CMPT 276 D100 - Milestone 1 Report

Relevant Links:

AI Disclosures on Github
Google Docs

Members:

Front-End:
Anton (Anton-Pan)
Sadab (SadabKhan01)
Back-End:
Bidisha (bidishaaroy)
Paul (pppaaaaul)

Milestone Requirements & Assignments:

[Anton] Part 1: API Selection and Usage (describe 2 approved APIs w/ purpose and usage)
[Anton] Part 2: Feature Descriptions (3 per API w/ user benefits, relevance and changes from proposal)
[Sadab] Part 3: Mid-Fidelity Prototype (interactive UI prototype w/ user flow and detailed interface)
[Sadab - Agile/Scrum] Part 4: SDLC Model Selection (justify choice w/ reasoning according to project goals)
[Bidisha] Part 5: Work Breakdown Structure (prioritized GH Issues, logical dependencies, and task importance)
[Anton] Part 6: Project Schedule and Milestones (timeline w/ milestone/deadlines for development/testing)
[Bidisha] Part 7: Risk Assessment (5 low, 5 medium, 5 high risks w/ mitigation strategies)
[Paul] Part 8: Data Flow Diagrams (DFDs showing data flow between APIs and app components)
[Paul] Part 9: MVC Model (high-level MVC structure showing separation of Model, View, and Controller)

Parts 1 & 2: API Selection, Usage and Features (Anton)

Note: The content in this section is partially copy-pasted from the M0 API section. Additional text has been added to comply with the additional M1 requirements, but no changes were requested by the T/A.

API #1: Launch Library II (https://thespacedevs.com/llapi)

LLII provides a semi-static database of all past, present and future space-related events, places and people, such as rocket launches, spacewalks, launch locations, and astronaut profiles.
This API will be used for creating an interactive/visual mapping of time-sorted space launches (*Feature #1*) on the 3D globe, with detailed information about them (*Feature #2*) available on-click. Space-based events (*Feature #3*) will appear on a collapsible panel on the side of the screen

Feature #1

A mapping of all orbital launches, with the ability to sort by time periods (eg: today, next month, last week, etc.). This feature will use the "launches" endpoint.

Feature #2

Information about those launches, including launch location, agency, crew members and spacecraft configuration. This feature will use, and aggregate data from, the "locations", "agencies", "astronauts" and "spacecraft" endpoints.

Feature #3

An "external events" panel/feed that shows unique and/or important non-launch events such as spacewalks and engine tests. This feature will use the "events" endpoint.

API #2: Celestrak (https://celestrak.org/)

Celestrak provides a TLE (two-line element) set for each celestial body orbiting the Earth, which provides position and trajectory data that can be used to calculate recent past, current, and near future positions. This API will be used to simulate satellite positions (*Feature #1*) and past/future movement (*Feature #2*), with the ability to focus on one single satellite (*Feature #3*).
This API will be used in conjunction with satellite-js as a helper library to parse and calculate satellite positions using SGP4.

Feature #1

A mapping of the current positions of the 100 brightest satellites, updating on some time interval to visibly show their movements across the globe. This feature will use this unnamed endpoint.

Feature #2

A search bar, where the user can search for a specific satellite, and have it centered on the globe and auto-selected for the position-over-time feature. This feature will use this unnamed endpoint.

Feature #3

Selecting a single satellite will show the user a slider, which they can use to drag the satellite's estimated position back and forth in time over a period of 24 hours (as the TLEs are only reasonably accurate to that amount). This feature will use this unnamed endpoint.

Note: Features 2 and 3 have swapped positions compared to Milestone 0.

