QUICK SHIFT

Software Architectural Documentation

SOFTWARE ARCHITECTURE
SS G653
in
Master of Engineering (Software Systems)

Submitted By

Team 02

Name	Email
Anand Kumar (2022H1120277P)	H20220277@pilani.bits-pilani.ac.in
Manish Mahajan (2022H1120275P)	H20220275@pilani.bits-pilani.ac.in
Ramananda Samantaray (2022H1120266P)	H20220266@pilani.bits-pilani.ac.in
Srijon Mallick (2022H1120279P)	H20220279@pilani.bits-pilani.ac.in

Under the supervision of

Dr. Tanmaya Mahapatra

Assistant Professor
Department of Computer Science and Information Systems

Birla Institute of Technology and Science
Pilani, Pilani Campus, Rajasthan (India) April, 2023

Introduction

This document outlines the software architecture for Quick Shift mobile/web app built with Flutter framework and a microservices architecture. The app will consist of multiple microservices, each responsible for a specific aspect of the app's functionality.

Quick Shift is a cross platform application that provides functionalities like –

- One can book a vehicle to shift his/her furniture to a new place (in case of town shifting) or to shift products from one place to another in business aspect.
- One can book maid servant, electricians, labour and plumber from local tender using the app.

Architectural Goals:

Our objective is to divide our app's functionality into different services. Each service will take care of each high-level functionality so that performance will increase.

System Architectural Overview:

In our application we have used microservice architecture. This app will consist of the following microservices:

1. **Authorization Service**: responsible for managing user login, authentication, logout.
2. **Shift Booking Service**: responsible for handling shift/ride requests, matching riders with drivers, and tracking the status of ongoing rides/shifts.
3. **Maid/Electrician/Labour Booking Service**: responsible for managing maid, electricians' availability from a tender.
4. **Payment Service**: responsible for handling payment transactions and generating receipts.

Each microservice will have its own database and will communicate with other microservices through an http request.

1. Authorization Service:

This Service will be responsible for managing user accounts, authentication, and authorization. It will provide the following APIs:

POST /users: create a new user account.

POST /drivers: create a new driver account.

POST /tenders: create a new tender account.

POST /login: log in to an existing user account.

POST /logout: log out of an existing user account.

GET /users/{userId}: retrieve user information for a specific user.

GET /drivers/{driverId}: retrieve driver information for a specific driver.

GET /tenders/{tenderId}: retrieve tender information for a specific tender.

This Service will store user, driver, and tender account information in its own database, including email address, password (hashed), name, and phone number.

2. Shift Booking Service:

This Service will be responsible for handling ride/shift requests, matching riders with drivers, and tracking the status of ongoing rides. It will provide the following APIs:

POST /rides: create a new ride request.

GET /rides/{rideId}: retrieve ride information for a specific ride.

PUT /rides/{rideId}: update the status of an ongoing ride (e.g., to cancel the ride).

This Service will store ride information in its own database, including the rider's ID, the driver's ID (if a driver has been assigned), the pickup and drop-off locations, and the current status of the ride (e.g., assigning, processing, completed).

3. Maid/Electrician/Labour Booking Service:

This service will be responsible for handling all tender booking request like maid/electricians booking and their status, their availability etc. It will provide the following APIs:

POST /request: create a new maid request.

GET /request/{rideId}: retrieve ride information for a specific request.

PUT /rides/{rideId}: update the status of an ongoing service of a request (e.g., to cancel the request).

This service will store all the information like maid's name and phone number, electrician's name and phone number etc in its own database.

4. Payment Service:

This service will be responsible for handling all the payments done against each and every booking request (ride/shift request, maid/electrician/labour service request) and generating receipts. It will provide the following APIs:

