Project: Campus Movie Finder Web Application

PROMPT: *A document explaining why you chose the technology stack that you are demonstrating; I'd like the team to consider a minimum of two technology stacks (Node, Django, etc). Describe the process and reasoning the team used to select the stack, and explain why you chose it over the other. The doc should be added to your 'docs' folder on github.*

Intro: Prior to discussing the specific technology we would use, our team first had a call to sketch out the big picture implementation goals we sought to achieve with our web application. In doing so, we laid out plans around movie search functionality, API integration, and the ability for interaction between users. At the end of this meeting, we set up a time the next week to go over potential tech stacks. The first thing our team considered for the design of our website was the backend since the most important aspect of our web application would be the way it interacted with API calls for movie information. We brainstormed different tech stacks that could best implement our goals and settled on the following two choices for their overall versatility: JavaScript/Node.js and Python/Flask. Both excel as broad-based guiding decision choices that can fully implement many different layers of the stack with unified language and data formatting through JSON. In order to reach a final choice, we debated the pros and cons of each before making the final decision.

Pros and Cons of JavaScript/Node.js: Node.js is an explicitly straightforward full-stack implementation with a unifying language in JavaScript and easy API handling. A part of the reason for its simplicity comes from the fact that Node.js is a packaged combination of a JavaScript engine, a libuv platform abstraction layer, and an underlying core library that is written in JavaScript. On top of the technical elements, JavaScript/Node.js offers a high volume of resources and open source tools for reference in the creation process as a product of its open-source community and general high usage. Lastly, JavaScript/Node.js offers a strong ability to manage data and user information across an event-driven centric architecture. However, even though JavaScript/Node.js can handle high amounts of information in a distributed system, it struggles with performing complex computations and can slow down with higher demands. Fortunately, our web app has little need for intense data processing which left us to consider the

other potential flaw of JavaScript/Node.js in its single-threaded approach to processing. The ramifications of single-threaded processing means that additional asynchronous mechanisms need to be written into our calls to avoid losing the fast nature of the JavaScript engine. To summarize, JavaScript/Node.js offers a full-stack implementation meaning we would not have to worry about meshing multiple languages or frameworks, but requires additional code workarounds to some of the underlying mechanical issues.

Pros and Cons of Python/Flask: While not as expansive as Javascript/Nodes.js, Python/Flask offers exceptionally easy use and quick startup with its low cost architecture and easy to mesh components. With the simple syntax of python, its extensive development tools and community support, it is easier to use and has the ability to be scaled up for more complex applications. In addition, the flexibility of the Flask framework enables developers to have creative control of the application development since many parts of the framework are open to change and that sets it apart from other web frameworks. Given these, Flask reduces time to market and while providing ease of testing and debugging. Unfortunately, the Flask handles requests sequentially so with the more users/requests to the app the more latency will be encountered. Additionally, while Python/Flask is heavily used in the field, it lacks some of the same tools available through JavaScipt/Node.js npm install functionality. Therefore, Python/Flask offers a strong, simple, and easy set of tools to start up a web application, but suffers in accessible resources which can require tedious code additions and sequential processing which leads to latency.

Final Decision: After breaking down the specific aspects, we went over everyone's technical experience with both and how comfortable they would feel with using either. While some reflected some prior use with Python/Flask, everyone said that they had at least used JavaScript/Node.js before or had better familiarity with it. Given the compressed time frame of the project and the ease of full-stack development, we decided that if Python/Flask would have a higher learning curve and fewer resources available, it would be best to use Javascript/Node.js for the backend. Additionally, by using JavaScript/Node.js, we could use it across the full stack and simplify our overall tech demands. Therefore, the technology stack we have chosen for the implementation of our campus movie finder web application is JavaScript/Node.js.