Helper APIs:

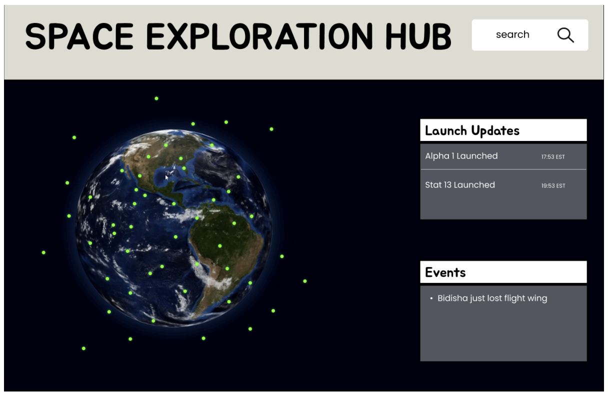
Satellite-JS (https://github.com/shashwatak/satellite-js)

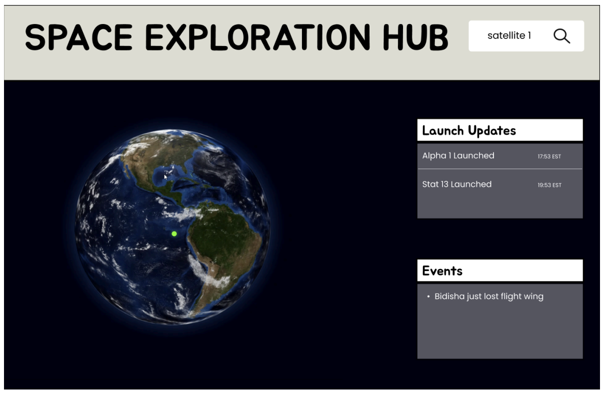
This library will be used with API #2 to convert TLEs to geodetic data so that satellite positions can be displayed on the 3D globe.

GlobeGL (https://github.com/vasturiano/globe.gl)

This is a wrapper library for the three-globe plugin for three.js. It allows for creation of a 3D globe, with features such as points, particles and rings, which will be used to represent launch events, satellites and their paths.

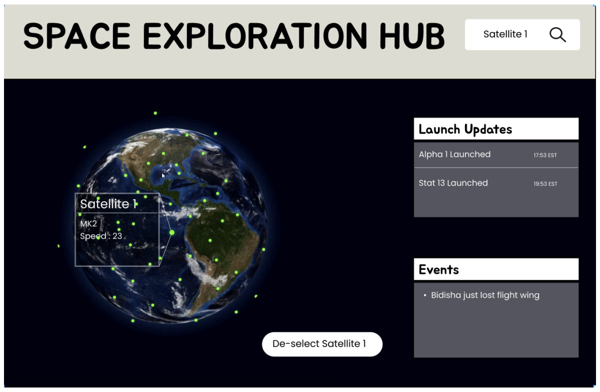
Part 3: Mid-Fidelity Prototype (Sadab)

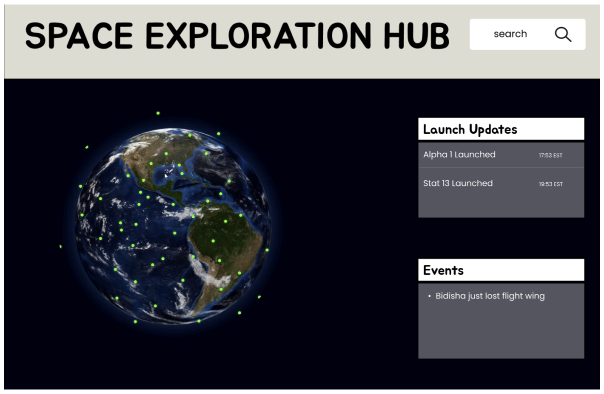

SPACE EXPLORATION HUB
search
Launch Updates
Alpha 1 Launched 17:53 EST
Stat 13 Launched 19:53 EST
Events
• Bidisha just lost flight wing


SPACE EXPLORATION HUB
satellite 1
Launch Updates
Alpha 1 Launched 17:53 EST
Stat 13 Launched 19:53 EST
Events
• Bidisha just lost flight wing

1) Search for a specific satellite.

2) It shows you the specific satellite.




SPACE EXPLORATION HUB
search
Launch Updates
Alpha 1 Launched 17:53 EST
Stat 13 Launched 19:53 EST
Events
• Bidisha just lost flight wing

3) You can deselect the satellite.

4) Takes you back to showing all satellites again.

Figma Link

Part 4: SDLC Model Selection (Sadab)

We are choosing Agile Scrum because it allows us to build and learn in small, manageable chunks. Instead of trying to plan every detail up front, we first create a mid-fidelity prototype, then start the real-time API integrations one by one. That way, if the data format from Launch Library II changes or we discover a more effective interaction for our globe controls, we can adapt without disrupting our entire schedule.

