The project specifications for this particular task were characterized by their simplicity and certain restrictions. The main directive was to create a reactive frontend using Vue.js delivered from a content delivery network (CDN), which would interface with a basic backend API built with Express.js. This backend API, in turn, would serve as the intermediary for interactions with a third-party API. The overarching aim was to develop an internet application that would function as a weather dashboard, with the potential for incorporating additional features in the future.
My background in internet application development encompasses a wide range of projects, from small-scale endeavors designed for use by small groups of friends for relatively short durations to services catering to hundreds or even thousands of paying customers. Especially with larger-scale projects, I place a significant emphasis on user experience. The choice of the development environment and framework assumes crucial importance in these contexts. Over my 8 years of experience in this domain, I have employed Vue.js on two separate occasions. Vue.js is a framework that excels in integrating with existing products, but for initiating a brand-new project, it might not have been my primary choice. Additionally, I have never incorporated CDN-provided JavaScript libraries into my projects. The rationale behind this decision is quite clear: loading an entire library dynamically for users can significantly extend the initial page load time, which ultimately results in a suboptimal user experience. Nevertheless, for this particular project, strict adherence to these specific requirements was deemed essential.
Given that this project is centred around a weather application, the focus was squarely on ensuring its ease of use and swift loading. The user interface was designed to be simple and intuitive, with an information section at the top providing a concise overview of the weather for the next five days. Vital information about today's weather, such as whether an umbrella is necessary or a rough temperature guide, was prominently featured at the top of the page, with more detailed data available below. Drawing upon my extensive experience in web development, I opted to design most of the interface using basic HTML and harnessed the power of CSS's flexbox layout for the design. CSS Flexible Box Layout (flexbox) is a highly adaptable CSS layout model that significantly simplifies the creation of responsive elements. It has been widely supported by most major browsers since 2021 and has been widely embraced by web developers for enhancing user experiences across various platforms, while simultaneously making the task of building responsive designs considerably less complex.
The business logic of the frontend for this project is characterized by its simplicity and effectiveness. All HTTP requests are funnelled through the axios library to the backend. Axios is a promise-based HTTP client for JavaScript, effectively wrapping the native libraries used by browsers or Node.js. Utilizing axios streamlines the management of these requests for developers, effectively eliminating much of the boilerplate code that's typically required for native HTTP requests. As the frontend sends requests for weather data, it processes the data received to provide valuable feedback to the user. In the event that this internet application experiences a significant uptick in user traffic, some of the calculations and data processing could potentially be offloaded to the backend, thereby reducing the client-side workload. However, with the present small dataset provided by the backend, this isn't an immediate concern.
The backend plays a pivotal role in this project, especially when it comes to handling interactions with the third-party API. The primary motivation for conducting these interactions on the backend is the safeguarding of an API key. API keys must be handled with the utmost confidentiality, yet it's all too common for them to be inadvertently exposed through unprotected repositories or, in some cases, even embedded in the client-side source code. Consequently, making third-party API

requests from a custom backend is undoubtedly the most judicious course of action, particularly in light of the stringent security constraints associated with this assignment. Additionally, certain preprocessing of third-party requests occurs on the backend to minimize the data transfer size and enhance manageability. This approach proves more efficient and practical, given that the frontend doesn't necessarily require all the data if it doesn't have a specific use for it. In the long run, this strategy paves the way for a more scalable and resource-efficient solution should the application's user base expand significantly.