ID that consists of hash code. The hash code is a compilation of the most vital information from an ISD that can still be generated and uniquely identified. Once an ISD is tagged with this unique hash code ID, it is then sent to the Amazon DynamoDB caching server for storage until a user requests this image again. The retrieval of an ISD is done through a query on the FastAPI Web Interface (see Section 4.1). There, the Interface will query for the hash code, if found, the ISD is sent back to the user, if not, the system will quickly generate the ISD, return to the user and cache the hash code and image in the database for easier retrieval next time.

4. **Caching server**

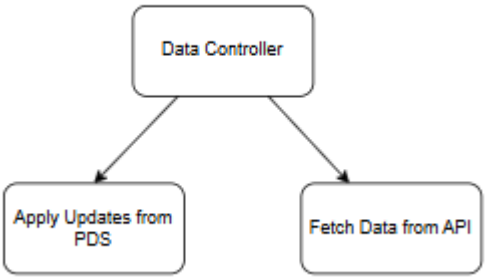
The Caching Server stores frequently accessed ISD data temporarily to reduce latency and offload the main server. It retrieves data quickly for repeated requests and ensures better performance by reducing the need to query the ISD Server frequently.



The CacheManager() is responsible for managing the cache, determining when to fetch data from the cache or forward the request to the ISD Server. It handles operations like storing data, retrieving it, and invalidating cache entries. The CacheStore() handles the actual storage and retrieval of cached data. The CachePolicy() defines the rules for data expiry, ensuring that latest ISD information is stored.

5. **ISD Cache**

The ISD Server provides the main data from the external API. It is the authoritative data source for the system and handles real-time data retrieval. The Caching Server relies on it for fresh data when cache misses occur.



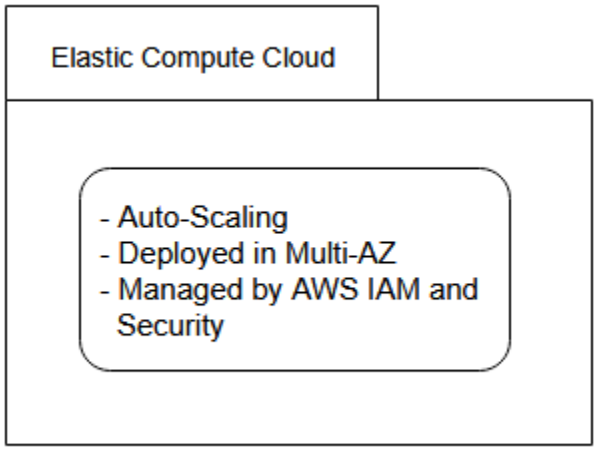
The DataController() manages the overall data retrieval and update operations, handling requests for data and pushing updates to the system. It interacts with the DataFetcher(), which retrieves data from external sources like our API. Additionally, the DataUpdater() handles data modifications, ensuring that updates are applied when required by acquiring the new info from our PDS (Planetary Data System) archive. These components work together to provide accurate and up-to-date data for the system.

6. **Elastic Compute Cloud (EC2)**

Amazon Elastic Compute Cloud (EC2) serves as the backbone of the system, as it provides scalable and reliable computing resources for hosting several system components. The web service will be hosted in the EC2 instances for processing user requests, handling ISD retrieval, and coordinating with the caching server and the ISD generation engine. Its scalability feature assures that the system will scale dynamically with respect to capacity whenever the traffic demand fluctuates and assure consistently high performance.

The configuration for the auto scaling of the EC2 instances is done through AWS Auto Scaling, letting the system manage itself better and optimize resources for cost efficiency.

In high-user-activity periods, more instances are provisioned to support the workload, while in the case of low traffic, the resources scale down to save on costs. The instances of EC2 are deployed in multiple availability zones to ensure high availability and fault tolerance, and ELB will distribute incoming traffic across all the instances evenly. Security is very critical to an EC2 deployment. The system will use the AWS Identity, Access Management(IAM), and security groups of AWS for security to grant only legitimate access. Amazon CloudWatch gives the potential for real-time monitoring, in addition to enabling the system proactive resource optimization against the metric variables of CPU and network.