Every two weeks, we will do a sprint that ends with something you can click on like a React increment that lights up a bit more of our vision, our Scrum master is Anton as he is the project manager and manages and coaches us. We are going to analyze each of the possible user scenarios and change websites UI and functionality, whether mapping orbital launches, showing spacewalks in the events, or centring a satellite when you hit the Search button. Then we will be doing a meeting once every week which will enable us to go over what we did that week and what we are going to do next week, and later we are going to discuss if any blockers came and if we resolved it or needs to be resolved.

At the end of each sprint, we will tell our progress to the TA(Vincent) who can act as the product owner and we can gather feedback, and bring changes to our plan for the next two weeks. We believe this cycle of build, review and improve will keep the project moving forward smoothly, help us catch issues early, and give us the freedom to adjust as we learn more about what our users would want. In the end, we will have a solid, interactive globe that shows live ISS orbits, real time launch updates, and a fun way to search for a specific satellite that you want to learn about.

Part 5: Work Breakdown Structure (Bidisha)

Issue No.	Issue Title	Priority	Depends On
1	Initialize repository & add project README	High	N/A
2	ReactJS project skeleton	High	#1
3	Install and configure Material UI and Globe.GL/ThreeJS libraries	High	#2
4	Global layout component (header, footer, main container)	High	#2,#3
5	Integrate Launch Library II “launches” endpoint (Feature 1)	High	#3,#4
6	Integrate Launch Library II “locations”, “agencies”, “astronauts” & “spacecraft” endpoints (Feature 2)	High	#5
7	Integrate Launch Library II “events” endpoint (Feature 3)	Medium	#5
8	Integrate Celestrak TLE feed for 100 brightest satellites (Feature 1)	High	#3,#4
9	Implement time-slider for single-satellite orbit playback (Feature 2)	Medium	#8
10	Add satellite search bar and selection logic (Feature 3)	Medium	#4
11	Fetch NASA APOD & display “Picture of the Day” (Feature 1)	Low	#4,#3
12	Fetch NASA DONKI space-weather events (Feature 2)	Low	#11
13	Implement NASA Earth imagery drag-and-drop selector (Feature 3)	Low	#11
14	Integrate Open-Notify ISS location and “People in Space” endpoints	Low	#4
15	Write unit & integration tests for API modules	High	#5,#8,#11, #14
16	Write end-to-end UI tests & CI pipeline configuration	Medium	#15
17	Project documentation and deployment scripts	Medium	#1,#2,#3, #15

Part 6: Project Schedule and Milestones (Anton)

Project Start: M1 - Report & Video (July 4 & 7)

By this point, the group should have the following completed:

- Choice finalization of 2 APIs with 3 features each
- A mid-fidelity prototype
- Choice finalization of an SDLC model (Agile/Scrum)
- A WBS of all project tasks, added to GitHub as tickets
- A feature schedule, detailing when each feature will be developed and tested
- Level 0 & 1 Data Flow Diagrams between the app and APIs
- An MVC model for all the components
- A video detailing:
    - A brief overview of the project (Sadab - 1m)
    - A high-level API/Feature overview, using personas/user stories (Anton + Bidisha 2m)
    - A quick walkthrough of the mid-fidelity prototype (Sadab - 2m)
    - A high-level DFD or MCV overview between the app and APIs (Paul - 1m)
    - Parts recorded separately, splice together at the end (Anton)

Halfway Point: M1.5 - Report & T/A Meeting (July 21-25)

By this point, the group should have the following completed:

- The basic GlobeGL interface should be fully implemented
- API #1 should be fully implemented
    - Feature #1 should be fully implemented
    - Feature #2 should be fully implemented
    - Feature #3 should be fully implemented
- API #2 should be fully planned and ready for implementation
- The CI/CD pipeline should be fully set up and configured
- Automated tests should be fully set up for API 1 features

Project End: M2 - Report + Code & Presentation (July 21-25)

By this point, the group should have the entire project completed with the following additions:

- API #1 should be fully implemented
    - Feature #1 should be fully implemented
    - Feature #2 should be fully implemented
    - Feature #3 should be fully implemented
- API #2 should be fully planned and ready for implementation

