|  |
| --- |
| **NASA API Web Application Documentation** |
| [IT21302862-Sri Samadhi L.A.S.S]  ] |

|  |
| --- |
|  |

****

|  |
| --- |
| NASA API REPORT |
| Setup the Environment  # React + Vite  This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.  Currently, two official plugins are available:  - [@vitejs/plugin-react](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react/README.md) uses [Babel](https://babeljs.io/) for Fast Refresh  - [@vitejs/plugin-react-swc](https://github.com/vitejs/vite-plugin-react-swc) uses [SWC](https://swc.rs/) for Fast Refresh  Step 1: Install Node.js and npm  Ensure you have Node.js and npm installed on your machine. You can download and install them from the official Node.js website.  Step 2: Install Dependencies  Use npm to install the project dependencies. Navigate to the root directory of your project and run:  ```  npm install  ```  Step 3: Run the Development Server  After installing the dependencies, you can run the development server using the following command:  ```  npm run dev  ```  React Component Testing with Jest and React Testing Library  This project utilizes Jest and React Testing Library for testing React components. Below are the instructions on how to run the test cases.  Installation  Make sure you have Node.js and npm installed on your machine.  ```  npm install  ```  Running Tests  Run All Tests  To run all the test cases, use the following command:  ```  npm run test:all  ```  This command will execute all the test suites available in the project.  Run Specific Test  To run a specific test file, provide the path to the test file after the npm run test:specific command.  For example, to run the SignIn.test.js test file located in the src/test directory:  ```  npm run test:specific src/test/SignIn.test.js  ```  Run Tests with Coverage  You can also generate a test coverage report by running:  ```  npm test -- --coverage  ```  Astronomy Picture of The Day (APOD) API |
| Overview The Astronomy Picture of The Day (APOID)API structures the APOD imagery and associated metadata so that it can be repurposed for other applications. The application allows users to retrieve and view images and information provided by NASA's APOD API. Implemented Features  * Search APOD images by date, date range, or count. * Display APOD images with titles and dates. * View detailed information about each APOD image in a modal.  How the Components Arranged & How to use  * The application provides a user interface to interact with the APOD API. * Users can select search options such as date, date range, or count, and input corresponding values. * Clicking the "Search" button fetches and displays APOD images based on the selected search criteria. * Users can click on individual APOD images to view detailed information in a modal.  API Requests  * **APOD API Endpoint:** **https://api.nasa.gov/planetary/apod** * **HTTP Request:** **GET** * **Query Parameters:**   + **date**: The date of the APOD image to retrieve.   + **start\_date**: The start of a date range.   + **end\_date**: The end of the date range.   + **count**: The number of randomly chosen images to return.   + **thumbs**: Return the URL of video thumbnails.   + **api\_key**: API key for expanded usage (default: **DEMO\_KEY**). |

### API Requests

* **APOD API Endpoint:** **https://api.nasa.gov/planetary/apod**
* **HTTP Request:** **GET**
* **Query Parameters:**
  + **date**: The date of the APOD image to retrieve.
  + **start\_date**: The start of a date range.
  + **end\_date**: The end of the date range.
  + **count**: The number of randomly chosen images to return.
  + **thumbs**: Return the URL of video thumbnails.
  + **api\_key**: API key for expanded usage (default: **DEMO\_KEY**).

### Implementation Details

The application is built using React and utilizes the Ant Design and Material-UI libraries for UI components. It communicates with the APOD API using Axios for making HTTP requests.

**Components:**

1. **PictureOfTheDay Component:**
   * Provides the main interface for interacting with the APOD API.
   * Allows users to select search options and input values.
   * Displays search results and APOD images.
2. **APODImage Component:**
   * Fetches APOD data based on search criteria.
   * Displays APOD images with titles and dates.
   * Handles modal display for detailed image information.

|  |
| --- |
| Mars Rover API |

****

# Overview

The Mars Rover Photos API provides access to image data collected by NASA's Curiosity, Opportunity, and Spirit rovers on Mars. This API allows developers, educators, and citizen scientists to easily access and utilize Mars rover image data. The API is maintained by Chris Cerami.

