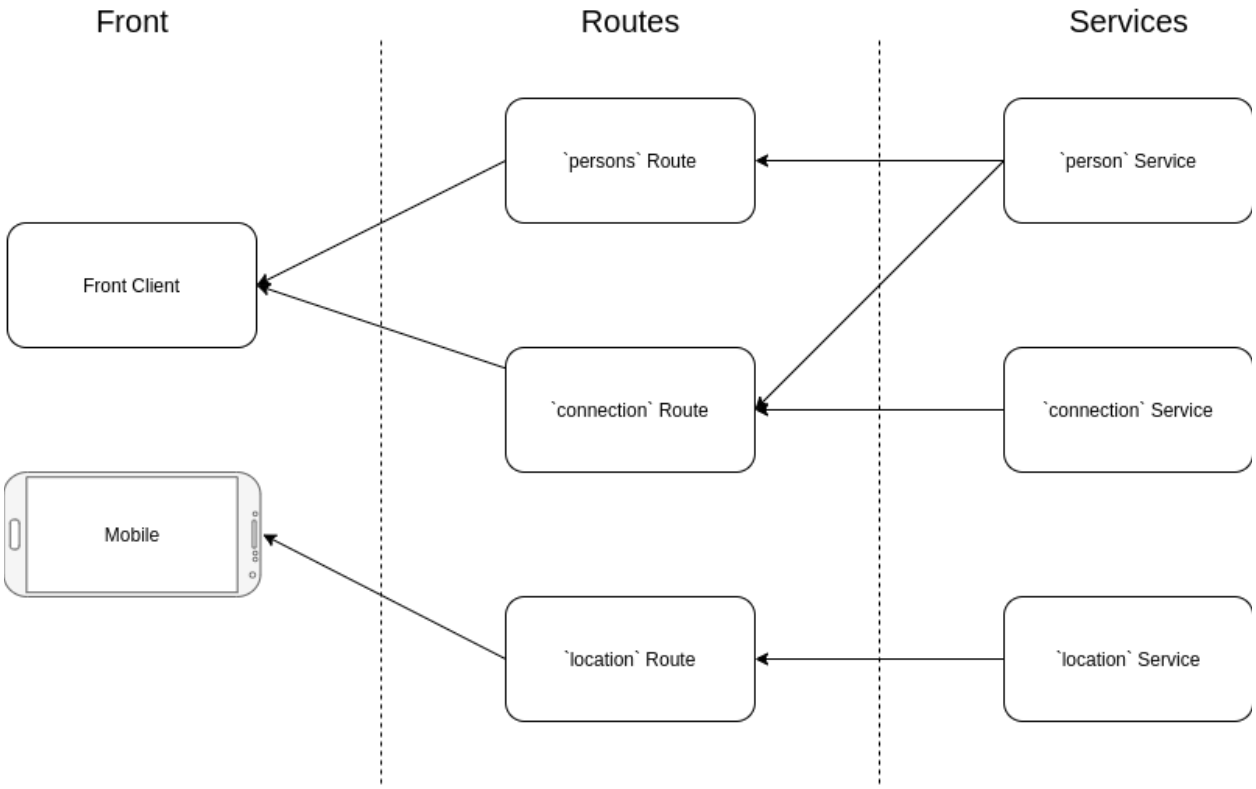
Dependency Graph

To understand the connections between the different services of the Udaconnect application we should build a dependency graph:

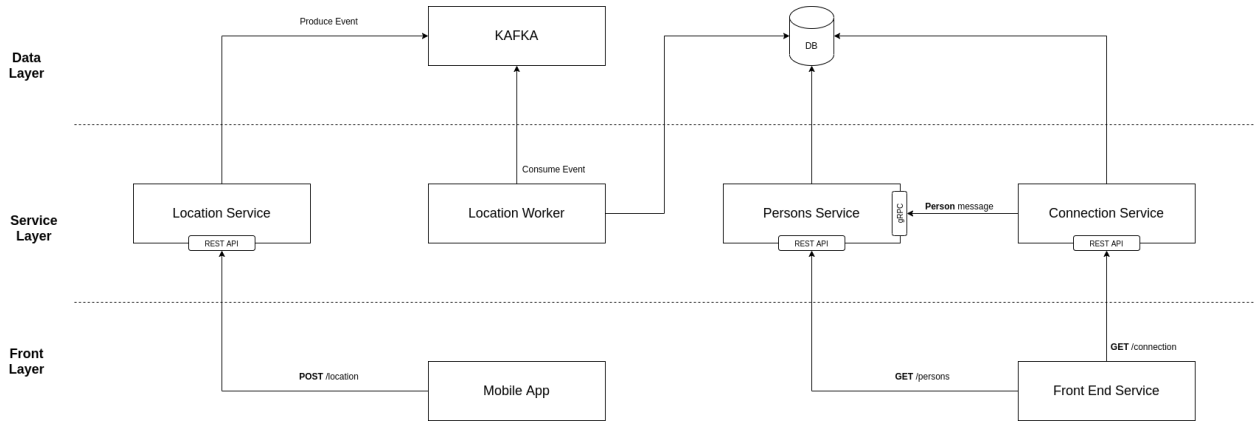


Conclusions:

- The *Front Client* is dependent on the *Persons* and *Connection* routes
- The *Persons* route is dependent on the *Person* service
- The *Connection* route is dependent on the *Person* and *Connection* services
- The *Location* route is dependent on the *Location* service

Architecture

The following diagram depicts the architecture of the Udaconnect application.



Decisions:

- The Mobile Application communicates with the Location microservice via a REST API.
- The Front End microservice communicates with the Connection microservice and the Persons microservice via a REST API.

We chose to implement these connections as REST APIs because of REST's wide adoption. REST is the current industry standard, thus, we wanted our front-facing APIs to be implemented as REST endpoints.

- The Connection microservice communicates with the Person microservice via gRPC.

We made that decision because for the internal communications we wanted to use a fast protocol, that also enforces a certain structure on our messages. Internally, it is easier to enforce and follow such practices because we have complete control.

- The Location service produces events that are stored in a KAFKA message queue. Later, these events are consumed by the Location worker, which processes them and stores the information in the database.

From the business requirements, we see that our application should expect a large volume of location data. Since there is no need for this service to be synchronous and to make it more robust, we introduce a message queue that will store these events and let the system handle them asynchronously at its own cadence.