System Design Document: Service-Booking-System

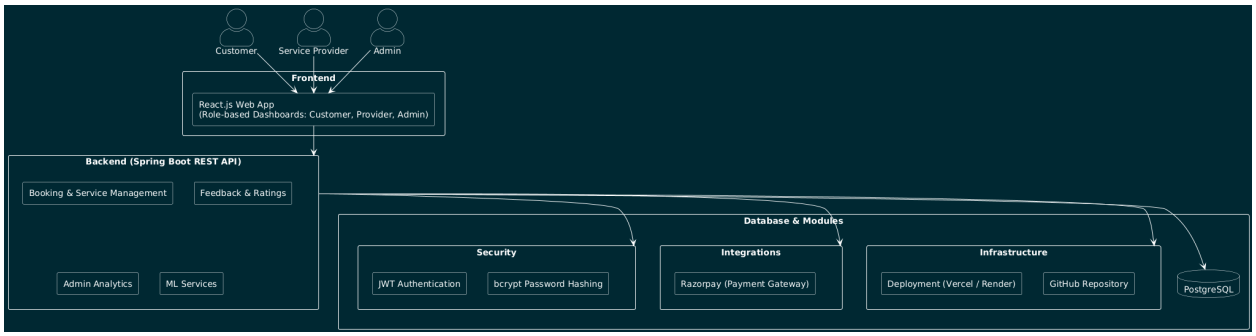
1. Introduction

This document outlines the system architecture, design, and technical specifications for the Service-Booking-System. It is intended for the development team to understand the system's components, their interactions, and the overall design principles.

2. Architecture Diagram

The system is designed using a monolithic architecture, which is separated into three main tiers: a frontend client, a backend API, and a database. This structure allows for a clear separation of concerns between the user interface, business logic, and data storage.