API Feature Deadlines

July 4

- Base GlobeGL UI
- API #1, Feature #1: Orbital Launch Mappings

July 11

- API #1, Feature #2: Launch Information
- API #1, Feature #2: External Events Panel

July 18

- API #2, Feature #1: Brightest Satellite Positions

July 25

- API #2, Feature #2: Satellite Search Bar & Centering

August 1

- API #2, Feature #3 Satellite Position Slide

Part 7: Risk Assessment (Bidisha)

Low:

- Reusable components aren’t covered by enough tests
    - Strategy: Set a minimum code‑coverage requirement and include coverage reporting in the CI pipeline

- Minor styling inconsistencies on different browsers
    - Strategy: Apply a CSS reset, cross-check styles across the main browsers, and automate cross‑browser testing

- Misalignment on UI/UX, which prompts redesigns
    - Strategy: Create and agree on a storyboard beforehand, and organize a design review

- Delays in getting the development environment up and running
    - Strategy: Provide a setup script and Docker container; pair-program the initial setup

- Minor version conflicts among npm dependencies
    - Strategy: Lock dependency versions in package.json and conduct an npm audit every week

Medium:

- Exceeding NASA API quota
    - Strategy: Use a monitoring dashboard to track usage, and if limits are reached, switch to a backup endpoint or cached results

- Poor performance when rendering 100+ moving objects on the globe
    - Strategy: Optimize with WebGL instancing and limit update rates

- TLE Data being inaccurate, leading to satellites being positioned incorrectly
    - Strategy: Use another source, like N2YO API, to validate TLE data and visually identify outliers

- Potential deprecation of the Open‑Notify endpoint
    - Strategy: Abstract API calls behind a service layer, plan migration to other ISS‑tracking APIs

- Rate‑limiting or downtime of Launch Library II API
    - Strategy: Store recent responses in a cache, implement exponential backoff on retries, and ensure that clear, user‑friendly error messages are displayed

High:

- Team members are not available during a critical sprint
    - Strategy: Train all members in all aspects by sharing knowledge and skills, maintain updated documentation, and have everyone review each other’s work so that other team members can easily step up and cover for unavailable members

- Identification of critical bugs during the late stages of integration testing
    - Strategy: Implement test-driven development for core features, run daily integration tests and immediately fix issues

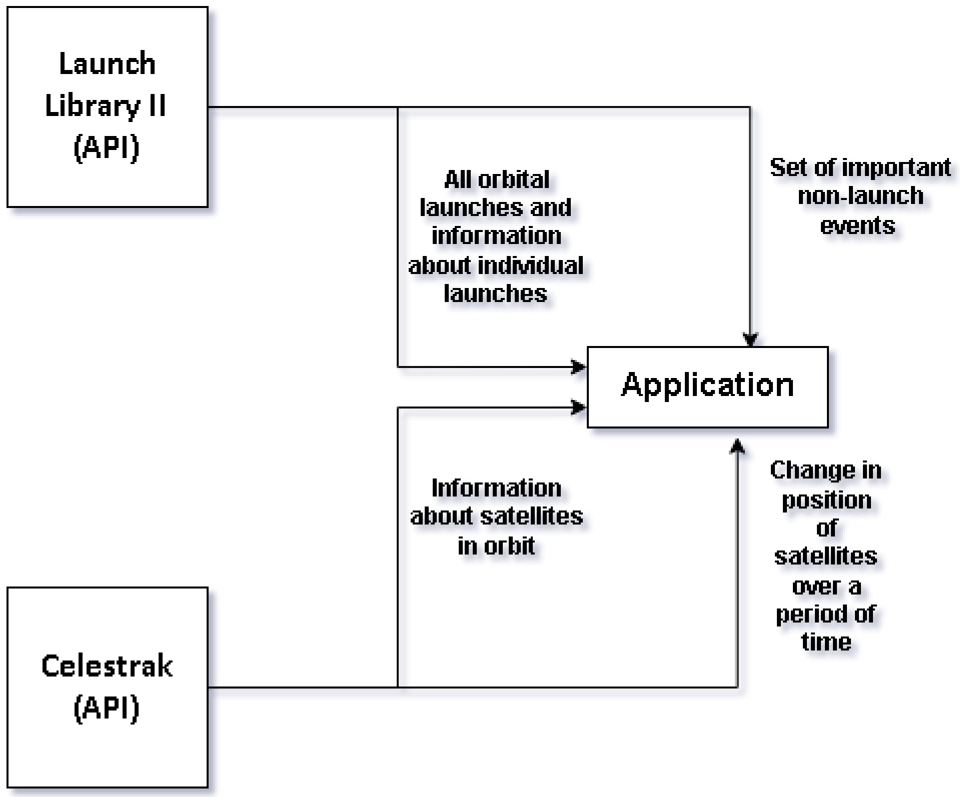
- Automated deployment pipeline breaks (downtime)
    - Strategy: Replicate the production environment in staging and monitor health checks

