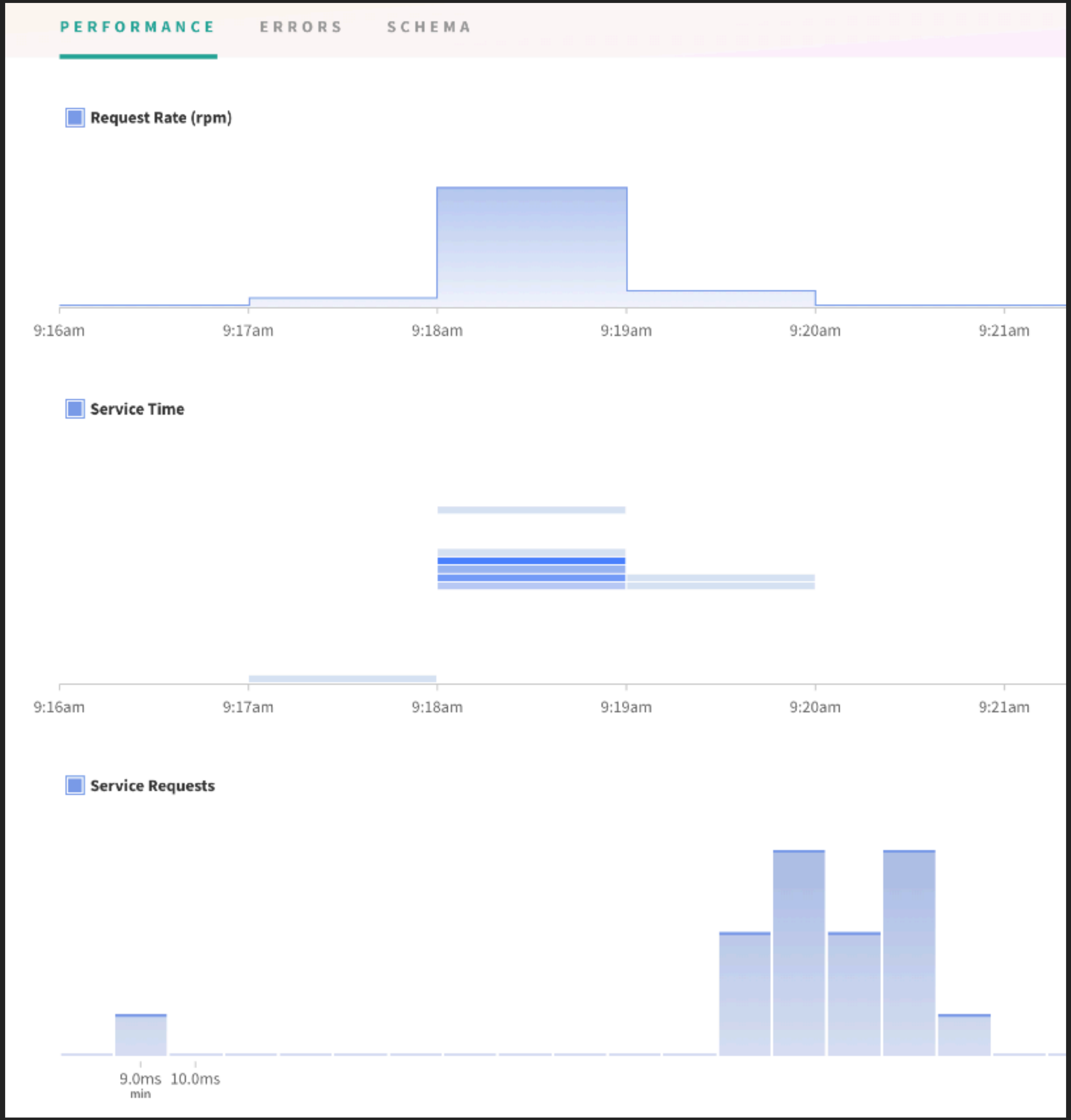
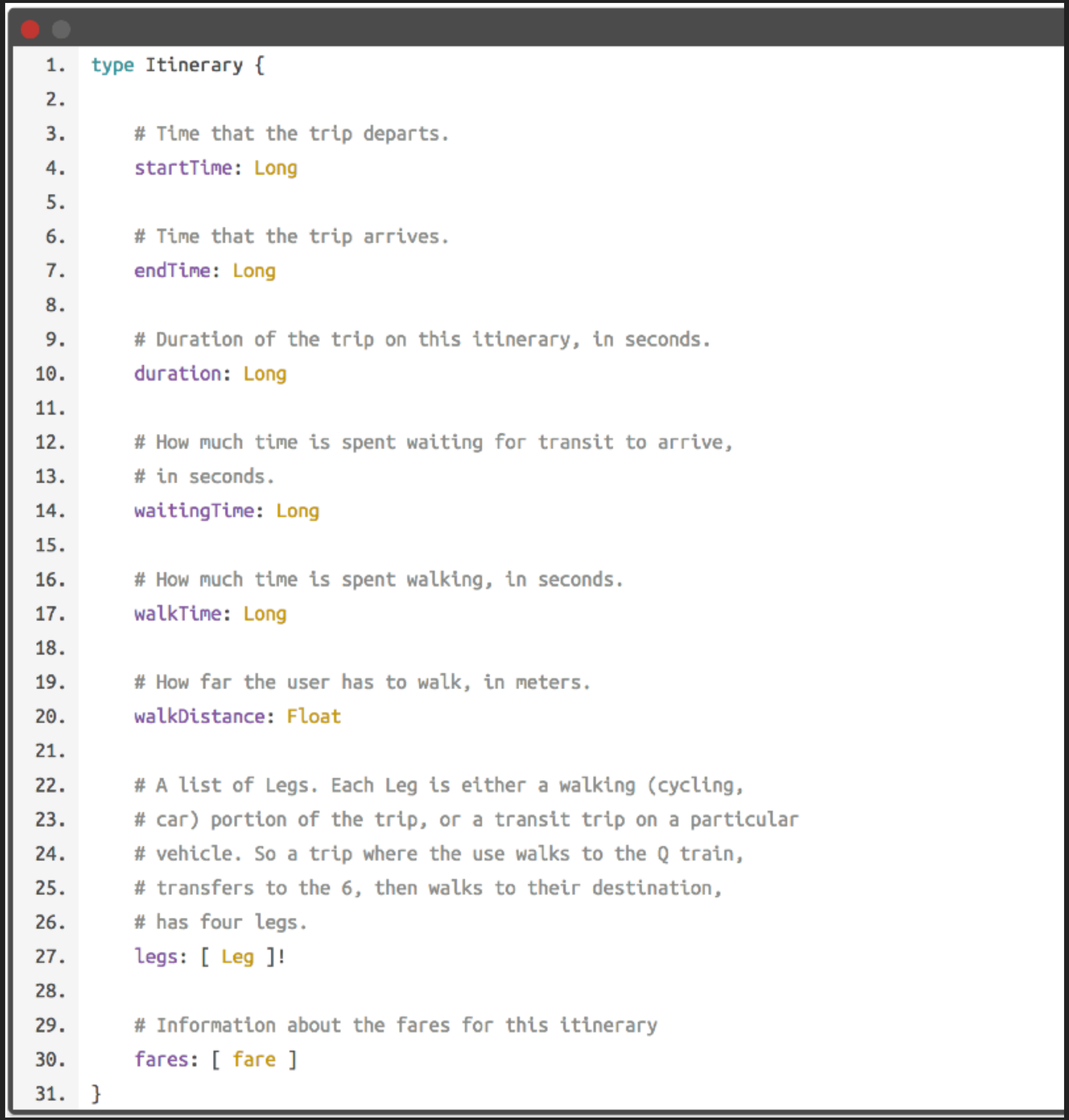


