TravelLog: An Exercise in Scalable Cloud Data Services

To help control the spread of viral pathogens, some countries use travel log services to track the movements of infected persons and warn others who may have been exposed to them. The service maintains a database of paths. A path is a sequence of timestamped coordinate triples in three-dimensional space, capturing the movements of a person. A person may have multiple paths separated by gaps in time, e.g., if a surveilled person disables their tracking device (e.g., turns off their phone) or moves out of range of a base station.

This midterm project asks you to consider the design of such a service: TravelLog. It supports an API for use by related elements: a surveillance infrastructure, a Web interface, and a suite of analytics tools. One purpose of TravelLog is to track potential contacts of an infected person. A person $P$ is a potential contact of a queried person $Q$ if their paths have crossed: $P$ and $Q$ were in proximity within a specified distance threshold $d$ and time threshold $t$.

The TravelLog service API has four operations:

- **Update.** Log a sequence of timestamped points for the location of person $P$ (called by surveillance infrastructure).
- **Person-interval.** Return the paths of a person $P$ over specified interval of time (called by Web UI).
- **Place-time.** Return all paths for a given region of space and an interval of time (called by Web UI).
- **Contacts.** Return a list of potential contacts for a person $P$ over an interval of time (called by analytics).

Propose an elastically scalable service architecture and deployment model for TravelLog.

- What are the component services (e.g., tiers)? Outline the functions of each component service.
- How does each component service scale? Outline how to partition data and functions within each service (sharding). What keys are used for sharding?
- What are the SLOs for each operation?
- How to handle failures within each component service?
- How to distribute incoming requests across the servers?
- How to map each request to a specific pod(s) of each component service that it touches?
- What controller trigger conditions would you use to grow/shrink each component service?
- How to protect the service state against failures?
- What is the hard state that must be protected, and what soft state (e.g., caching) is present?
- What specific techniques and technologies from CPS 512 are applicable for your design, and which are not?
Your design may use standard building blocks from the course, e.g., a key-value store. You do not need to describe their internal details, but be specific and concrete about how and why you incorporate/apply them to the problem, and which optional features are important and why (e.g., transactions).

What assumptions can you make about the workload to simplify the design or make it more efficient? These assumptions might involve, e.g., the relative frequency of each operation, request patterns/locality, time or distance thresholds, path characteristics, and so on. How do these assumptions influence your design?

This exercise is open-ended. There are many difficult design questions for such a service. I am interested in your thoughts and insights about these issues even beyond the specific topics discussed. For example, you might propose a specific data structure and indexing strategy to enable fast queries and sharding. You may also have thoughts about privacy issues and how to address them. To the extent that the questions are vague, feel free to fill in the details yourself.

Your response should be a short design paper (PDF) that is readable as a stand-alone document. You may format and structure the document in your own way. Take care to address the specific questions as concretely as you can in a short document: no more than 6 pages.

References:
A travel log of the times in South Korea
https://co.vid19.sg/