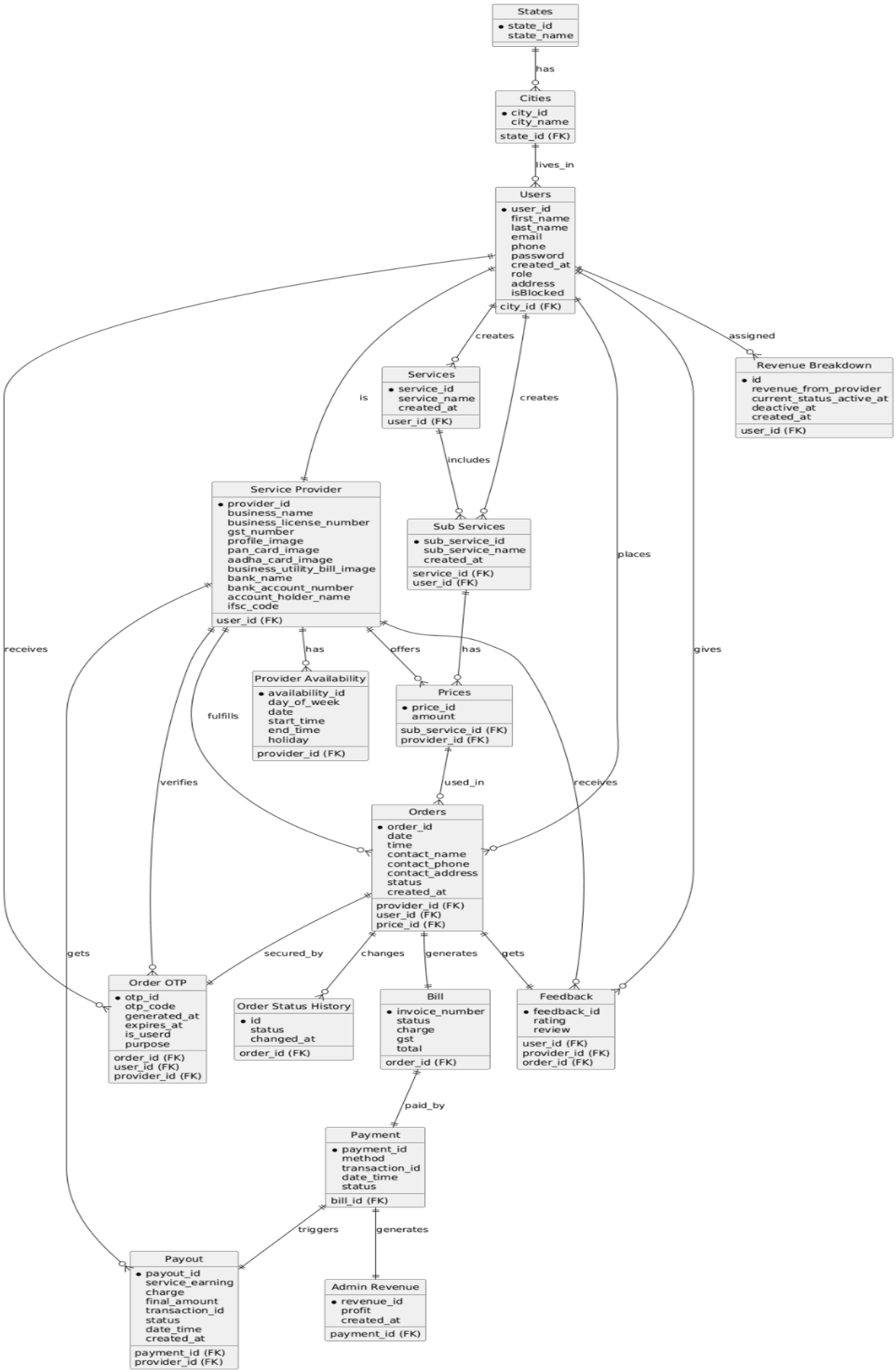
- **Frontend**: A responsive single-page application (SPA) built with React.js, providing the user interface for Customers, Service Providers, and Admins.
- **Backend**: A robust RESTful API built with the Spring Boot framework, handling all business logic, data processing, and authentication.
- **Database**: A PostgreSQL relational database serves as the persistent data store for all application data.
- **External Integrations**:
    - **Razorpay**: Used for processing online payments securely.
    - **Cloudinary**: Used for storing and managing user-uploaded images like profile pictures and business documents.
- **Security**: Implemented using JSON Web Tokens (JWT) for stateless authentication and authorization, with passwords hashed using bcrypt.



3. Database Schema (ER Diagram)

The database schema is designed to be relational, ensuring data integrity and consistency. It captures all necessary entities, including users, services, orders, payments, and their relationships.

- The core entity is `users`, which stores common information for all roles.
- The `service_provider` table extends the `users` table with business-specific details.
- `orders` are linked to a `user` (customer) and a `service_provider`.
- Each `order` is associated with a `bill`, which in turn is linked to a `payment`.
- `services` and `sub_services` define the catalog, while `prices` connect a provider to a specific sub-service with a cost.


States
state_id
state_name
has
Cities
city_id
city_name
state_id (FK)
lives_in
Users
user_id
first_name
last_name
email
phone
password
created_at
role
address
isBlocked
city_id (FK)
creates
assigned
Revenue Breakdown
id
revenue_from_provider
current_status_active_at
deactive_at
created_at
user_id (FK)
Services
service_id
service_name
created_at
user_id (FK)
is
creates
includes
Service Provider
provider_id
business_name
business_license_number
gst_number
profile_image
pan_card_image
aadha_card_image
business_utility_bill_image
bank_name
bank_account_number
account_holder_name
ifsc_code
user_id (FK)
Sub Services
sub_service_id
sub_service_name
created_at
service_id (FK)
user_id (FK)
places
receives
has
offers
has
gives
Provider Availability
availability_id
day_of_week
date
start_time
end_time
holiday
provider_id (FK)
fulfills
Prices
price_id
amount
sub_service_id (FK)
provider_id (FK)
verifies
used_in
receives
Orders
order_id
date
time
contact_name
contact_phone
contact_address
status
created_at
provider_id (FK)
user_id (FK)
price_id (FK)
gets
secured_by
changes
generates
gets
Order OTP
otp_id
otp_code
generated_at
expires_at
is_userd
purpose
order_id (FK)
user_id (FK)
provider_id (FK)
Order Status History
id
status
changed_at
order_id (FK)
Bill
invoice_number
status
charge
gst
total
order_id (FK)
Feedback
feedback_id
rating
review
user_id (FK)
provider_id (FK)
order_id (FK)
paid_by
Payment
payment_id
method
transaction_id
date_time
status
bill_id (FK)
triggers
generates
Payout
payout_id
service_earning
charge
final_amount
transaction_id
status
date_time
created_at
payment_id (FK)
provider_id (FK)
Admin Revenue
revenue_id
profit
created_at
payment_id (FK)