MONITORING AND PROFILING

- Tooling such as the Apollo Engine allows for detailed performance tracking, schema management, error tracking, and caching
- Errors can be tracked down to the individual field level
- Schemas can be analyzed to determine exactly how much a particular data point is used and who is using it
- Tracking demonstrates which components are placing the heaviest load on the server, and what data that component is using



```
type Itinerary {

    # Time that the trip departs.
    startTime: Long

    # Time that the trip arrives.
    endTime: Long

    # Duration of the trip on this itinerary, in seconds.
    duration: Long

    # How much time is spent waiting for transit to arrive,
    # in seconds.
    waitingTime: Long

    # How much time is spent walking, in seconds.
    walkTime: Long

    # How far the user has to walk, in meters.
    walkDistance: Float

    # A list of Legs. Each Leg is either a walking (cycling,
    # car) portion of the trip, or a transit trip on a particular
    # vehicle. So a trip where the use walks to the Q train,
    # transfers to the 6, then walks to their destination,
    # has four legs.
    legs: [ Leg ]!

    # Information about the fares for this itinerary
    fares: [ fare ]
}
```

DOCUMENTATION

- Documentation can be automagically generated from the GraphQL schema
- By automating documentation, it is ensured to be up to date
- Due to the nature of GraphQL, documentation will be simpler than it would be for REST. One only needs to know the structure and permissions of the model rather than memorizing endpoints and the data they return