

**Software Design  
Document  
for  
Collaborative Visualization for  
Solar System Treks**

**Version 1.0 approved**

**Prepared by Christopher Smallwood, Abdullah Alshebly  
Stanley Do, Montague La France, Zipeng Guo, Johnny Lee, Jose Garcia,  
Miguel Sanchez, Odasys Soberanes, David Tang**

**Sponsor: NASA JPL**

**May 14, 2021**

Table of Contents.....	2
Revision History.....	4
1. Introduction.....	6
1.1. Purpose.....	6
1.2. Document Conventions.....	6
1.3. Intended Audience and Reading Suggestions.....	6
1.4. System Overview.....	6
2. Design Considerations.....	7
2.1. Assumptions and dependencies.....	7
2.2. General Constraints.....	7
2.3. Goals and Guidelines.....	7
2.4. Development Methods.....	7
3. Architectural Strategies.....	8
4. System Architecture.....	11
5. Policies and Tactics.....	14
5.1. Specific Products Used.....	14
5.2. Requirements traceability.....	14
5.3. Testing the software.....	14
6. Detailed System Design.....	18
6.1. User Interface Module.....	18
6.1.1 Responsibilities.....	18
6.1.2 Constraints.....	18
6.1.3 Composition.....	18
6.1.4 Uses/Interactions.....	19
6.1.5 Resources.....	19
6.2. Main Control Module.....	19
6.2.1 Responsibilities.....	19
6.2.2 Constraints.....	19
6.2.3 Composition.....	19
6.2.4 Uses/Interactions.....	19
6.2.5 Resources.....	19
6.3. Chat Module.....	20
6.3.1 Responsibilities.....	20
6.3.2 Constraints.....	20
6.3.3 Composition.....	20
6.3.4 Uses/Interactions.....	20
6.3.5 Resources.....	20
6.4. Tools Module.....	20
6.4.1 Responsibilities.....	20
6.4.2 Constraints.....	21
6.4.3 Composition.....	21
6.4.4 Uses/Interactions.....	21
6.4.5 Resources.....	21
6.5. Imports/Exports Module.....	21
6.5.1 Responsibilities.....	21
6.5.2 Constraints.....	21

6.5.3	Composition.....	21
6.5.4	Uses/Interactions.....	21
6.5.5	Resources.....	21
6.6.	States Module.....	22
6.6.1	Responsibilities.....	22
6.6.2	Constraints.....	22
6.6.3	Composition.....	22
6.6.4	Uses/Interactions.....	22
6.6.5	Resources.....	22
6.7.	Collaborative Session Module.....	22
6.7.1	Responsibilities.....	22
6.7.2	Constraints.....	22
6.7.3	Composition.....	22
6.7.4	Uses/Interactions.....	23
6.7.5	Resources.....	23
6.8.	WebSocket Server Module.....	23
6.8.1	Responsibilities.....	23
6.8.2	Constraints.....	23
6.8.3	Composition.....	23
6.8.4	Uses/Interactions.....	23
6.8.5	Resources.....	24
7.	Detailed Lower level Component Design.....	25
7.1.	CollaborativeSession.js.....	25
7.1.1	Classification.....	25
7.1.2	Processing Detail.....	25
7.1.3.2	Restrictions/Limitations.....	25
7.1.3.3	Performance Issues.....	25
7.1.3.4	Design Constraints.....	25
7.1.3.5	Processing Detail For Each Operation.....	25
7.2.	Chat Module.....	25
7.2.1	Classification.....	25
7.2.2	Interface Description.....	25
7.2.4	Processing Detail.....	25
7.2.4.1	Restrictions/Limitations.....	26
7.2.4.2	Performance Issues.....	26
7.2.4.3	Design Constraints.....	26
8.	Database Design.....	27
9.	User Interface.....	28
9.1.	Overview of User Interface.....	28
9.2.	Screen Frameworks or Images.....	28
9.3.	User Interface Flow Model.....	31
10.	Requirements Validation and Verification.....	32
11.	Glossary.....	35
12.	References.....	37

## Revision History

Name	Date	Reason For Changes	Version
Jose Garcia	12/6/2020	Added Section 9 User Interface	0.1
David Tang	12/6/2020	Added Section 1.1, 2.1, 2.2	0.1
David Tang	12/8/2020	Added Section 1.3 and Glossary	0.1
Christopher Smallwood	12/29/2020	Added Section 12 References	0.2
Montague La France	12/29/2020	Added Section 3 Architectural Strategies, Section 5 Policies & Tactics	0.2
Stanley Do	12/29/2020	Assisted with Sec 3, started on Sec 4 & 6.	0.2
Johnny Lee	12/29/2020	Checked for missing terms in Glossary	0.2
Jose Garcia	12/29/2020	Revised Section 9 User Interface	0.2
Zipeng Guo	12/29/2020	Added Section 1.2,1.4,2.4	0.2
David Tang	12/30/2020	Started Section 6	0.2
Stanley Do	1/7/2021	Added additional modules to Section 6	0.2
Stanley Do	1/13/2021	Made changes to some modules, added new version of DFD	0.2
Stanley Do	1/21/2021	Finished first version of Section 6, made general revisions	0.2

David Tang	1/28/2021	Revised Section 1 to 5.	0.2
David Tang	1/30/2021	Revised Section 6 and 9.	0.2
David Tang	5/8/2021	Updated Section 1 to 4.	1.0
David Tang	5/9/2021	Added to Section 4, 5, 6, 9, 10. Updated database information in Section 8.	1.0
Stanley Do	5/9/2021	Updated the DFD Level 1, removed useless data, added instructions for installing and running code, created a new UI diagram, detailed component design for collaborative sessions.	1.0
David Tang	5/10/2021	Updated Section 9.3, Table of Contents	1.0
Stanley Do	5/13/2021	Added to Section 7.2	1.0
David Tang	5/14/2021	Revised Section 5, 6, 7. Updated table of contents.	1.0

# **1. Introduction**

## **1.1 Purpose**

This document will be about the software but will focus more on its collaborative visualization requirements and design. The purpose of CVSST is to explore planetary bodies using actual data from the Jet Propulsion Laboratory (JPL) with multiple users.

## **1.2 Document Conventions**

The font we are using for each section title is Times New Roman with a font size of 20. For the subtitles, we are using a font size of 14 and bullet points for better readability and organization.

## **1.3 Intended Audience and Reading Suggestions**

This document is meant for project managers, developers, users, documentation writers and people with some background in computer science. This includes staff, faculty, advisors, and our NASA JPL liaisons. The recommended reading sequence for this document is to start with the Introduction followed by the specific section for the reader's topic of interest.

## **1.4 System Overview**

The Collaborative Visualization for Solar System Treks (CVSST) provides collaborative markup of 3D solar system terrain such as the creation of waypoints, rapid navigation toward waypoints, text annotations, and freely drawn "ink" annotations on the existing Solar System Treks web portal. CVSST shall also provide networked "rooms" that let the user markup and share different states of 3D solar system terrain simultaneously, communicate via a text communications system, and create waypoints for rapid navigation. Upon release, CVSST shall be open to the public and be used for scientific research, mission planning, educational purposes, and general exploration.

