Functional Requirements

Data Caching

Google Directions API calls are expensive, especially when they are made from a user's smartphone every time they preview their own or another user's itinerary (used for previewing the itinerary's path). This is inefficient, and ties our application's functionality to third party services. Moreover, with the insight that a small subset of popular itineraries will likely be viewed many more times than others, it's important that these itineraries in particular be stored in such a way that they can be retrieved quickly, granting a better user experience and avoiding congestion on our databases.

Proposed Solution

Firstly, instead of making Google Directions API calls when a user previews an itinerary, we propose making the API call when the itinerary is created and then storing the resulting polyline data along with the itinerary. This will result in a more seemless experience for the user, and cut costs for us.

Secondly, to manage the efficient retrieval of popular itineraries, we will leverage a caching system that can be queried by our application servers alongside the main databse. Since an itinerary can be uniquely identified by its `uid`, an in-memory key-value store such as Redis or Memcached are strong candidates. This will require that our application servers maintain metrics such as number of recent requests for a given itinerary so that they can keep the cache up to date. It will also require that our application servers make requests to both the main database as well as the cache.

Content-Delivery

As we expand our app with more social aspects, we expect that the amount of digital media *(e.g. images, videos)* that we will need to store will grow significantly. The retrieval of this type of data could be a huge burden on our servers, and runs the risk of causing long loading times for the users, harming the overall application experience.

Proposed Solution

We propose the use of CDNs (content delivery networks) for this. We know that itineraries, for example, in New York, are most likely to be viewed in New York as well. Any content related to these itineraries can thus be stored more locally so that users can access them with lower latency, and without interfering with the retrieval times of content destined for other areas.

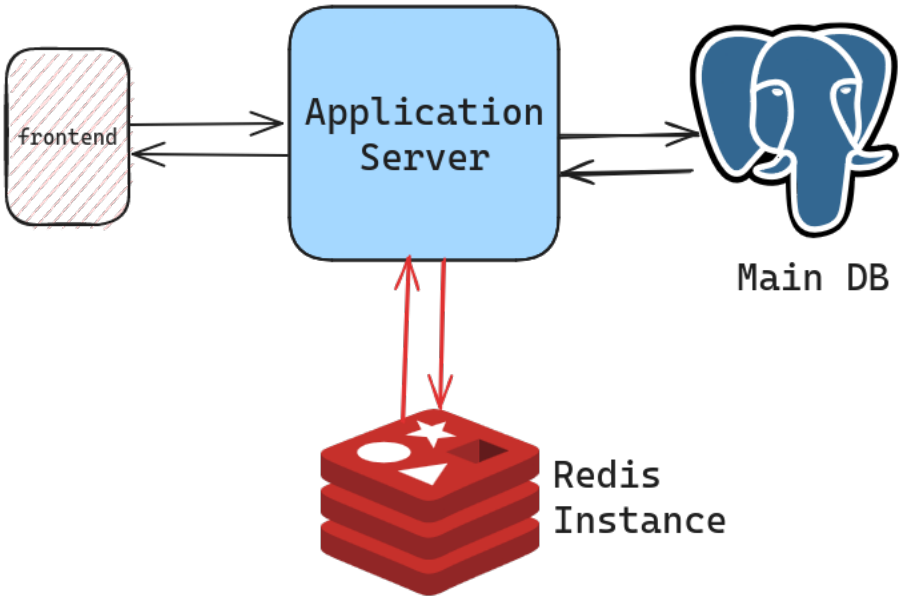

Figure 1: Data Retrieval Architecture