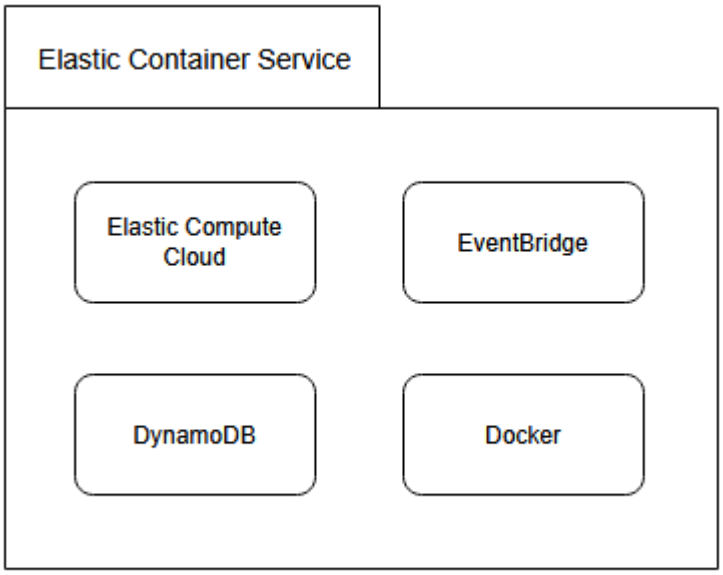


The containerized applications will be hosted on EC2 instances in conjunction with the Amazon Elastic Container Service for effective management and orchestration. Such architecture will ensure that all services, including the caching server and the ISD generation engine, work cohesively in a unified environment. With its scalability, high

availability, strong security, and performance optimization all combined, EC2 forms the basis for a reliable and efficient cloud-based ISD generation solution.

7. **Elastic Container Service**

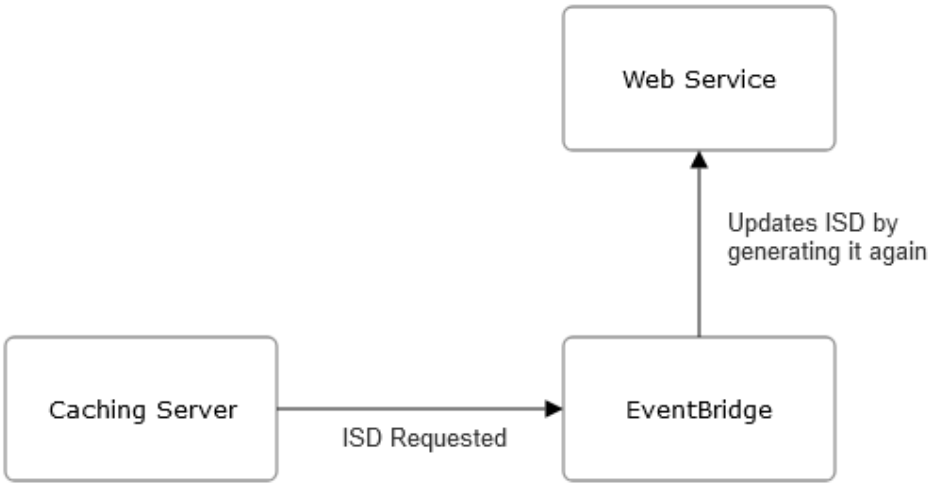
The Elastic Container Service (ECS) will host and link together all of the Amazon AWS services required for the complete solution, including DynamoDB, EventBridge, Docker and EC2 Servers. Since the EC2 Servers will host the Web Service, the entire solution will sit inside of a singular ECS container.



The ECS container will only be accessed and controlled in the back-end of the program and has no direct access from the user.

8. **EventBridge**

EventBridge is an Amazon AWS service that will trigger code processes based on system events. Our solution will utilize it to run ISD cache updates every time a previously generated ISD is requested by a user. When an ISD already exists in the cache, EventBridge will request a re-generation of the ISD from the Web Service and Cache, then return the new ISD to the user.

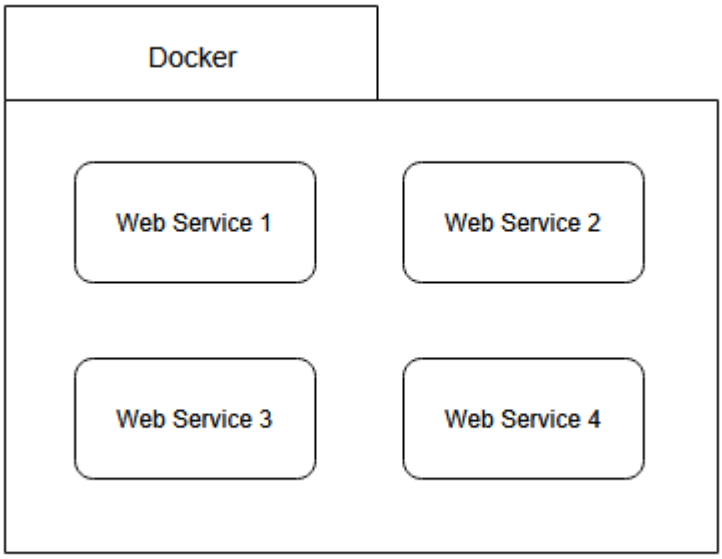


EventBridge will run automatically with no input from the user or publicly accessible components. The user will only interact with EventBridge by requesting a previously generated ISD.