## 2. Design Considerations

### 2.1 Assumptions and Dependencies

- Users are expected to have consistent and stable Internet connections to use this web application.
- Users are expected to be familiar with an Internet browser and be familiar with handling the keyboard and mouse.
- Users may need to install the WebXR Emulator extension which enables users to run WebXR content in desktop browsers without using a real XR device. The extension will emulate the WebXR API on browsers that don't support it.

### 2.2 General Constraints

- **Hardware Limitations** – The user's browser may require a significant amount of graphics rendering, which requires a more powerful GPU. This may lead to performance issues and hurt the user experience.
- **Software Limitations** – The software's markup functionality shall be implemented through the CesiumJS framework, which may create some issues for developers when implementing custom markups if it is not supported.
- **Network Constraints** – The user will require a sufficient Internet connection to have a smooth collaborative experience. Due to the use of WebSockets, high latency may result in performance hiccups in the software.
- **WebXR Limitations** – While VR does provide a more immersive user experience, web-based VR may prove to be an extremely challenging requirement due to its poor browser support and lack of stable/usable technologies.

### 2.3 Goals and Guidelines

Whilst using the Agile Methodology, we are emphasizing usability over readability. Therefore, we may be sacrificing good documentation to produce more prototypes and usable software.

### 2.4 Development Methods

This project uses the Agile Development model. We met weekly with the project advisor and the JPL team via Zoom to exchange information. We have compact and self-organizing teams. We continuously deliver new features and work. In this project, we can adapt based on changes in demand.

### 3. Architectural Strategies

- Programming Languages
  - JavaScript
  - HTML/CSS
- Database
  - Types of information used by various functions.
    - HTTP/ JSON Requests
    - WMTS map data
    - Model Layer Data
    - Mesh data
    - Coordinate data
    - Text data
    - Session Data
  - Frequency of use
    - Data pulled based on every host request of the session.
  - Accessing capabilities
    - Database access based on the current session host and members.
  - Data entities and their relationships
    - Solr
    - ArcGIS
      - Accumulo
    - Oracle
      - Postgres
- Library
  - Dojo Toolkit
  - CesiumJS
  - WebSockets
  - Express
  - Bootstrap
  - Need to choose a database.
- Reuse of existing software components to implement various parts/features of the system.

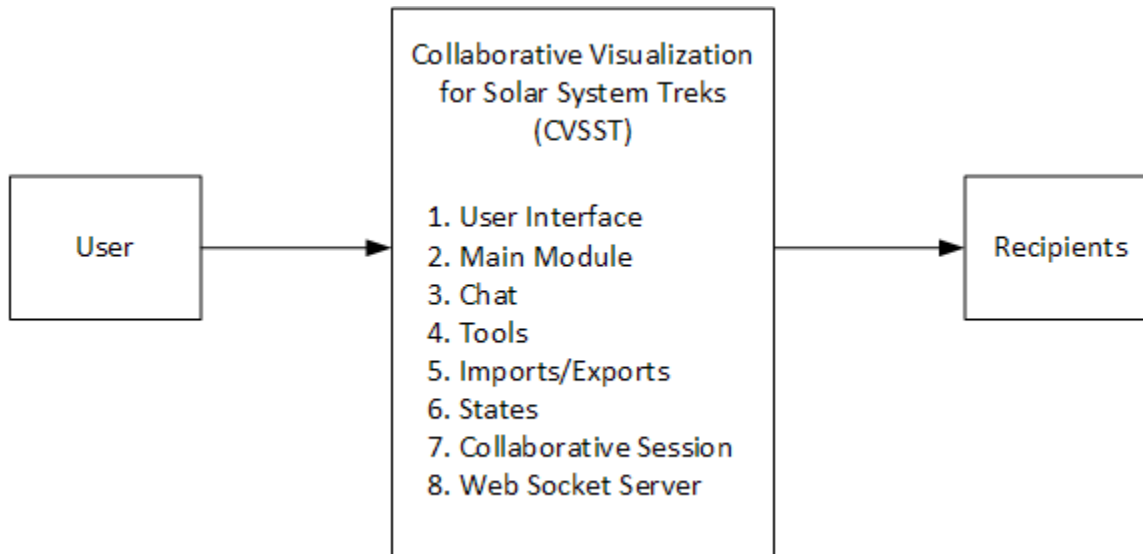


- Reuse of existing software
  - Current Trek system
- Components to implement:
  - Sessions
    - Collaboration in session rooms
    - Communication using a textbox
      - The system shall provide a functionality where users can communicate using text.
    - List of people in the current session
    - Session authentication (Password entry)
      - The system shall allow the host to create a password for the user to use for entry.
      - Host has an option to set a password.
    - Session states
      - The system shall keep track of the current viewed and past viewed layers, along with entities from the current session. (Administrative/Host privilege)
    - The system shall provide a functionality where a user obtains administrative rules and decides what the participants can and cannot do.
      - Users without administrative privileges cannot change the current layer/entity. They can only use tools such as polyline or annotations on layers/entities.
- Future plans for extending or enhancing the software
  - Sessions
    - Create rooms for users to join to collaborate.
- Error detection and recovery
  - Session Capacity
    - The system shall limit the number of users in the current session.
    - If a request is made to a maxed capacity session then the request will not be made.
      - Display an error message to the user.
  - Input Validation
    - The system shall validate session passwords.
      - If incorrect, prompt the user with an error response.
- Memory management policies
  - Data retention requirements
    - Data shall be saved temporarily during each session.

- The system shall allow users to revisit previously used data from the current session.
- External databases and/or data storage management and persistence
  - N/A
- Distributed data or control over a network
  - A separate Node.js server will be used for WebSocket connections.
  - A separate server will run a database connected to the Node.js server.
  - The Node.js server will be run on a testing server.
- Generalized approaches to control
  - SSH access to the Node.js servers.
- Concurrency and synchronization
  - Node.js for the WebSocket server is asynchronous in that it can handle multiple WebSocket connections in parallel.
    - While the WSS connection is active, synchronization between the client and web server shall be achieved with serialization and processing.
- Communication mechanisms
  - WebSockets
  - HTTP Requests
- Management of other resources

## 4. System Architecture

Figure 4-1. Context Diagram, DFD Level 0



At the top-most level, the software must accept input from a user. It will call the modules according to the user's input. The recipients would receive the data sent by the modules. The system was broken down into modules such that each module represented a major feature.

