Problem 3:

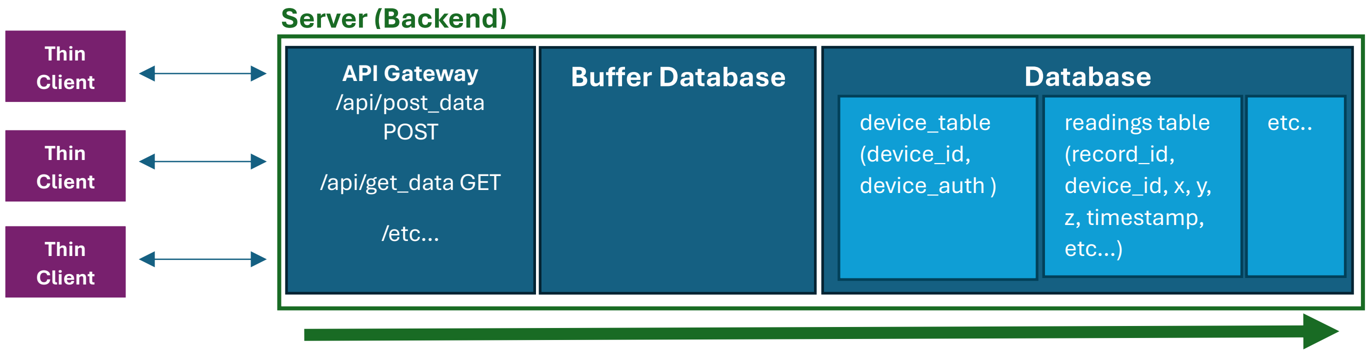
Hypothetically, we are building an app that collects accelerometer data from thin clients (sensors with limited processing capacity)

A. Assuming that the device is sending continuously at 16000Hz for the X,Y, and Z acceleration measurements, what would your strategy be in handling and processing the data? How would you design the server infrastructure? Please enumerate the steps, software, algorithms, and services that you would use to ensure that the servers can handle the incoming data from our users. Diagrams can be really helpful for this.

B. If the sensors were upgraded to modern mobile devices, how would you change your architecture from A?

Answer for A:

Server Infrastructure Overview



We're receiving continuous accelerometer data (X, Y, Z) at 16,000Hz per device which means 16,000 sets of (X, Y, Z) per second, per client. This is a high-throughput data ingestion and processing challenge. So, the goal is to design a scalable backend that can ingest, process, and store this entire process.

1. Data upload

One of the most sensitive parts of the entire operation arises from the thin clients' limited processing capacity, and the 16kHz throughput that must be matched by the server. For starters, the entire backend should be deployed as a service to ensure maximum uptime and reduce gaps in the readings. The upload of data is to be facilitated by a backend framework (**Flask** or **Springboot**) with endpoints set up to receive post requests from the clients. This brings up another key point of discussion: multiple clients, multiple requests simultaneously. A multi-threaded approach would ensure that concurrent processing is supported as multiple batches of requests are received from multiple clients.

2. Validation

The devices may also be assigned unique credentials, identifying devices from each other and sterilizing data upload. The readings must first be validated using these credentials, and once deemed authentic, can proceed to be stored in the database. However, this may impede the overall data throughput, as such, another thing that I would consider is implementing a buffer between the thin clients and the database. This is where the unvalidated readings (readings whose devices are not yet validated) would be temporarily stored until they are processed in the system.

The validation process entails checking the device_id and device_auth for each batch of data received and checking if the credentials between the data batch and the database matches. If yes, then the data would be stored into the readings table for usage.

3. Data Storage

Even though I am more adept with relational databases like MySQL, I recognize that for this type of high-frequency, timestamped sensor data, a **time-series database** such as **InfluxDB** or **TimescaleDB** would be much more optimized for storing time-series data. These database types are also optimized for read-write throughput which is a massive boon for the entire hypothetical operation.

Answer for B:

If the thin clients would be upgraded to specifications matching those of mobile phones it would be a tremendous help for the entire operation. This means other server-side functionality could be offloaded to the devices themselves. Several improvements can be made to the size of a single batch of data that can be sent. This means more data can be transmitted in a single transaction which would mean that the validation process would be able to accept more data per batch.

Another thing I would implement is the compression of the data batch done within the thin client itself. With more computational power, the data size can be reduced significantly instead of sending it raw. This would mean more data can be sent through a single transaction. However, this means that a decompression algorithm would also have to be made server side to read the data.