9. **Docker**

Docker is a container program that allows us to run multiple instances of a program or component simultaneously, as well as sandbox an instance for testing without disrupting

the rest of the solution. We will be utilizing docker to allow us to test functionality and edge cases of our program without causing unintended effects on the rest of the system.



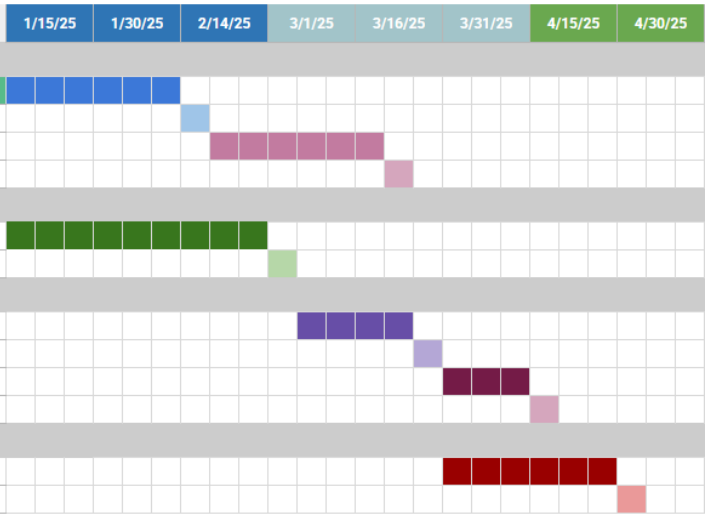
Docker will be used as an internal testing mechanism and will not be publicly accessible. The user will not be able to interact with or use Docker in any way.

5. Implementation Plan

Planning ahead, we have created a timeline to document and prescribe the progress and steps we are taking to implement our solution. As shown in the Gantt chart below, the main phases of our implementation plan are the ISD Generation, Caching Server, ISD Caching and ISD Updating

phases. Each main phase is broken down into subphases that are followed by a five day testing period. The ISD Generation and Updating phases have multiple subphases, with the ISD Generation phase containing Manual ISD Generation and ISD Generation Mechanism subphases, and the ISD Caching phase containing Caching ISD on Caching Server and Retrieving ISD from Caching Server subphases

TASK TITLE	TASK OWNER	START DATE	END DATE	PCT OF TASK COMPLETE
ISD Generation				
Manual ISD Generation	Team	1/28/25	2/10/25	50%
Manual ISD Generation Testing	Team	2/10/25	2/15/25	0%
ISD Generation Mechanism	Team	2/15/25	3/21/25	0%
ISD Generation Mechanism Testing	Team	2/21/25	3/26/25	0%
Caching Server				
Caching Server	Team	1/28/25	3/1/25	0%
Caching Server Testing	Team	3/1/25	3/6/25	0%
ISD Caching				
Caching ISD on Caching Server	Team	3/6/25	3/26/25	0%
Caching ISD Testing	Team	3/26/25	3/31/25	0%
Retrieving ISD from Caching Server	Team	3/31/25	4/21/25	0%
Retrieving ISD Testing	Team	4/21/25	4/25/25	0%
ISD Updating				
Automatic ISD Updating	Team	3/31/25	4/31/25	0%
Automatic ISD Update Testing	Team	4/31/25	5/5/25	0%



To begin our implementation timeline, we will begin by testing out ISD generation, this will be done through manual testing until we figure out how to correctly generate ISDs. After figuring out the ISD generation process, we will then develop a mechanism to generate ISDs automatically and deploy it on a web service. At the same time as we test ISD generation, we will begin working on creating a caching server. Following correctly setting up our caching server, we will find out how to cache ISDs on it and implement the mechanisms to accomplish that on our web service. Once we have done that, we will create a system for retrieving ISDs that we have already cached. Finally, we will also be developing a way to automatically update ISDs

that we have stored in our caching server. After the development and testing of automatic ISD updating, our solution will be fully implemented and all the different components will be working together to accomplish our complete solution.

6. Conclusion

In conclusion, we are developing a streamlined cloud-based web service for generating planetary image support data for researchers and scientists. This project will help USGS, NASA and all planetary scientists process planetary imagery and will lower the barrier for future scientists by making ISD generation and retrieval more accessible. Our solution will encompass various amazon AWS services to create a web service and caching server for efficient ISD generation and retrieval. This will allow researchers and scientists to easily study planetary ephemerides without needing to do all the leg work themselves. We will be utilizing FastAPI, Amazon AWS services, including ECS, EC2 and DynamoDB, as well as Docker for testing solution components. We will begin our implementation process by creating an ISD generation mechanism, followed by a caching server for ISDs and finally creating a web service that can automatically generate and update ISDs, then return them to the user. This design document is a way for us to detail the exact specifications of how we plan to implement our solution, creating a schedule for our team to follow and a method for communicating our progress to our client. This is the first step in the process of developing our working and complete solution to help planetary scientists of all calibers around the world.