### 4.1.1 The User Interface Module

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

### 4.1.2 The Main Module

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

### 4.1.3 The Chat Module

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

### 4.1.4 The Tools Module

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

### 4.1.5 The Imports/Exports Module

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

#### **4.1.6 The States Module**

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

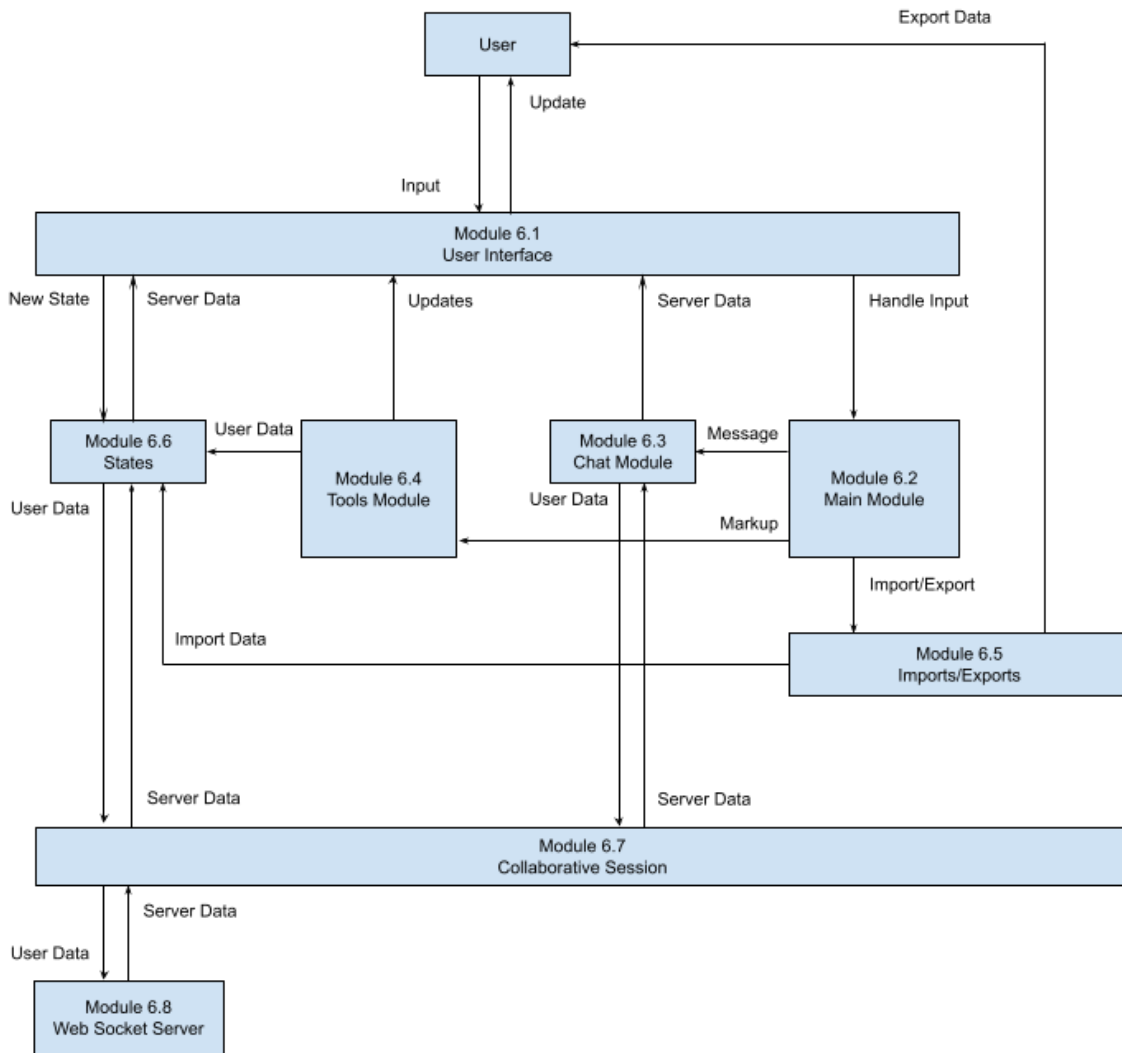
#### **4.1.7 The Collaborative Session Module**

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

#### **4.1.8 The WebSocket Server Module**

Refer to DFD 0 or DFD 1. A detailed description is in Section 6.

Figure 4-2. Level 1 DFD



# 5. Policies and Tactics

## 5.1 Specific products used

- IDE: Visual Studio Code
- Database: We decided against using MongoDB. There could also be a better database and database design than MongoDB. A temporary solution is to have users export data occasionally. A database will be chosen later. Refer to Section 8 for more details.
- Library: Dojo Toolkit, Express, WebSockets, Bootstrap, CesiumJS

## 5.2 Requirements traceability

- Requirements are often discussed in meetings with JPL. After coding and testing, the team goes back to the SRS document to see what requirements have been met. The requirements not met will be brought up with JPL.

## 5.3 Testing the software

### 5.3.1 Testing the features of the modules

- Sessions
  - Testing max capacity of session set by host.
    - Display error messages to users attempting to enter a full session.
  - Test the various platforms the software supports.
    - PC, Mac, mobile
  - Test input textbox
    - Have clients spam textbox and provide invalid input.
      - Discover limitations to be fixed and debugged.
  - Test Administration privileges
    - Ability to control what users can do in a room.
    - Go back and view past layers/entities.
      - Test limitations of memory.
      - Speed and consistency.
    - Ability to remove users from a session.
    - Ability to swap admin roles between a host and user.

### 5.3.2 Engineering trade-offs

- Sessions do not currently support VR/AR capabilities.

### 5.3.3 Coding guidelines and conventions

- Coding Guidelines
  - Code must have uniform indentations for readability.
  - Functions must have a comment at the head describing their purpose.
  - If possible, avoid brute force search algorithms. (Look for more optimal methods)
- Conventions
  - Commented description of the script's purpose at the head of the code.

### 5.3.4 The protocol of one or more subsystems, modules, or subroutines

- Due to this software being built upon existing software, communications between interfaces shall occur through JavaScript code and various implemented modules such as:
  - JSON parsers
  - Database queries
  - API responses
- The software shall require a web browser and use WSS over HTTPS, ensuring fully encrypted communication via SSL. The software shall also query data from JPL's APIs in the form of JSON and/or text/file format. While the WSS connection is active, synchronization between the client and web server shall be achieved with serialization and processing. Although WebSockets allow for a low-latency and full-duplex communication system, data transfer issues may arise when used in large-scale systems. The software system's backend web server shall also require a fast database, which shall assist in the prevention of network bottlenecks. This software shall also include a chatroom service, allowing users to communicate via text in real time.

5.3.5 The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality

- N/A

5.3.6 Plans for maintaining the software

- The system shall use libraries compatible with the existing SST system. This is done to ensure a smooth transition from the testing server to JPL's server in final transition deployment.

5.3.7 Interfaces for end-users, software, hardware, and communications

- Some software interfaces include:
  - **Dojo Toolkit, <https://dojotoolkit.org/>**
    - Used for the frontend code and user interface.
    - Dojo Toolkit does not support ECMAScript 6 and higher, so functional notation cannot be used.
  - **Esri ArcGIS API for Javascript Version 3.X, <https://developers.arcgis.com/javascript/3/jssamples/>**
    - Used for 2D visualization.
  - **CesiumJS Framework projections, <https://cesium.com/cesiumjs/>**
    - Used for creating interactive web applications for sharing dynamic geospatial data.
    - Used for 3D visualization.
  - **WebGL, latest version found at, <https://www.khronos.org/webgl/>**
    - Used for rendering advanced inter-converting 3D and 2D graphics in any compatible web browser without using plug-ins.
  - **WebSockets, latest version found at <https://socket.io/>**
    - Used for transferring data from client to server in real-time and with low latency.

- **Existing APIs in use in the existing Trek application.**
  - Used for transferring data from client to server in real-time and with low latency.