## Implemented Features

Photos can be queried based on Martian sol (rotation or day) or Earth date. Additionally, results can be filtered by the camera with which the photo was taken. Each query returns a maximum of 25 photos per call, with pagination support for larger result sets.

## API Requests

Photos can be queried based on Martian sol (rotation or day) or Earth date. Additionally, results can be filtered by the camera with which the photo was taken. Each query returns a maximum of 25 photos per call, with pagination support for larger result sets.

**Querying by Martian Sol**

**Endpoint:** **/rovers/{rover\_name}/photos**

* **Parameters:**
  + **sol** (int, required): Martian sol of the photo.
  + **camera** (string, optional): Abbreviation for the camera (e.g., FHAZ, RHAZ).
  + **page** (int, optional, default: 1): Page number for paginated results.
  + **api\_key** (string, required): API key for expanded usage.
* **Example Queries used:**
  + Retrieve photos for a specific sol:

**GET https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api\_key=DEMO\_KEY**

* + Retrieve photos for a specific sol and camera:

**GET / https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&camera=fhaz&api\_key=DEMO\_KEY**

* + Retrieve photos for a specific sol, paginated:

**GET https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&page=2&api\_key=DEMO\_KEY**

**Querying by Earth Date**

**Endpoint:** **/rovers/{rover\_name}/photos**

* **Parameters:**
  + **earth\_date** (YYYY-MM-DD, required): Earth date corresponding to the Martian sol.
  + **camera** (string, optional): Abbreviation for the camera (e.g., FHAZ, RHAZ).
  + **page** (int, optional, default: 1): Page number for paginated results.
  + **api\_key** (string, required): API key for expanded usage.
* **Example Query used:**
  + Retrieve photos for a specific Earth date:

**GET https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?earth\_date=2015-6-3&api\_key=DEMO\_KEY**

# Test Cases

* **Setting Up Jest:** Configuring Jest for testing React components and utilities required understanding Jest's configuration options and integrating it with the project's build setup.
* **Writing Test Cases:** Creating effective test cases for React components, including mocking dependencies and testing UI interactions, required a thorough understanding of Jest and React testing library.
* **Integration Testing:** Testing interactions between components and ensuring proper state management across the application posed challenges in integration testing.

# Challenges Faced with Endpoint Data and Responsiveness

**Handling Large Data from Endpoints**

* Performance Impact: Loading large amounts of data from endpoints can impact the performance of the application, leading to slower loading times and increased memory consumption.
* Optimization: Optimizing data fetching and processing methods is crucial to mitigate performance issues. Techniques such as pagination, lazy loading, and data caching can help improve the user experience.
* Server-Side Solutions: Collaborating with backend developers to implement server-side optimizations, such as optimizing database queries and implementing caching mechanisms, can alleviate the burden on the client-side application.

