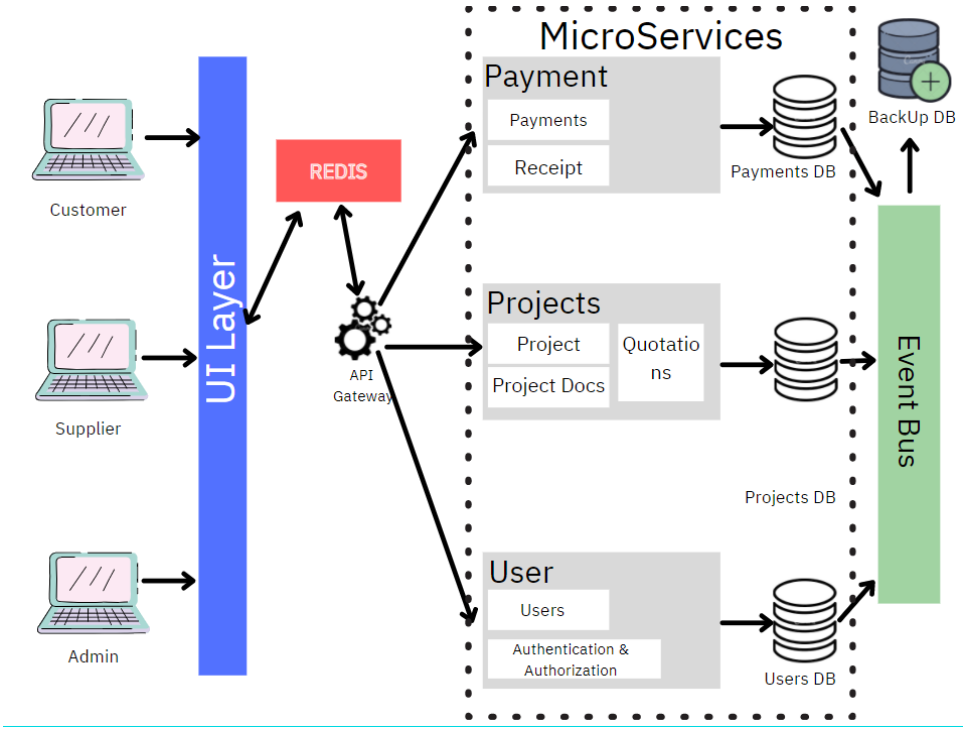
System Design



1. Architecture Pattern

The **Microservice Architecture** Style best suited to given problem. As we looking for product-oriented design the business logic can be decomposed into independent service which are loosely coupled and operate independently. In the provided problem we can have **user, project and payment** services.

- User service can handle all the user related operations (Authentication, Authorization, CRUD operation and Manage child users).
- Project service can handle the creation, updating, deletion and reading projects. It can manage all the project documents (design files) and generating and revising the project quotation.
- Payment service can handle the payment operations.

RESTful APIs are used to communicate between the services and client.

2. Technology Stack

I. Client-Side Technology Stack
   a. Programming Languages: HTML, CSS, JavaScript
   b. Framework/Library: ReactJS
   c. Automation Framework: Selenium

II. Server-Side Technology Stack
   a. Programming Language: Python
   b. Framework: Django
   c. Database: PostgreSQL
   d. API Gateway: Kong
   e. Cache and Queuing: Redis
   f. Containerization: Docker & Kubernetes
   g. Testing: pytest
   h. API Testing: Postman

3. Scalability

We can scale the application either with vertical scaling or horizontal scaling. In the vertical scaling we can increase the resources of the particular pod. In horizontal scaling we can introduce the new instances alongside the existing instance and traffic is routed among them efficient using a load balancer.

4. API Design

A. User - /user

- **a.** POST /register - register customer/supplier/child user
- **b.** GET /customers - get all customers
- **c.** GET /suppliers - get all suppliers
- **d.** POST /login - user login
- **e.** PUT /{userID} - update user
- **f.** DELETE /{userID} - delete user
- **g.** GET /{userID} - get user

B. Project - /project

a. POST /create - create project
b. PUT /{projectID} - update project
c. DELETE /{projectID} - delete project
d. GET /{projectId} - get project details
e. POST /upload - add project files
f. DELETE /{projectID}/{resourceID} - delete project resource
g. GET /{projectID}/resources - get all project resources
h. GET /{projectID}/{resourceId} - get project resource
i. POST /quotation - generate project quotation
j. GET /{projectID}/quotation - get proposed quotation
k. PUT /{projectID}/quotation - update proposed quotation

C. Payment - /payment

a. POST /make - make a payment
b. GET /{paymentID}/status - get the payment status
c. PUT /{paymentID}/status - update payment status

5. Bug Tracking

Debugging can be accomplished with efficient logging. If we follow proper logging practices and log every service. When paired with a crash reporting tool, which will show the exact line of code where error occurred making the debugging faster. We can also incorporate some monitoring tool which constantly monitors the URL and system.