### 5.3.8 Hierarchical organization of the source code into its physical components (files and directories).

- Most of the code is under src/jpl/dijit/
- There is no need to look at the other directories since they are mostly library files used to create a build.
- Src/jpl/dijit/
  - /css - This is the collection of individual styling sheets for each module.
  - /images - Any images you want to include locally can be included here.
  - /templates - These are all the HTML files for the different modules.
  - /ui - These are more JS files that have to do more with user input interaction.
  - /events - For events that transcend several modules their event strings can be found here.
  - /plugins - These are files that are from a third party library but had to be modified to work with Trek.
  - /utils - These are files that don't exactly create a module but provide some sort of logic, for example label formatting or WKT conversions.
- Src/jp/dijit/ControlBar.js - This module contains all the overlay buttons seen on the Trek interface and their event handlers.

### 5.3.9 How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

#### 1. Steps

##### 1.1. Installing Node.js and Npm

- 1.1.1. Follow the directions in this link <https://www.npmjs.com/get-npm> to install. Make sure that you install the version 14.15.4. Installing the LTS version of Node will also install npm.
- 1.1.2. Run the MSI and go through the installation with default settings.
- 1.1.3. Test if Node is successfully installed by opening your terminal (command prompt) and typing "Node". It should say something along the lines of "Welcome to Node.js v14.15.4". Exit the Node terminal by pressing CTRL+C twice or by typing ".exit".
- 1.1.4. After exiting out of the node terminal, type "npm version" in the command prompt. It should display all of the npm packages you have installed along with their versions.

##### 1.2. Cloning the CVSST repository using Git

- 1.2.1. Create a new folder in a directory you would like to store your application and name it "cvsst-test-1.0" .
- 1.2.2. Using Git Bash, navigate to the directory of your newly created folder and enter the command (without quotation marks):

"git clone <https://github.com/stanleydo/cvsst-test-1.0.git>".



- 1.2.3. Using your command prompt, navigate to the directory of your cloned application and enter the command, “npm install”.
  - 1.2.4. After entering “npm install”, type “npm start”. This will launch the application on port 4000, which is the application’s default number.
  - 1.3. Installing Moseif CORS Changer
    - 1.3.1. To use the CVSST application, a Chrome extension called Moseif CORS changer must be installed. Navigate to the Chrome Extension Store and install the Moseif Origin & CORS Changer extension.
    - 1.3.2. Turn on the extension when you are in development. *Warning: Do not visit any sketchy website, as this extension opens up many security risks such as cross site requests. Turn off when finished.*
  - 1.4. Accessing CVSST
    - 1.4.1. Once the application is launched using “npm start”, you can test it out on your browser by navigating to “<https://localhost:4000>”. Note that it is “https” not “http”.
    - 1.4.2. Go to “<https://localhost:4000/src/>” to run it.
- 5.3.10 Tactics such as abstracting out a generic DatabaseInterface class
- Database structure was not set up. Refer to Section 8 for details.

# 6. Detailed System Design

## 6.1 User Interface Module

### 6.1.1 Responsibilities

The User Interface Module serves as the messenger between the user and the Main Module. It provides a graphical user interface (GUI) for the user and allows the user to interact with the entire system.

### 6.1.2 Constraints

Some constraints may include a limitation of space for additional widgets and/or possible obstruction of views from other elements in the application. There may also be a very high number of entities which may require additional graphical processing power.

### 6.1.3 Composition

The User Interface Module shall be composed of various widget-like components following the Dojo Toolkit standards. Each User Interface Module shall consist of simple HTML and CSS elements, all of which can be accessed by their corresponding lower-level modules.

The components are listed below.

#### 1. Chat Component

- a. The Chat Component shall consist of a main chat box for containing all chat messages.
- b. The Chat Component shall consist of a text input box for sending messages.
- c. The Chat Component shall consist of a single button used to send messages.

#### 2. Tool Component

- a. The Tools Component shall consist of a dropdown bar consisting of all the collaborative states in the session.
- b. The Tools Component shall consist of buttons indicating a variety of tools: free-hand “ink” drawings, polylines drawings, text, circles, squares, and triangles.
- c. The Tools Component shall reveal additional attributes for each button allowing for additional modifications on the markup. Some attributes include drawing sizes, font size, color, and width of lines.

#### 3. Import/Exports Component

- a. The Import/Exports Component shall consist of an import button and an export button.

#### **4. States Component**

- a. The States Component shall consist of all of the markups and entities that have been placed on the 2D and 3D terrain.

#### **5. Collaborative Sessions Component**

- a. The Collaborative Sessions Component shall consist of a single button which reveals a modal.
- b. The modal shall have two buttons: joining and creating a collaborative session.

#### **6.1.4 Uses/Interactions**

The user will input data into the system via click events, scroll events, and key press events.

#### **6.1.5 Resources**

This module requires a miniscule amount of memory within the user's browser and a sufficient GPU to render additional entities or markups.

## **6.2 Main Control Module**

### **6.2.1 Responsibilities**

The Main Control Module serves as the main component where all of the CVSST modules are instantiated and initialized. It also handles all user input and directs the data to the correct modules for additional processing.

### **6.2.2 Constraints**

There have been no constraints identified at this time.

### **6.2.3 Composition**

The Main Control Module shall consist of functions which initialize modules 6.3, 6.4, 6.5, 6.6, and 6.7.

### **6.2.4 Uses/Interactions**

The Main Control Module will handle events from the UI. Based on the commands, it will interact with the Chat Module, Tools Module, Imports/Exports Module, or States Module.

### **6.2.5 Resources**

This module requires a miniscule amount of memory within the user's browser.

## **6.3 Chat Module**

### **6.3.1 Responsibilities**

The Chat Module shall obtain data directly from the user interface and update the user interface with new data from module 6.7. This module shall handle all chat events from the UI, update the chat box with data from the session, and send new data to the session, which will go to the server.

### **6.3.2 Constraints**

The constraints are character limits and number of messages that can exist in a collaborative session at once. Depending on the amount of data being sent, network latency and memory usage may increase. A maximum of 500 characters per message shall suffice.

### **6.3.3 Composition**

The Chat Module shall consist of functions which send data to the collaborative session. It shall also contain functions which handle new data that is sent from the session, functions that update the UI, and functions that obtain data from the UI.

### **6.3.4 Uses/Interactions**

When there is a message being sent via text, the Main Control Module will handle the event and call functions within the Chat Module which retrieve user input from the UI. Once the input is validated, the data is sent to module 6.7 for processing then to module 6.8 for storage into the database. The WebSocket server will recognize the data, store it in the database, and send the new entry to all users in the same collaborative session. When the WebSocket Server Module receives this new entry, the data will be sent back to the Chat Module to update the UI accordingly.

### **6.3.5 Resources**

This module requires a chat box which will contain all chat messages, an input text form for user chat messages, and a send button to send the messages.

## **6.4 Tools Module**

### **6.4.1 Responsibilities**

The Tools Module shall provide all of the collaborative functionalities for markups and drawings on 2D and 3D terrain. This module shall obtain user input from the UI, update the tools component in the UI, and send all markup data to module 6.6 for processing. This module should also keep track of the tools currently selected.

### **6.4.2 Constraints**

The user will be limited to the number of tools they can use at once. There will also be limits to attributes within the markup data for size, colors, line types, and font sizes.

### **6.4.3 Composition**

This module shall consist of multiple subcomponents for each type of markup.