- More than the 6 planned features absorbed, leading to expansion of scope, which leads to timeline slippage
    - Strategy: Set strict priority levels for issues, conduct weekly backlog-grooming sessions, and lock the scope one sprint before the milestone

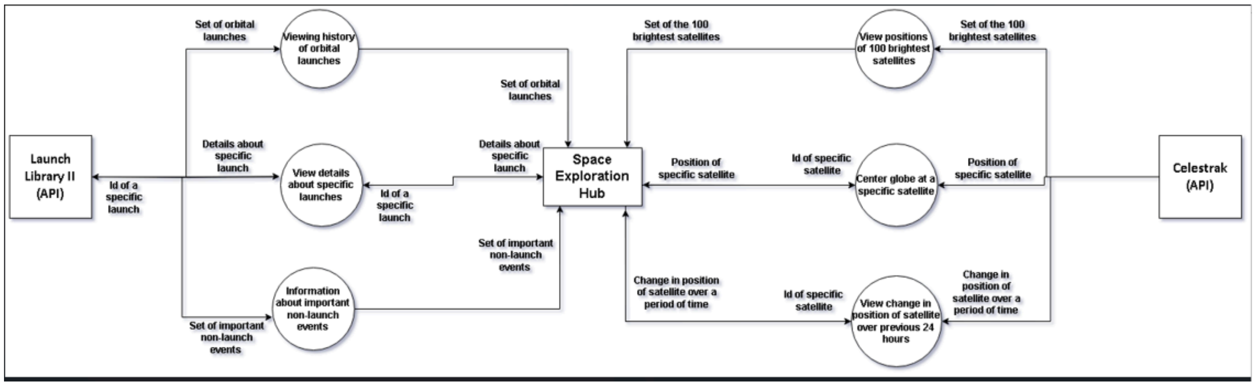
- Unvalidated API inputs create security vulnerabilities
    - Strategy: Perform client-side input sanitization

Part 8: Data Flow Diagrams (Paul)

(**LEVEL 0** Diagram)

