Note – This project was made for a SWE Internship Opportunity. Below, I document some design choices I made.

Scalability -

In case our app goes viral, we would want to be highly scalable. We can achieve some part of this scalability by running our code on services like AWS lamdas. But a more important aspect can be the design of our application. Below I list some design choices I made.

Caching Weather Data - Making an weather API call every time we need to send an email, can be a costly operation. We would be 1) relying on the weather API to 'work' every time 2) slowing down our critical path to send an email and 3) our scalability would be proportional to the scalability of the weather API.

Since weather data often does not change rapidly, a reasonable approach would be to cache weather data and use the cached weather data to service requests. Therefore, if we receive a million requests, we will not be calling the weather API a million times, instead we would be calling it once per each city (100 in our case). Another advantage is that, if the weather API we are using is down, we can simply use our cached data to service the request. Of course, we need to renew our cached data periodically, the app currently does it every 5 hours.

Making the service even faster - Although, the above approach greatly reduces the number of API calls, it still requires us to call the weather API if the cache data is old. In this case, before responding, we would still have to make an API call. This slows down the response time for such requests.

Another option, would be to run a async function which updates all the weather data every 5 hours. This way the service which is responsible for sending the email, need not worry about the data in the cache being old and does not need to make any API call. The tradeoff is that we might be updating weather data of cities we are not receiving requests from. I believe this tradeoff is reasonable given that we are able to respond to requests faster.

Bottle Neck –

Given that we are able to respond to requests without making API weather calls for each request. I believe the bottle neck for the application might be the database speed. This would be a scenario where we are pushing the bottle neck out of our application to other services that we are using. (I am not sure of this, and would have to test more to really identify the bottleneck).

Security –

Security for our application mainly arises from user input. Therefore we would want to validate the user input to make sure they are valid requests and to prevent against attacks like SQL injection. Although, it suffices to have these checks in the backend, we also replicate the checks on the frontend. By doing this, we can reduce the load on the backend, by rejecting invalid requests before they come to the backend. The app currently checks for valid email at both the front and back end and also prevents against SQL injection attacks by using place holders.

Re-Usability –

In the front end, the weather component can be re-used anywhere in the application (different webpages) to allow the user to send/ receive emails. The front end also has an email service, which sends a request to the backend, to send an email to a user for a particular location. This service can also be used by other components

The backend had three main services which can be used by many components.

The email service has one function that takes an email, a message and a location and sends the email with the message and the weather added. This is highly reusable by any component in the app, as it very generic.

The validation service, has a validation function which validates requests (validates email). This function can be reused by many other future components to validate requests.

The weather service, has two functions, one for updating the weather of all cities and another for generating a weather message for a particular weather. Both can be reused by other components.

Although, the current code is written in a re-usable way, I believe there is much scope for making the methods more ‘generic’ to increase re-usability. This would be future work.

Re-Inventing the Wheel?

Tasks for which I used external solution/packages .

Validation – I used JOI for email validation at the backend, and email tag in forms module in the front end to validate emails.

Sending Email – For sending email, I used the node mailer module.

Both of these packages, are very good at the tasks and have been tested, therefore it is a good choice to use them, instead of coding up our own solutions.

Usability –

The app could be made more easy and fun in many ways. For example, we could let the user select a quote or a joke that they want to send with the email. Or have the option to schedule the email to be sent at a specified time. We can also improve the UI of the app, to make it more appealing.

End-
Suggestions for improving and feedback are always appreciated !