**Strengths and weaknesses of the architecture**

For developing the architectural structure of our web application, we implement the server-side caching of the weather data with Python and PostgreSQL, in order to avoid accessing the OpenWeatherMap API too often. This evaluation explores the strong capabilities and unique limitations of python scripting and PSQL following server-side caching and database interaction.

The integration of server-side caching greatly improves the power of our architecture, reducing one of the main challenges of getting real-time data from external APIs. This approach reduces the frequency of calls to the OpenWeatherMap API, therefore reducing latency and improving performance. By storing the latest weather data, we ensure that users get quick responses, which demonstrates the responsiveness of the system.

Additionally, using Python scripts to interface with a PSQL database increases the strengths of our application. The versatility of Python and the reliability of PostgreSQL combine to offer a stable backend capable of efficiently managing data. This setup ensures a seamless flow of data from external API lookups to client-side presentation and demonstrates a well-organized operation.

Despite these improvements, dependence on external services such as the OpenWeatherMap API still presents a vulnerability. Such reliance introduces potential points of failure beyond our control and subjects the availability of our service to third-party reliability. Caching also reduces API calls, but must be carefully managed to avoid stale data to ensure users have immediate access to the most relevant data.

Another weakness lies in the architectural requirement for strict security practices. The transmission of sensitive data, particularly database credentials, requires strong encryption and secure access protocols. This is necessary to fortify against corruption and ensure data integrity.

In conclusion, our architecture is an example of a dynamic system with active caching and reliable data management, all of which provide user-centric performance. However, we must carefully address potential service interruptions, security vulnerabilities, and scalability requirements.