**Application Framework Assignment 2 -NASA Open API Project**

**Introduction**

This is a React application that loads data from NASA's public APIs and displays it to the user. To read the NASA API details, users must first register and then log in using their email and password. There are two separate APIs: "Picture of the Day" and "Mars Rover Photos". To use NASA APIs we want to generate an API KEY using <https://api.nasa.gov/>.

**NASA APIs**

1. **Picture of the Day**

Each day a different image or photograph of our fascinating universe is featured, along with a brief explanation written by a professional astronomer.

API - <https://api.nasa.gov/planetary/apod>

1. Mars Rover Photos

This API is designed to collect image data gathered by NASA's Curiosity, Opportunity, and Spirit rovers on Mars and make it more easily available to other developers, educators, and citizen scientists. This API is maintained by Chris Cerami.

Each rover has its own set of photos stored in the database, which can be queried separately. Several possible queries can be made against the API. Photos are organized by the sol (Martian rotation or day) on which they were taken, counting up from the rover's landing date. A photo taken on Curiosity's 1000th Martian sol exploring Mars, for example, will have a sol attribute of 1000. If instead, you prefer to search by the Earth date on which a photo was taken, you can do that, too.

Along with querying by date, results can also be filtered by the camera with which it was taken, and responses will be limited to 25 photos per call. Queries that should return more than 25 photos will be split into several pages, which can be accessed by adding a 'page' param to the query.

Each camera has a unique function and perspective, and they are named as follows:

API - <https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api_key=DEMO_KEY>

<https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?earth_date=2015-6-3&api_key=DEMO_KEY>

**Faced challenges and solutions.**

**1. React Component and Routing Issues**

Component Setup and Rendering Issues: Initially, there were problems with setting up and rendering React components correctly, which included handling routing and ensuring that components loaded as expected.

Routing with Authentication: There were specific challenges related to implementing routing with authentication checks, including redirecting users based on their authentication status.

Solutions: Through careful debugging and leveraging React Router for effective routing management, I was able to ensure that components were rendered correctly. I also implemented conditional rendering based on authentication states to manage user access.

**2. Context API and Authentication Handling**

Integration of Context API: Implementing the Context API for managing authentication states across the application posed challenges. I needed to ensure that the authentication state was correctly updated and accessible throughout my application.

Authentication Flow Debugging: Debugging the authentication flow (login functionality) was problematic, requiring thorough checks to ensure that tokens were managed correctly, preventing unauthorized access to certain routes.

Solutions: By creating a consistent context for authentication and using hooks like useContext to maintain state across the application, I managed state effectively. Debugging involved checks on token storage and retrieval, ensuring the application behaved as expected on user authentication changes.

**3. Testing with Jest and React Testing Library**

Setup and Configuration of Jest: Configuring Jest to work with the React environment, especially handling ES6 imports and other modern JavaScript features, was challenging. This included setting up Babel to work with Jest and ensuring that Jest could parse modern JavaScript syntax correctly.

Writing and Running Tests: There were issues related to writing effective test cases and ensuring that the Jest environment correctly recognized and ran these tests without syntax errors or configuration issues.

Solutions: By configuring Babel with the appropriate presets and plugins, I enabled Jest to recognize and process modern JavaScript syntax. Writing tests involved using the React Testing Library effectively to simulate user interactions and ensure component functionality.

**5. Deployment and Build Issues**

Netlify Deployment: I encountered specific problems while trying to deploy on Netlify, particularly with setting the correct base directory and ensuring that the build scripts ran correctly.

Backend Deployment on Render: Deploying the backend on Render involved dealing with server startup issues and database connectivity.

Solutions: On Render, server and database connectivity issues were resolved by configuring environment variables and ensuring correct port settings.

**6. Error Handling and Debugging**

Console Errors and Warnings: Throughout the development process, I faced various console errors and warnings that required systematic debugging to resolve, such as issues with imports, module resolutions, and deprecated features.

Solutions: Systematic debugging practices were employed, including the use of Chrome DevTools and careful examination of console outputs. This approach helped in quickly identifying and resolving issues.

**7. API Integration and Axios Usage**

Handling API Requests: Integrating and managing API requests using Axios involved challenges, particularly handling asynchronous operations and ensuring that data fetched from APIs was managed and updated correctly in the state.

Solutions: I utilized async-await syntax to handle asynchronous operations more cleanly and used React state management techniques to update the UI based on API data effectively.