### **6.4.4 Uses/Interactions**

The user will select which tool they would like to mark up the 2D or 3D Solar System terrain with, then click on a location on the terrain to place their drawing. The tools component keeps track of the tool, the tools' user-defined attributes, and the location of the desired markup. The Tools Module then sends this data to module 6.6 to be rendered in the UI and sent to module 6.7 to be synchronized with other users.

### **6.4.5 Resources**

This module will require the use of existing SST functions such as polylines, entity placement, and Fly To functions.

## **6.5 Imports/Exports Module**

### **6.5.1 Responsibilities**

The Imports/Exports Module will handle all configuration imports and exports for CVSST states. This module shall allow users to export configurations in text format to be imported for future use. This module shall obtain data from module 6.7 and send imported data to module 6.6.

### **6.5.2 Constraints**

There may be a limit as to how much data can be stored within the configuration file. Larger files may take longer amounts of time to process, which may be taxing for the system.

### **6.5.3 Composition**

The Imports/Exports Module will consist of functions that will parse a configuration file and pass it to module 6.6 for rendering onto the UI.

### **6.5.4 Uses/Interactions**

The user can export and download a configuration file which saves all the current data of a session or state.

### **6.5.5 Resources**

The Imports/Exports Module will read data from the browser's memory to obtain configurations.

## **6.6 States Module**

### **6.6.1 Responsibilities**

It allows the user to save their session. The changes made during a session can be saved. The user can go back to a previous session by choosing a state. Refer to the Software Requirements Specification document for the definition of state.

### **6.6.2 Constraints**

The number of states that can be kept. The time it takes to go back to a state.

### **6.6.3 Composition**

N/A

### **6.6.4 Uses/Interactions**

It is used by the Main Control Module. The States Module uses the WebSocket Server Module. This can affect the state data stored in the database.

### **6.6.5 Resources**

Utilizes the WebSocket server's memory and browser memory.

## **6.7 Collaborative Session Module**

### **6.7.1 Responsibilities**

The Collaborative Session Module serves as the main source of data transfer between the client and public WebSocket server. It is responsible for sending and receiving data back-and-forth to the WebSocket server. It also transmits data between the Chat Module, Tools Module, States Module, and Import/Exports Module.

### **6.7.2 Constraints**

Large amounts of data that is mapped to a session may take a significant amount of time to load depending on network speed. For a smooth and seamless experience, users are recommended to have a good Internet connection. High latency may result in bottlenecks in the system.

### **6.7.3 Composition**

The Collaborative Session Module will contain methods which send and receive data back-and-forth to the WebSocket Server Module.

#### **6.7.4 Uses/Interactions**

When new data is input by the user, the data is transferred to the Collaborative Session Module and sent to the server. Once the server stores the new data within the database, the server sends back the new data to all other users. The WebSocket Server Module will receive this new data and send it to the correct modules to update the UI.

#### **6.7.5 Resources**

The WebSocket Server and database are needed. A race condition can be deciding which modules' data should the WebSocket Server Module accept first. It can be resolved by allowing one module to finish sending its data before accepting data from another module.

## **6.8 WebSocket Server Module**

### **6.8.1 Responsibilities**

The WebSocket Server Module shall store any data from the client into the database. It is in charge of all the data coming to and from the clients. It keeps track of all collaborative sessions, validates data, and controls all read and write functions into the database.

### **6.8.2 Constraints**

The WebSocket Server Module may have a limit as to how many sessions can exist at a single time. Whilst handling these sessions, there may be performance bottlenecks if the server is not able to handle requests as fast as they are made.

### **6.8.3 Composition**

The WebSocket Server Module will consist of similar subcomponents which handle data from the chat, states, and imports/exports.

#### **1. The Chat Sub-module**

- a. The Chat Sub-module is in charge of processing and storing all data from the client. This module shall read and write data only from the chat collections in the database.

#### **2. The State Sub-module**

- a. The State Sub-module is in charge of processing and storing all state data from the client. This involves any entities stored in the states and layers being used by the user.

### **6.8.4 Uses/Interactions**

The WebSocket Server Module shall receive data in the form of JSON or text from the client WebSocket, then process the data so that it can be written to or read from the database.

#### **6.8.5 Resources**

The WebSocket Server Module requires a stable Internet connection, a database that can be connected locally for storing data, and a sufficient amount of computational power.



# 7. Detailed Lower level Component Design

## 7.1 CollaborativeSession.js

### 7.1.1 Classification

This file is a singleton module that is accessible anywhere in the code. It contains references to the control bar, WebSocket Module, and States Module. The WebSocket instance within the Collaborative Session Module is referenced by the Chat Module, States Module, and Tools Module to store and transfer data between clients over a network.

### 7.1.2 Processing Detail

The module is instantiated in ControlBar.js as a singleton.

### 7.1.3.2 Restrictions/Limitations

Since the module is instantiated with ControlBar.js, other classes which use CollaborativeSession.js may not reference the Collaborative Session until it has been instantiated. Modules which load before ControlBar.js will reference a null value if attempting to access the Collaborative Session Module before it is loaded.

### 7.1.3.3 Performance Issues

There are no performance issues with this module.

### 7.1.3.4 Design Constraints

Since this module is a singleton, it is not possible to have multiple WebSocket connections in one instance of the application. This may be useful in the future when we want to visit multiple rooms using a single tab.

### 7.1.3.5 Processing Detail For Each Operation

The collaborative session serves as both a client and a listener when using WebSockets. It handles all emits being sent to the server and all messages received from the server.

## 7.2 Chat Module

### 7.2.1 Classification

An instance of the Chat Module class shall be instantiated in the ControlBar.js file which exists in the existing Trek code. The Chat Module uses the collaborative session instance to emit data to the server.

### 7.2.2 Interface Description

There is no interface for the Chat Module.

### 7.2.4 Processing Detail

Before a chat message is sent, it is checked for leading white spaces or empty strings. If there is no actual message, nothing will happen at all.

#### **7.2.4.1 Restrictions/Limitations**

Users are not able to send images or hyperlink any kind of text in the chat. Users are also not allowed to have the same name. The size of the chat box is also fixed, so some users may have various sized text boxes and font sizes.

#### **7.2.4.2 Performance Issues**

It is possible for a user to send an enormous amount of text data which can cause performance issues when attempting to render all of the text.

#### **7.2.4.3 Design Constraints**

There are no design constraints for the Chat Module.

## 8. Database Design

The database design was more complex than we previously thought because MongoDB structure differs greatly from relational SQL. Documents can hold a maximum of 16 MB of data, which may become a major issue if not handled properly. After discussing the issues of using MongoDB with the JPL team, we realized there could be a better usage of the database. While a NoSQL database is fast and clean, it seems that our use of a relation database would better fit our needs. A temporary solution is to have users export their data occasionally, allowing them to save their changes locally.