**Responsive Design Challenges**

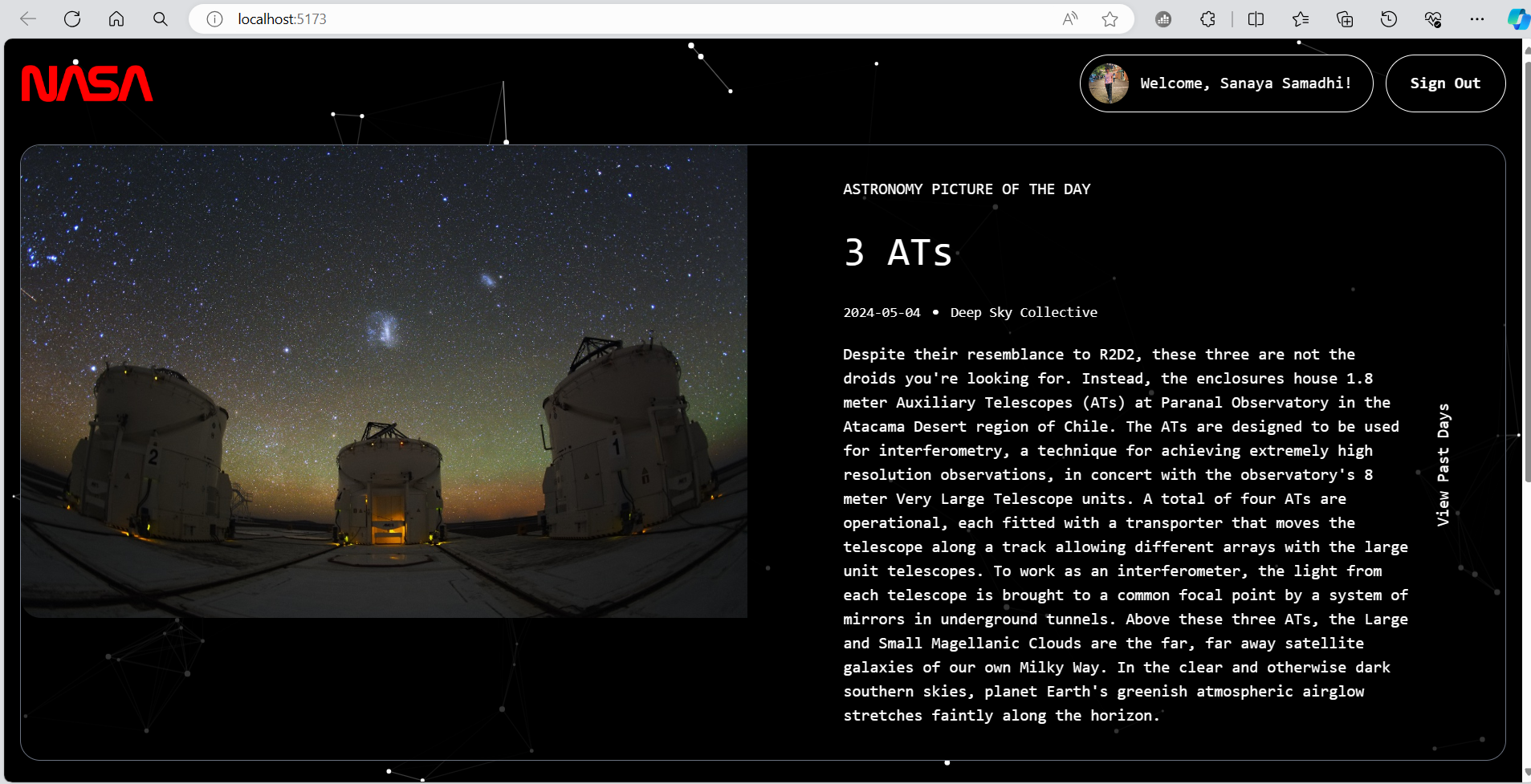
* Variety of Devices: Ensuring that the application is responsive across a wide range of devices, including desktops, tablets, and mobile phones, presents challenges in terms of layout and content organization.
* Viewport Compatibility: Designing layouts that adapt seamlessly to different viewport sizes and orientations requires careful consideration of CSS media queries and responsive design principles.
* Testing Across Devices: Testing the application's responsiveness across various devices and screen sizes is essential but can be challenging due to the wide array of device configurations and resolutions.
* UI/UX Consistency: Maintaining consistency in UI/UX design elements across different screen sizes while ensuring optimal user experience on each device poses a challenge in responsive design implementation.

