This rubic breaks the API Hack into several key objectives. Each one is scored from 1 to 5.

Objective	Rate 1	Rate 3	Rate 5
User flow	Some basic functionality of the app is broken, preventing users from navigating the app.	The flow of the app is clear to most users, but allows for some undesirable results with edge cases.	The app includes a landing page that explains what the app is for and how to use it. It's consistently clear what actions the user should take. The app takes into account multiple possible user flows and plans for edge cases (like invalid searches).
Content	The app does not contain original content, or the content is unclear/unreadable. The app is cloned or tutorial driven.	The app presents data from a third party API with original content. The student only makes requests to one endpoint in one API. Usecases for the app may be unclear.	The app does something useful, original, and interesting. The student combines data from multiple sources (multiple APIs or multiple endpoints in the same API) in an original way.
Design	The app loses some basic functionality at different viewport sizes, or forces the user to scroll horizontally. The color or font choices make the text difficult to read. The overall appearnce of the app seems unfinished.	The app maintains basic functionality across different viewport sizes. The display adjusts to avoid horizontal scrolling, but some elements may look squished or displaced. The student uses colors and font that are readable for most users, but may be difficult for visually impaired users. The overall appearance of the app could use some polish, but doesn't have any glaring issues.	The app displays correctly and maintains basic functionality across different viewport sizes, on both mobile and desktop devices. The student uses high-contrast colors and appropriate font to make the app readable for users (including users who may be visually imparied). The overall appearance of the app is polished and professional.
JS architecutre	The student's JS code is messy and difficult to read. It may execute with unexpected side effects.	Much of the student's JS code is attached to a single $(document).ready() function, or is otherwise disorganized. The JS executes as expected.	The student's JS code is separated into single-purpose, reusable, clearly named functions. The student's code is readable and neat. There's consistent indentation, quotation marks, and semicolons. Variables are clearly named. Comments are used appropriately.
API integration	The app is not integrated with a third party API.	The student correctly structures the base URL, search parameters, and callback to retrieve data from one third party API. They may not plan for handling edge cases like missing data or invalid inputs.	The student correctly structures the base URL, search parameters, and callback to retrieve data from multiple third party APIs, and gets the APIs to "talk to each other". The students plans for expected edge cases.
Accessibility	The student does not implement a11y best practices. Basic requirements (like setting the lang attribute) are missing or incorrect. The app is missing ARIA live attributes, or they've been implemented on the wrong elements, rendering them ineffective.	The student attempts to implement a11y best practices, but demonstrates some confusion. Elements and role attributes may be used inappropriately. The student implements ARIA live on the correct elements, but may include some redundancy or inappropriate politeness settings.	The student implements a11y best practices when structuring their HTML. The student set the lang attribute on the HTML element and set the role attribute for any HTML5 sectioning elements. Div elements are used minimally and appropriately. The student uses semantic HTML elements when possible/appropriate. The student set the alt attribute for any images. The student implements ARIA live on the correct elements with appropriate politeness settings.