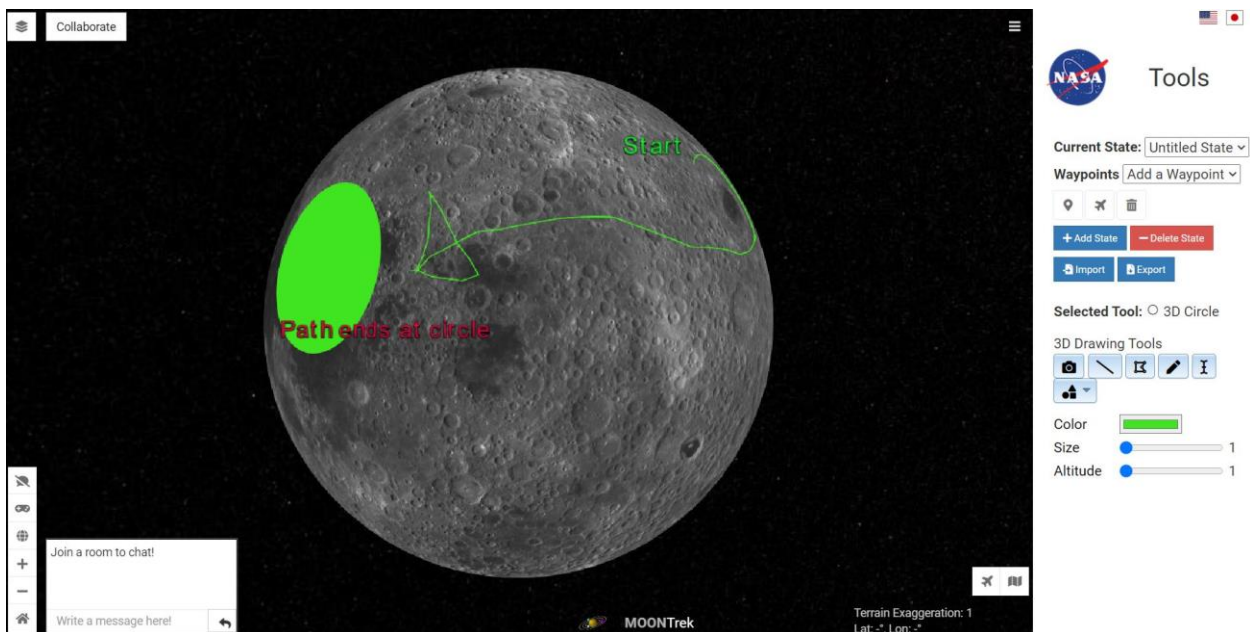
# 9. User Interface

## 9.1 Overview of User Interface

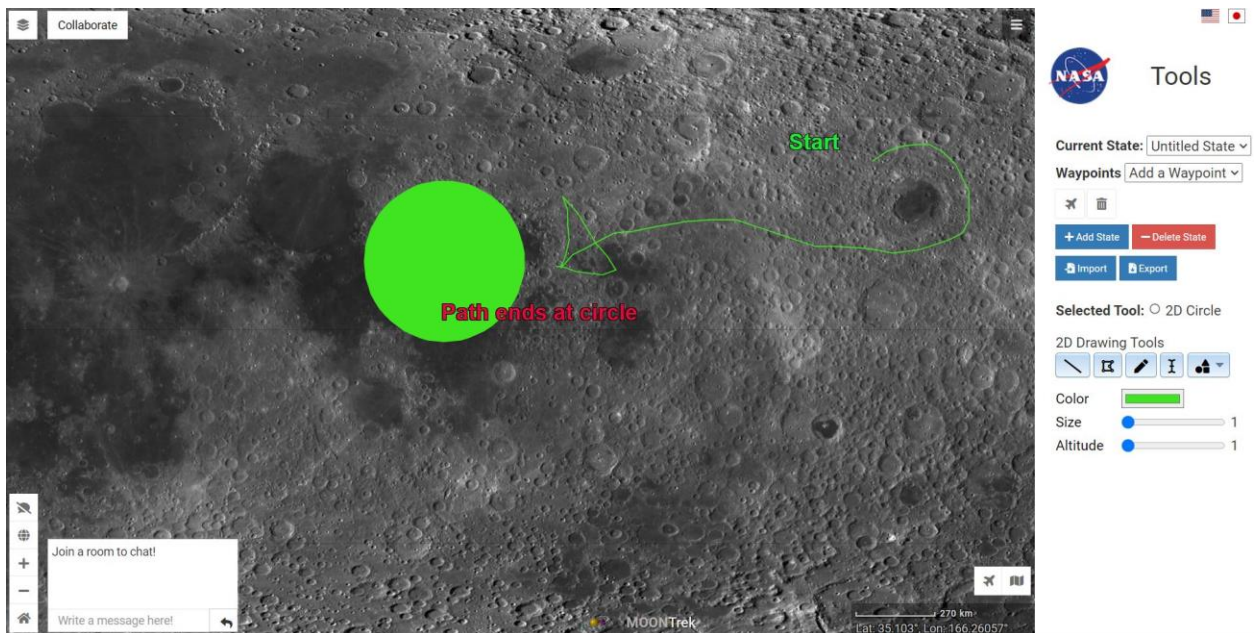
- The system shall provide intuitive input systems for users.
  - Some input systems may include simplified icons for buttons, sliders, and easy contrasting colors.
- The system shall provide a functionality where users can use text communication.
- The system shall provide a functionality where users can share SST together.
- The system shall provide a functionality where users can invite a person or group of people to a room session.
- The system shall provide a functionality where users can share changes that one user made on the planet with another user.
- The system shall follow the color scheme, design, and theme of the underlying SST software.
- The system shall provide a functionality where users can save a session state of the planet.
- The system shall provide a functionality where users can annotate on the planet.
  - The user shall be able to draw a path from one point to another.
  - The user shall be able to draw a circle, triangle or square onto the planet.
  - The user shall be able to modify the objects by changing their color or size of the object.
- The system shall provide a functionality where a user obtains administrative rules and decides what the participants can and cannot do.
  - The host shall have the ability to lock the screen or unlock the screen for the participants.
- The system shall provide a functionality where users can search an area of the planet and navigate to that area.
- The system shall provide a functionality where users can put a waypoint onto the planet.

## 9.2 Screen Frameworks or Images

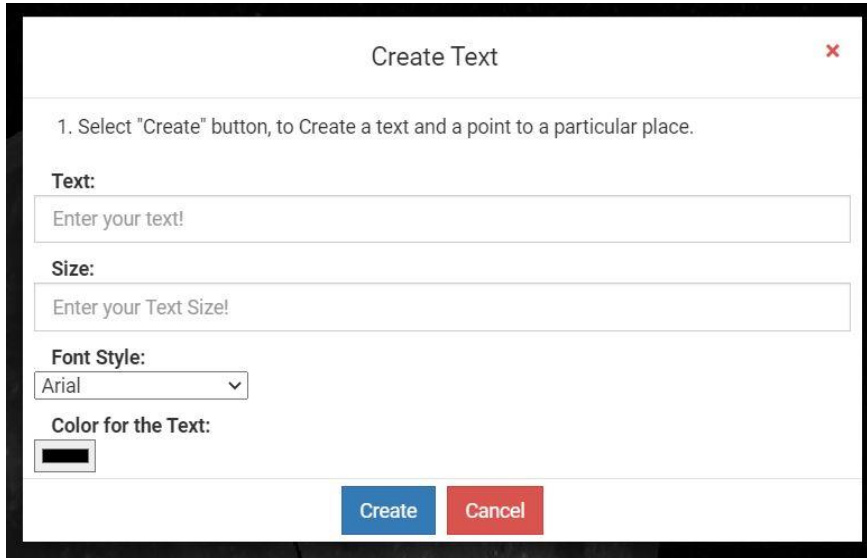
The screenshot below is the user interface for viewing 3D terrain. It interacts with all modules.