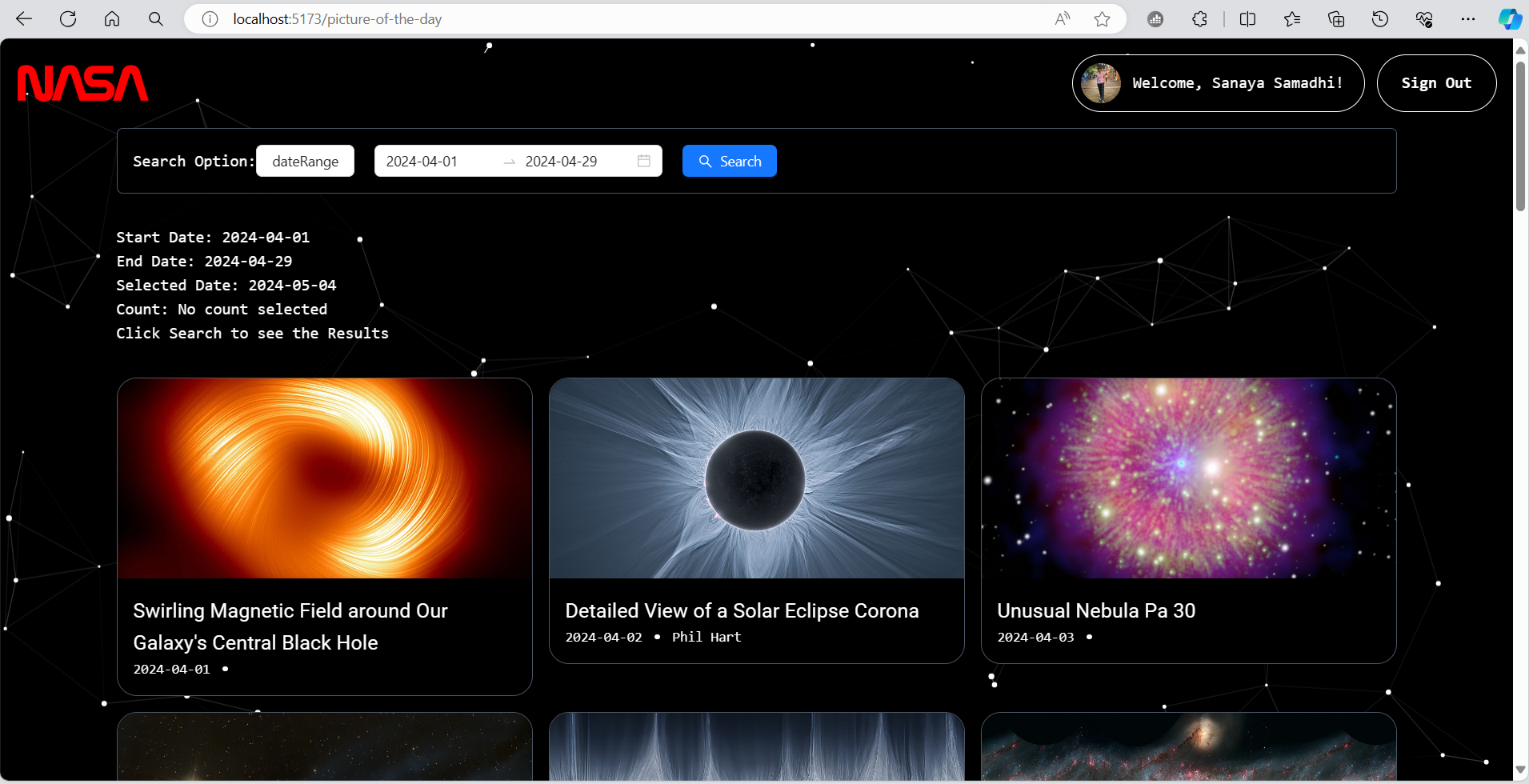
# Resolving Challenges with Endpoint Data and Responsiveness