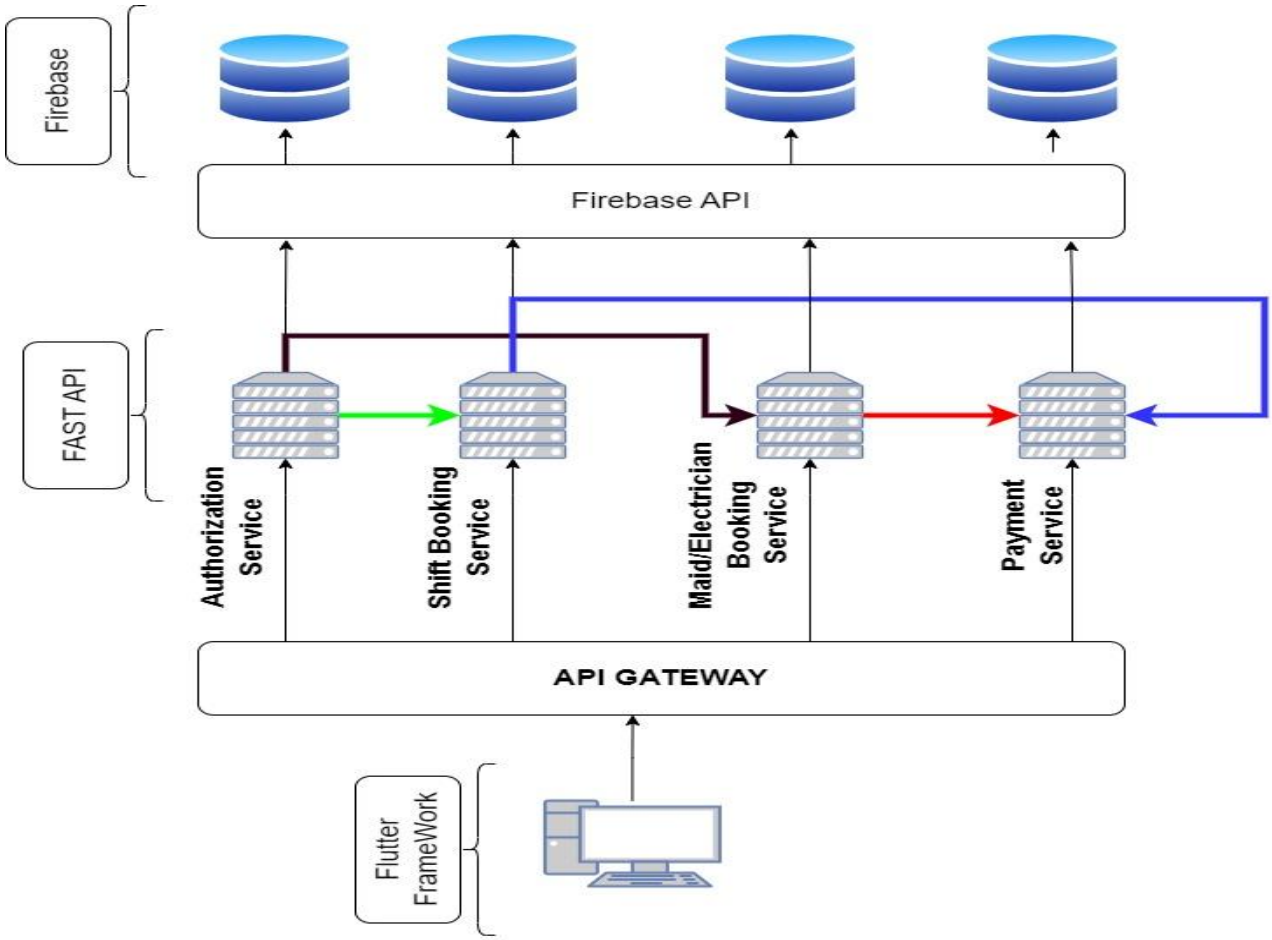
POST /payments: process a payment for a completed ride.

GET /payments/{paymentId}: retrieve payment information for a specific payment.

This service holds separate database for holding data about payment.

Design Implementation:

Our design implementation is as follows:



Shift booking service co-ordinates with payment service for each shift ride's payment status

Maid booking service co-ordinates with payment service for each bookin's payment status

Authorization service communicates with shift booking service to exchange information of assigned drivers

Authorization service communicates with maid booking service to exchange information of assigned tenders

Here we have used flutter framework as frontend, Fast API for running multiple servers in different ports and firebase for database storage.
Few services are communicating with each other for data exchange using https request (REST API).

Deployment:

As of now the backend servers or all the services are not deployed, they are running on localhost port.
Database storage is deployed on Firebase-Cloud Fire-Store

Quality Attributes:

Three quality attributes can be addressed to the architecture-

1. **Reliability**
2. **Maintainability**
3. Availability

As we have used microservice architecture in our application, so this app can be more reliable than monolithic architecture as we have separated all the major functionalities into services.

This application can be maintained easily as if maid booking service will be down for maintenance, then other services will be active.

If one service is not available to the client due to technical issues then other services should provide its service without any failure.

Future Scope:

This application can be converted to reactive architecture using observable design patterns and all the servers those are running on localhost can be deployed in cloud.