The screenshot below is the user interface for viewing 2D terrain. It interacts with all modules.



The screenshot below shows the input dialog to create text that will be placed on the 2D and 3D terrain. It interacts with the Tools Module.



Create Text

1. Select "Create" button, to Create a text and a point to a particular place.

**Text:**  
Enter your text!

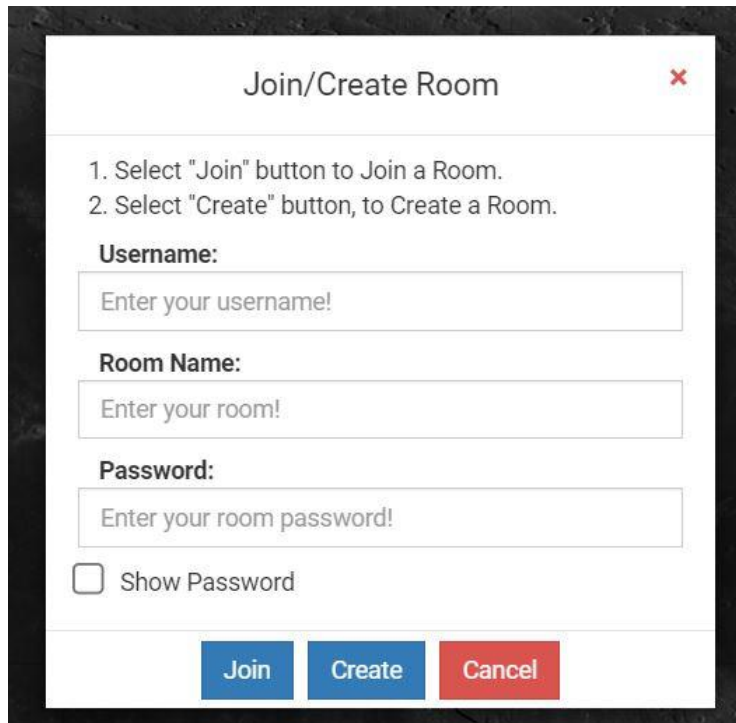
**Size:**  
Enter your Text Size!

**Font Style:**  
Arial

**Color for the Text:**  
[Color Swatch]

Create Cancel

The screenshot below shows the input dialog to join/create a room. It interacts with the Collaborative Session Module.



Join/Create Room

1. Select "Join" button to Join a Room.  
2. Select "Create" button, to Create a Room.

**Username:**  
Enter your username!

**Room Name:**  
Enter your room!

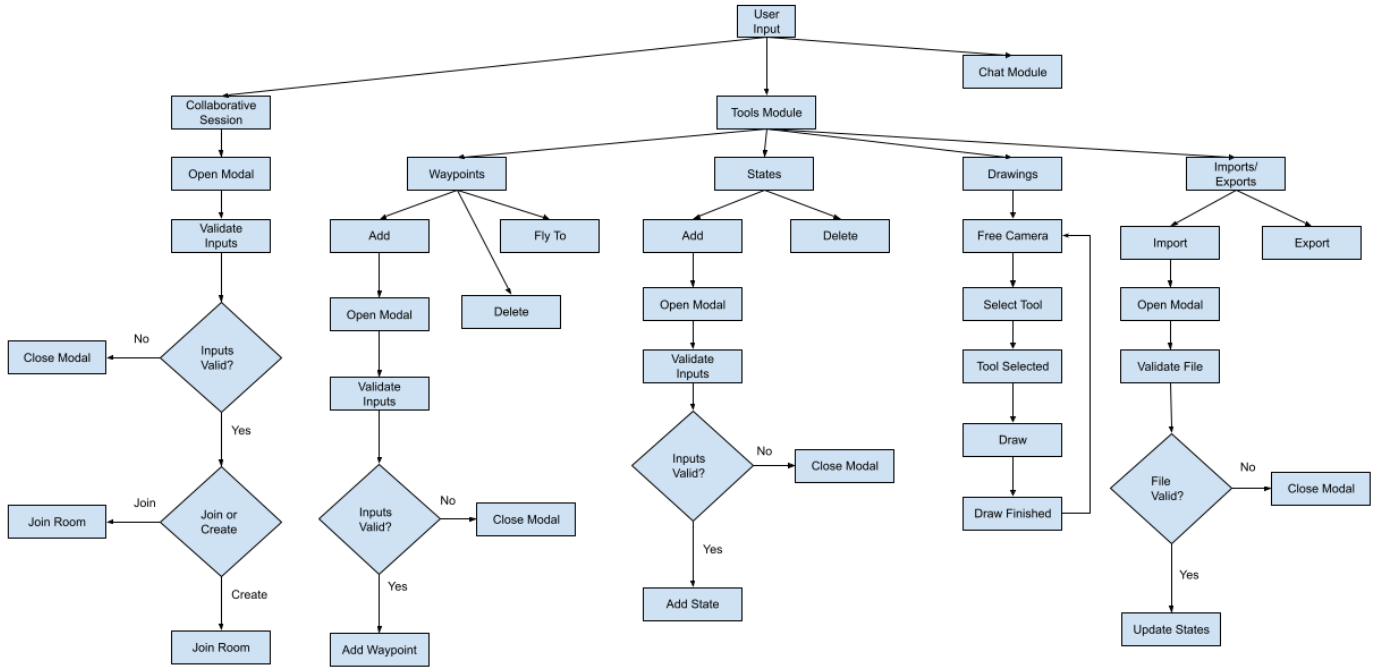
**Password:**  
Enter your room password!

Show Password

Join Create Cancel

### 9.3 User Interface Flow Model

The User Interface Flow Model below shows that the modules will be called according to the provided user input.



# 10. Requirements Validation and Verification

The following are functional requirements found in Section 4 of the SRS document.

Requirements	Modules.UI/Components and Testing Methods
<p>The system shall allow users to host a “collaborative session” (See Section 1.4 of the SRS).</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall reflect all visuals (See Section 1.4 of the SRS) to other users in a session as they are placed.</li> <li><b>ii.</b> The system shall synchronize all visuals between users in a session unless configured otherwise.</li> <li><b>iii.</b> The system shall share the same 2D/3D solar system terrain between users in a session.</li> <li><b>iv.</b> The system shall display the same 2D/3D solar system terrain between users in a session.</li> <li><b>v.</b> The system shall provide a text communications system for users in a session.</li> <li><b>vi.</b> The system shall provide a voice-based communications system for users in a session.</li> <li><b>vii.</b> The system shall allow users to invite other users to their session.</li> <li><b>viii.</b> The system shall allow users to see a list of all users in the session.</li> </ul>	<ul style="list-style-type: none"> <li>● User Interface Module</li> <li>● Chat Module</li> <li>● Tools Module</li> <li>● Collaborative Session Module</li> <li>● WebSocket Server Module</li> <li>● Testing: <ul style="list-style-type: none"> <li>○ User created a room (session) with name and password. Other users were invited to join. They inputted the room’s name and password and successfully joined.</li> <li>○ Everyone was able to see the same drawings (visuals) made on both 2D and 3D solar system terrain in real-time.</li> <li>○ There was a textbox where users typed a message in the chat. Everyone was able to send and receive messages from the chat.</li> </ul> </li> <li>● Not implemented: <ul style="list-style-type: none"> <li>○ Voice-based communications system</li> <li>○ List of all users in the session</li> </ul> </li> </ul>
<p>The system shall provide tools for markup onto 2D/3D Solar System terrain in varying sizes and colors.</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall provide an “ink-based” annotation tool for free-hand drawing.</li> </ul>	<ul style="list-style-type: none"> <li>● User Interface Module</li> <li>● Tools Module</li> <li>● Testing: <ul style="list-style-type: none"> <li>○ Users were able to do free-hand drawing on 2D and 3D terrain.</li> <li>○ Users were able to place squares, circles, and triangles</li> </ul> </li> </ul>