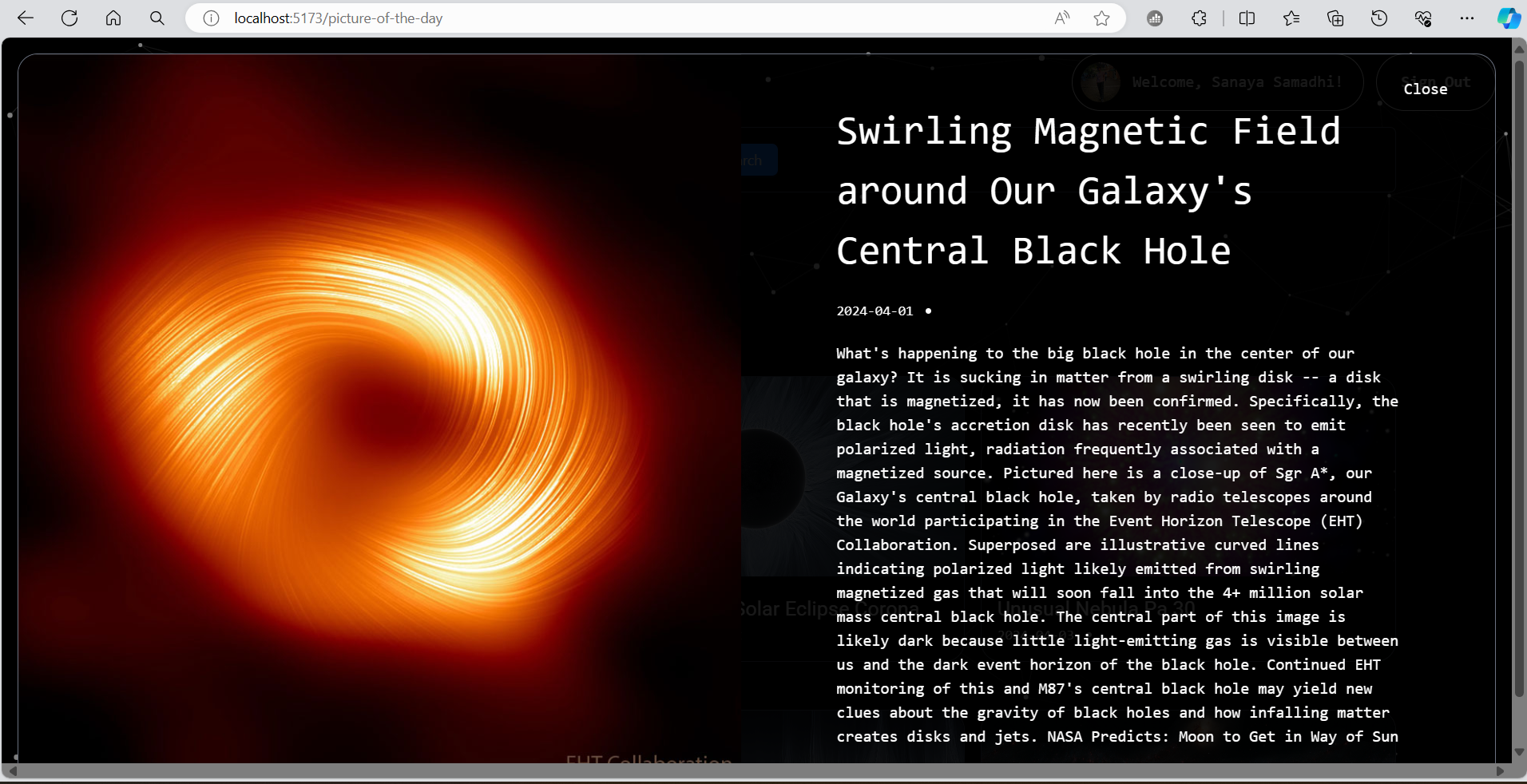
**Responsive Design Solutions**

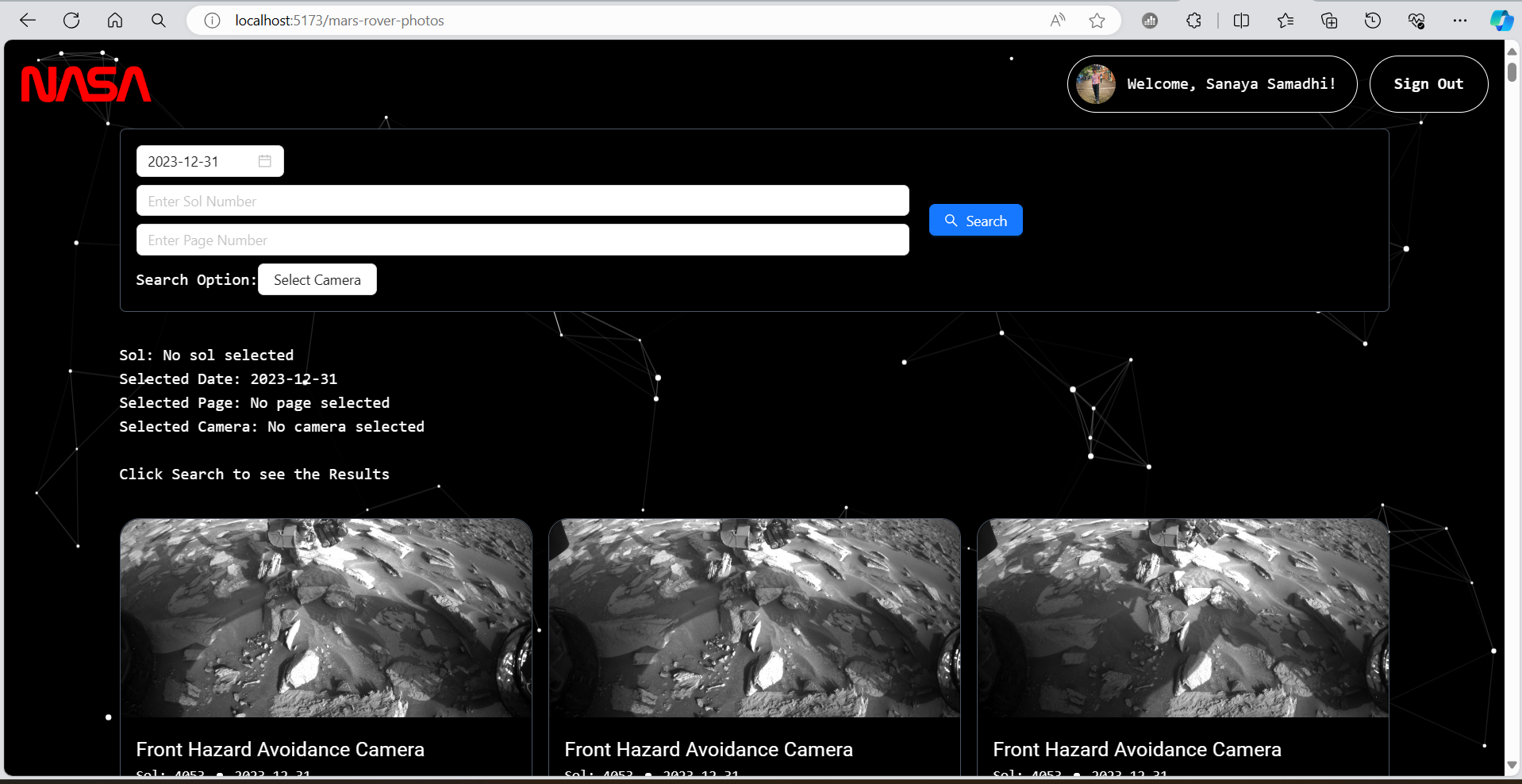
* Viewport Testing: Employed browser developer tools to test the application's responsiveness across a range of devices and screen sizes, identifying and addressing layout issues.
* Tailwind CSS (TSS): Tailwind CSS was utilized for the styling and layout of the website. Tailwind CSS is a utility-first CSS framework that offers a wide range of pre-designed classes, allowing for rapid development and easy customization of responsive user interfaces.

