InSpace Application

Software Requirements Documentation (SRD):

Date: 11/23/2020

Team: It's Not a Bug, It’s a Feature

Members: Sytiva Wheeler, Taylor Smith, Logan Hunt

WE HAVE ABIDED BY THE UNCG Academic Integrity Policy ON THIS ASSIGNMENT.

Table of Contents

1. Introduction

2. Overall Description

3. Functional Requirements

4. Technical Requirements

5. NonFunctional Requirements

1. Introduction

1.1 *Purpose*

Demonstrate to users an interactive view of the solar system that our planet Earth resides in. From this view, the user will have access to information about the different celestial bodies that inhabit it. The types of information accessible includes a celestial body's name, eccentricity, axial tilt, and more.

1.2 *Intended Audience*

The software discussed in this document is intended for Ike Quigley, the professor grading the coursework that this software was made for as well as people interested in astrological data who want an easy to use interface to navigate various bodies in the solar system.

1.3 *Definitions/ Astronomical Jargon*

- Semimajor Axis: Longest radius of a planet's orbit and average distance of a planet to the sun.
- Perihelion: Position of a planet's closest approach to the sun.
- Aphelion: Position of a planet's furthest distance from the sun.
- Eccentricity: Parameter that determines the amount by which a planet's orbit deviates from being a perfect circle.
- Inclination: The tilt of an object's orbit around another celestial body.
- Equatorial Radius: Distance from a planet's center or core to its equator.
- Sideral Orbit: Time of an orbit for one celestial object around another.
- Sideral Rotation: Solar rotation period at the equator of a celestial body.
- Axial Tilt: Degrees at which a celestial object tilts on its axis.

1.4 *Scope*

Providing access to information about the celestial bodies in our solar system, like their orbits, moons, rotations and other astronomical information. Can also provide information about the time sunrise,sunset, moonrise and moonset takes place in a day using the location of the user.

1.5 *Technical Challenges*

Many of the technical challenges that came up in the making of this project related mostly to inexperience. None of the members have dealt with API before starting on the project. Also, many of the API found did not work correctly, were no longer working, were expensive or did not have the correct astronomical information we required. There was confusion on how MVC architecture worked and how the project should be structured. Formulating the math for the rendering engine proved especially difficult to get right. Keeping track of the orbits of planets and moons relative to each other and displaying them was very time consuming.

1.6 *References*

Listed below are the API used in the project as well as links to the websites in which their documentation can be found.

Solar System API - Solar System Open Data API used in project

IP Address API - IPInfo Geolocation API used

IPGeolocation Astronomy API - Astronomy API used in project

2. Overall Description

2.1 *Product Features*

Cosmic Perspective features a clean GUI with all 8 planets orbiting the sun. The view of the planets can be zoomed in on and zoomed out of by clicking and dragging. Clicking on the planets provides the user with information on that celestial body, which is accessed using the Solar System Open Data API. Saves info on user's location as well as sun and moon rising and setting information to a data store.

2.2 *User Characteristics*

Anyone who would like to know more about the solar system if they do not have an advanced understanding of it already.

2.3 *Operating Environment*

The operating environment is any computer (desktop/laptop) that uses the windows OS.

2.4 *Design and Implementation Constraints*

Determining whether an error with an API was the fault of the API, the programmer, or neither. For example, it can be a case of neither parties fault when you use an api to find the time the moon rises and it comes up null. This can happen not because of the api or programmer, but because the moon does not rise every night due to its constant and earth's rotation. It can happen when there is a new moon as well, or a phase of the moon in which it cannot be seen.

2.5 *Assumptions and Dependencies*

The Solar System OpenData API will always be used, the app will be a windows desktop application, JavaFX will always be used.

3. Functional Requirements

3.1 *Primary*

Display to users information about the celestial bodies of our solar system when clicked.

3.2 *Secondary*

- Show users an accurate model of where the solar system's celestial bodies are and how fast they orbit the Sun.
- Saves the user the sunrise/sunset and moonrise/moonset times for their area to a data store(area retrieved with location API). Info will be saved to a text file.
- Saves the location information of the user (Ip address, city, state, country, location coordinates) to a data store. Used for getting sunset/rise and moonset/rise data. Info will be saved to a text file.

4. Technical Requirements

4.1 *Operating Systems/Compatibility*

Compatible with the Windows OS and any OS capable of running java programs (any system that has a JVM).

4.2 *Interface Requirements*

4.2.1 User Interface: Users must use the mouse connected to their computer to click on planets in the interactive solar system diagram for the information to display.

4.2.2 Hardware Interface: User must have a computer with a working internet/network adapter, and a computer mouse.

4.2.3 Software Interface: Java, JSON, Organized class structure based on MVC architecture. API Interfaces, Adapters, Translators, Services package used to hold “Services” to be used by the functional Models package, controllers to connect views and models, events package used to handle user interface/hardware interface interaction, and listeners to define interfaces for events.

4.2.4 Communications Interface: User must establish connections to the following APIs (linked above):

a) Solar System OpenData API
b) IPGEOLocation API (Astronomy features only)
c) IPInfo API

5. Nonfunctional Requirements

5.1 *Performance Requirements*

Should make calls to the API whenever a user clicks a planet.

5.2 *Safety/Recovery Requirements*

In the event of a crash the software will

5.3 *Security Requirements*

No login required or security of any kind. Anyone can access or use this software.

5.4 *Policy Requirements*

MVC Architecture compliant, follows Ike Quigley's coding style guidelines, Connects to at least one API, and makes use of a persistent data store.

5.5 *Software Quality Attributes*

5.5.1 Availability: 24 hours 7 days each week. None of the software included in the program has time restrictions.

5.5.2 Correctness: The information of the planets must be up to date (i.e. Earth's moon/ any other moons are not planets, the sun is a star, not a planet).

5.5.3 Maintainability: If any API becomes deprecated, multiple backup API should be tested for accuracy before being considered for use. Its translator will use the same interface as the previous. Only API url,access keys/credentials and helper methods should need to be updated.

5.5.4 Reusability: Location and sunrise/set API classes can be used with other programs given that they need the same information. If more information is needed, minimal updates to the classes are required. APIConnect class can continue to be used for any API that requires a JSON as a string without needing to update it.

5.5.5 Portability: This software is able to be ported to any windows 10 operating system or any hardware with a JVM or with the ability to install a JVM.(How easy is it to port to different hardware/OS in the future)

5.6 Process Requirements:

5.6.1 Development Process Used:

Agile/Scrums and Iterative Process

5.6.2 Time Constraints:

The software must have finished development by December 1, 2020.

5.6.3 Cost:

$0

5.6.4 Delivery Date:

December 1, 2020