<ul style="list-style-type: none"> <li><b>ii.</b> The system shall allow users to draw simple 2D shapes (squares, circles, triangles, etc.) on the 2D/3D terrain.</li> <li><b>iii.</b> The system shall allow users to project 2D text onto the 2D/3D terrain.</li> <li><b>iv.</b> The system shall allow users to create polyline (See Section 1.4 of the SRS) annotations.</li> </ul>	<ul style="list-style-type: none"> <li>on 2D and 3D terrain.</li> <li>○ Users were able to put text on 2D and 3D terrain.</li> <li>○ Users were able to create polylines on 2D and 3D terrain.</li> </ul>
<p>The system shall allow users to create interactive navigation waypoints on 2D/3D Solar System terrain.</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall allow users to move the navigation waypoints.</li> <li><b>ii.</b> The system shall allow users to modify the navigation waypoints.</li> </ul>	<ul style="list-style-type: none"> <li>● User Interface Module</li> <li>● Tools Module</li> <li>● Testing: <ul style="list-style-type: none"> <li>○ Users were able to place waypoints only on 3D terrain.</li> <li>○ Users were able to use Fly To function to go to the waypoints on 2D and 3D terrain.</li> <li>○ Users can delete waypoints.</li> </ul> </li> <li>● Not implemented: <ul style="list-style-type: none"> <li>○ Placing waypoints on 2D terrain.</li> <li>○ Moving and modification of waypoints on 2D and 3D terrain.</li> </ul> </li> </ul>
<p>The system shall provide administrative powers to a session's host.</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall allow the host user to have control of all other users' displays in a session.</li> <li><b>ii.</b> The system shall allow the host user to transfer host privileges to other users' displays in a session.</li> <li><b>iii.</b> The system shall allow the host user to prevent collaborative markups from specific users in a session.</li> <li><b>iv.</b> The system shall allow the host user to grant other users permissions to all collaborative tools.</li> <li><b>v.</b> The host shall have the ability to disable use of specific tools by others.</li> </ul>	<ul style="list-style-type: none"> <li>● Tools Module</li> <li>● Collaborative Session Module</li> <li>● WebSocket Server Module</li> <li>● Administrative powers for the host were not implemented.</li> </ul>

<p><b>vi.</b> The host shall have the ability to give permission to the user to use specific tools.</p>	
<p>The system shall allow users to create multiple “SST states” (See Section 1.4 of the SRS) within a session.</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall allow users to access different states within a session.</li> <li><b>ii.</b> The system shall reflect all changes made within each SST state.</li> </ul>	<ul style="list-style-type: none"> <li>● User Interface Module</li> <li>● States Module</li> <li>● Collaborative Session Module</li> <li>● WebSocket Server Module</li> <li>● Testing: <ul style="list-style-type: none"> <li>○ Users were able to create multiple states. They can choose a state in the dropdown menu.</li> </ul> </li> <li>● Not implemented: <ul style="list-style-type: none"> <li>○ Waypoints are not saved with states.</li> <li>○ Waypoints do not work in a room (session).</li> <li>○ Track/lock camera position and store position to be loaded later.</li> </ul> </li> </ul>
<p>The system shall allow users to save their current “SST state”.</p> <ul style="list-style-type: none"> <li><b>i.</b> The system shall allow the user to extract all markup to be saved for later use.</li> <li><b>ii.</b> The system shall allow the user to extract layer data to be saved for later use.</li> <li><b>iii.</b> The system shall allow the user to import saved markup to be visualized onto SST.</li> <li><b>iv.</b> The system shall allow the user to import saved layer data to be visualized onto SST.</li> </ul>	<ul style="list-style-type: none"> <li>● User Interface Module</li> <li>● Imports/Exports Module</li> <li>● States Module</li> <li>● WebSocket Server Module</li> <li>● Testing: <ul style="list-style-type: none"> <li>○ Users were able to export a state into a file.</li> <li>○ Users were able to import a state by uploading a file.</li> <li>○ The imported state showed markup.</li> </ul> </li> <li>● Not implemented: <ul style="list-style-type: none"> <li>○ Waypoints are not imported/exported with states.</li> <li>○ Extract and import layer data (much more complex than initially thought).</li> <li>○ Track/lock camera position and store position to be loaded later.</li> </ul> </li> </ul>

# 11. Glossary

**SDD:**

Software Design Document

**SRS:**

Software Requirements Specifications

**UI:**

Abbreviation for User Interface. The UI is also known as the frontend of the application and is the part of the system that the user interacts with to complete tasks.

**HTTP:**

Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems

**HTML:**

Hypertext Markup Language is the standard markup language for creating web pages.

**Application program interface (API):**

Functions or methods for accessing software services or libraries.

**Augmented Reality (AR):**

A technique in computer graphics that superimposes (places) a computer-generated object into a device's camera view to alter the perception of the real world.

**Virtual Reality (VR):**

Virtual reality is a simulated experience that can be similar to or completely different from the real world.

**Operating System (OS):**

The software that allows any computer to communicate, modify, and terminate any hardware and software communications based on the end-user's decisions.

**JSON:**

Abbreviation for JavaScript Object Notation. Objects that are often used to return data from APIs back to the front

**CVSST:**

Collaborative Visualization of Solar System Treks

**JavaScript:**

A programming language that is heavily used for web applications

**WMTS:**

A Web Map Tile Service is a standard protocol for serving pre-rendered or run-time computed georeferenced map tiles over the Internet.

**CesiumJS:**

CesiumJS is an open-source JavaScript library for world-class 3D mapping

## 12. References

**CSULA CS Department & Senior Design Faculty**, SRSTemplateSeniorDesign-v2, Fall 2020.  
SRS template provided by the CS department for this project.

**Collaborative Visualization of Solar System Treks Requirements Document**, 2021-5-14-  
CVSST-SRS, Spring 2021.

<https://csns.calstatela.edu/download?fileId=7837680>

Alternative link: <https://csns.cysun.org/department/cs/project/view?id=7808905>

**Collaborative Visualization of Solar System Treks Personas Document**,  
CVSST\_Personas\_Document, Fall 2020.

<https://csns.calstatela.edu/department/cs/project/resource/view?projectId=7808905&resourceId=7809369>

Alternative link: <https://csns.cysun.org/department/cs/project/view?id=7808905>

**Telescope Moon Trek Senior Design Team 2019**, SRS\_Fall\_19, Fall 2019  
SRS document written by another team from Senior Design Fall 2019.