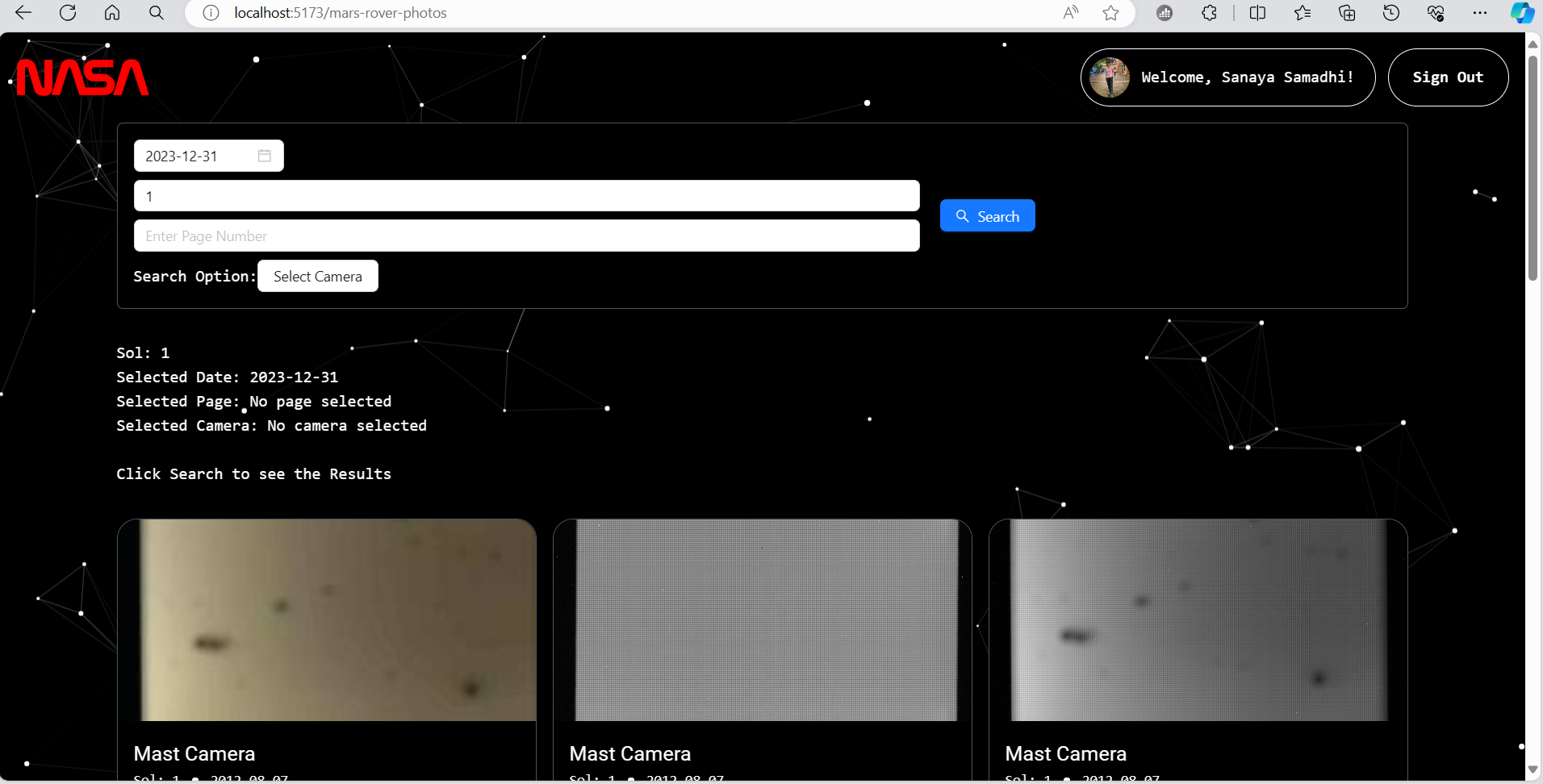
# Screenshots of the developed web application

