FACULTY OF COMPUTING

SE3040 – Application Frameworks
BSc (Hons) in Information Technology
Specialized in Software Engineering
3rd Year

NASA API Integration Documentation for Astro Fetch

Introduction

This documentation provides guidance on integrating various NASA APIs with Astro Fetch React App. The APIs covered include:

- Astronomy Picture of the Day (APOD)
- Mars Rover Photos
- Mars Rover Manifest
- Space Weather Alerts
- Satellite Positions

Prerequisites

Before integrating the NASA APIs into your React application, ensure the following prerequisites are met:

- Node.js and npm are installed on your system.
- Basic knowledge of React and JavaScript.

Getting Started

1. **Set Up a React Application**: If you haven't already, set up a React application using Create React App or any other method of your choice.
2. **Install Dependencies**: Install the necessary dependencies for making HTTP requests and managing state.
3. **API Key**: Obtain an API key from NASA API's official website. This key will be used to authenticate your requests.

SE3040 – Application Frameworks
BSc (Hons) in Information Technology
Specialized in Software Engineering
3rd Year

Integrating APIs

1. Astronomy Picture of the Day (APOD) with favorite session

- **Endpoint**: **GET https://api.nasa.gov/planetary/apod**
- **Description**: Retrieves the Astronomy Picture of the Day along with relevant details.
- **Usage**: https://api.nasa.gov/planetary/apod?api_key=Demo_Key

2. Mars Rover Photos with favorite session, querying, search and filtering

- **Endpoint**: **GET https://api.nasa.gov/mars-photos/api/v1/rovers/{rover}/photos**
- **Description**: Retrieves photos taken by the Mars rovers with facility to search and filter by rover name, camera name and earth dates.
- **Usage**:

```
'https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos',
     {
       params: {
         earth_date: `${earthDateRange.start}/${earthDateRange.end}`,
         page: currentPage,
         api_key: 'WtewiVSRwqmGbWS61tM0Fod967so34mrT9U4xkAg',
       },
     }
```

3. Mars Rover Manifest

- **Endpoint**: **GET https://api.nasa.gov/mars-photos/api/v1/manifests/{rover}**
- **Description**: Retrieves the mission manifest for a Mars rover.
- **Usage**:

```
https://api.nasa.gov/mars-photos/api/v1/manifests/Curiosity',
     {
```

params: {

api_key: 'Demo_Key',

4. Space Weather Alerts

- **Endpoint**: **GET https://api.nasa.gov/DONKI/notifications**
- **Description**: Retrieves space weather alerts issued by NASA.
- **Usage**: https://api.nasa.gov/DONKI/notifications?api_key=Demo_Key

5. Satellite Positions

- **Endpoint**: **GET https://api.nasa.gov/ISS-Tracker/Space_Station/TrackThisCraft**
- **Description**: Retrieves the current position of satellites.
- **Usage**: https://api.nasa.gov/ISS-Tracker/Space_Station/TrackThisCraft

Challenges Faced & Solutions

1. **Fetching the data and load those into a user interactive manner**
   - Get the json and look for data that are necessary for users and load them by hiding the unwanted things.
2. **In Mars rover endpoint filtering**
   - This endpoint have too many images so loading those all is a performance down. I have use react pagination to overcome that issue and load the data by manifests page attribute and filtering functionality to search by dates and rovers and cameras.
3. **API Rate Limiting**
   - NASA APIs often have rate limits to prevent abuse. Continuous requests from a client-side application can hit these limits. Implement caching strategies to minimize the number of requests sent to NASA APIs. Cache responses locally or on a server to avoid hitting rate limits. Additionally, consider implementing throttling on the client side to limit the frequency of API requests.
4. **Maintain responsiveness**
   - Maintaining responsiveness is a key challenge faced and solution is using tailwind css responsive attributes to make the responsiveness.

5. **Handling Asynchronous Data Fetching**
    - Fetching data from NASA APIs asynchronously and updating React component state with the fetched data.
    - Utilize React's built-in useState and useEffect hooks to manage component state and perform asynchronous data fetching respectively. Use libraries like Axios or the native fetch API to make HTTP requests to NASA APIs

6. **Rendering API Data in Components**
    - Displaying data retrieved from NASA APIs in React components
    - Pass the fetched data as props to child components and render it using JSX. Use conditional rendering to handle loading states or error messages while waiting for data to be fetched.

7. **Error Handling**
    - Dealing with errors such as network failures or incorrect API responses.
    - Implement error handling mechanisms using try-catch blocks or .catch() methods when making API requests. Display user-friendly error messages in the UI to inform users about any issues encountered while fetching data.