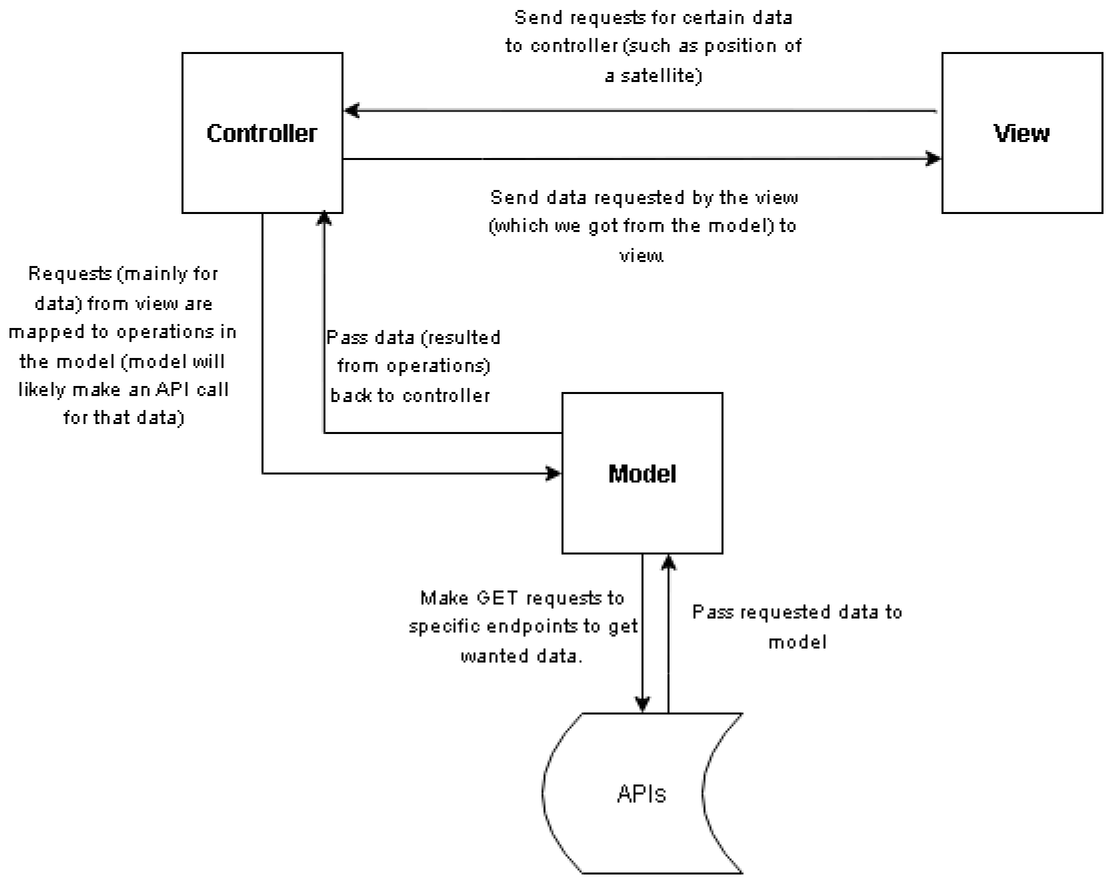

Launch Library II (API)
All orbital launches and information about individual launches
Set of important non-launch events
Application
Information about satellites in orbit
Change in position of satellites over a period of time
Celestrak (API)

(**LEVEL 1** Diagram)


Launch Library II (API)
Id of a specific launch
Set of orbital launches
Viewing history of orbital launches
Set of orbital launches
Details about specific launch
View details about specific launches
Id of a specific launch
Details about specific launch
Set of important non-launch events
Information about important non-launch events
Set of important non-launch events
Space Exploration Hub
Set of the 100 brightest satellites
View positions of 100 brightest satellites
Set of the 100 brightest satellites
Position of specific satellite
Id of specific satellite
Center globe at a specific satellite
Position of specific satellite
Change in position of satellite over a period of time
Id of specific satellite
View change in position of satellite over previous 24 hours
Change in position of satellite over a period of time
Celestrak (API)

(**NOTE :** *The original picture was large and therefore hard to display the image in the word document, I would recommend zooming in on the diagram to clearly see the details.*)

Part 9: MVC Model (Paul)


Send requests for certain data to controller (such as position of a satellite)
Controller
View
Send data requested by the view (which we got from the model) to view.
Requests (mainly for data) from view are mapped to operations in the model (model will likely make an API call for that data)
Pass data (resulted from operations) back to controller
Model
Make GET requests to specific endpoints to get wanted data.
Pass requested data to model
APIs

The view will display the UI (globe and menu options) to the user, and send requests to the controller for data when requested by the user.
The controller will handle requests from the view, mapping each request to some operation to be performed by the model, such as if the view requests data for a specific orbital launch, then the controller will ask the model to retrieve that data and the controller will also send that data back to the view.
The controller is the component that glues the view and model.
The model will contain the business logic (code that enforces rules for the data of the application) and have access to the data, in our case, data will be retrieved from API calls, or sometimes stored locally.
For each request, the model performs some operations and then sends the results back to the controller.
The model will have no direct reliance on the controller in any way because the controller should be called functions of the model which return some result.

Our biggest reason for using MVC architecture is minimizing dependencies. The view and model(the heart of the application) should not rely on any other component (the view will call controller functions, but nothing more), this allows the application to be more easily tested, more easily developed, and the model is reusable with other views and controllers.