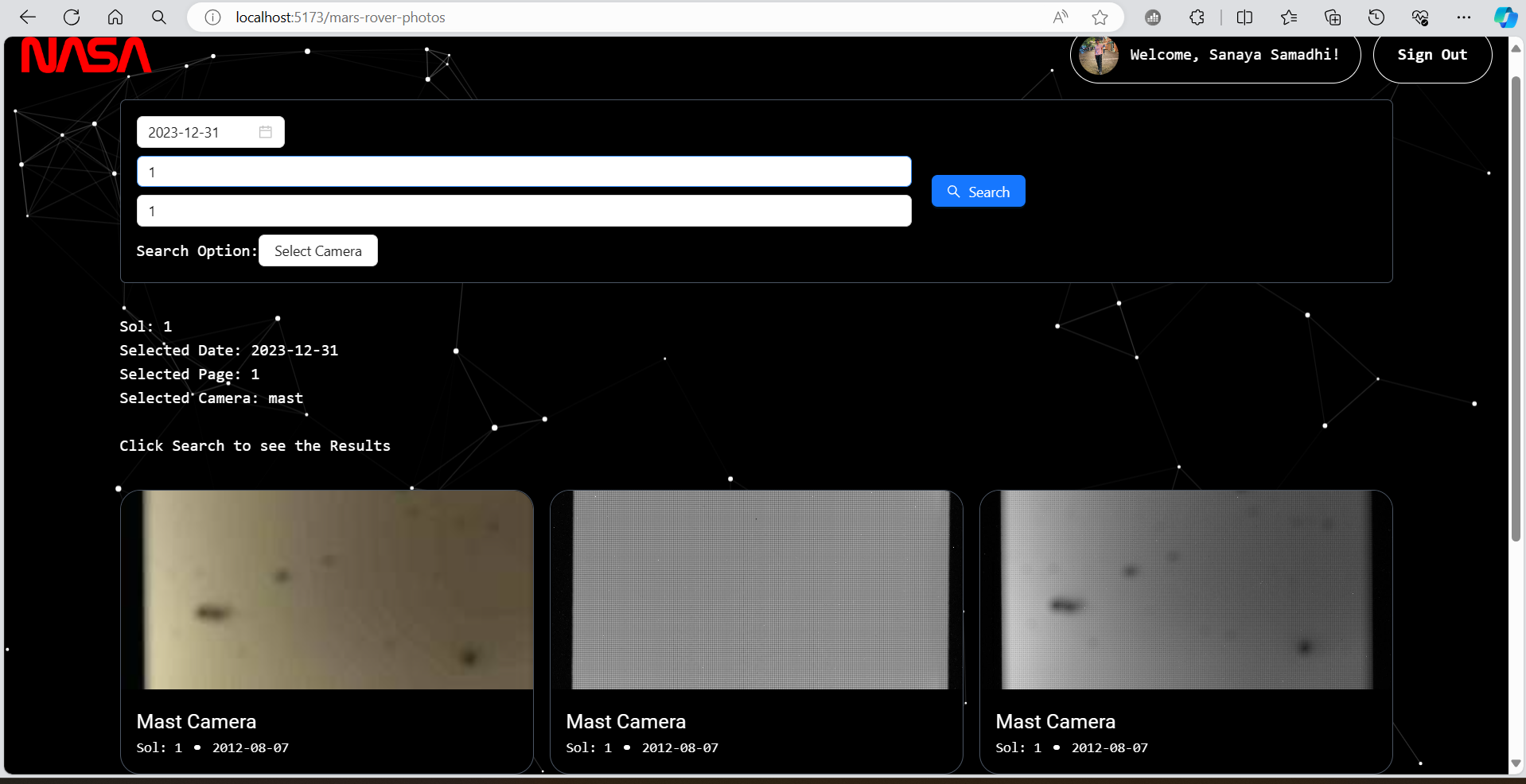
****

****

****

****

****

****