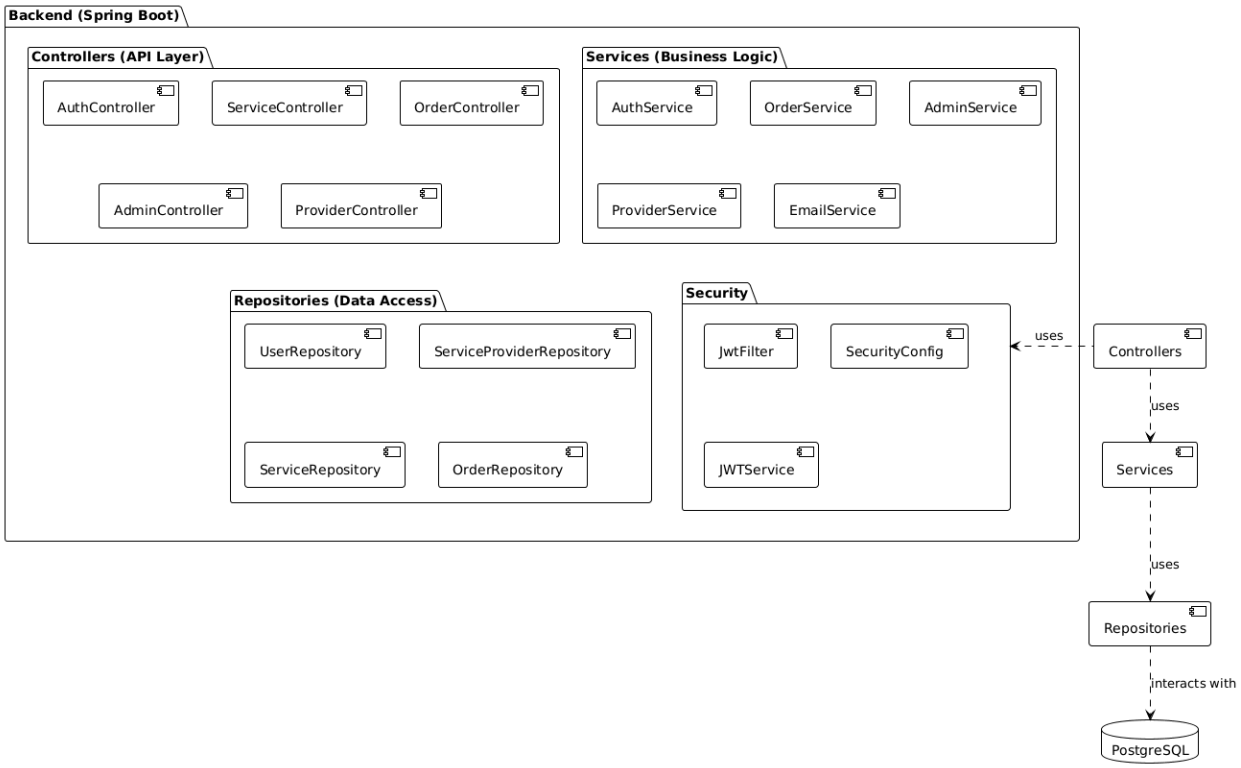
4. Class & Component Diagrams

4.1 Component Diagram

The backend is structured into logical components based on their responsibility. This modular design promotes separation of concerns and maintainability.

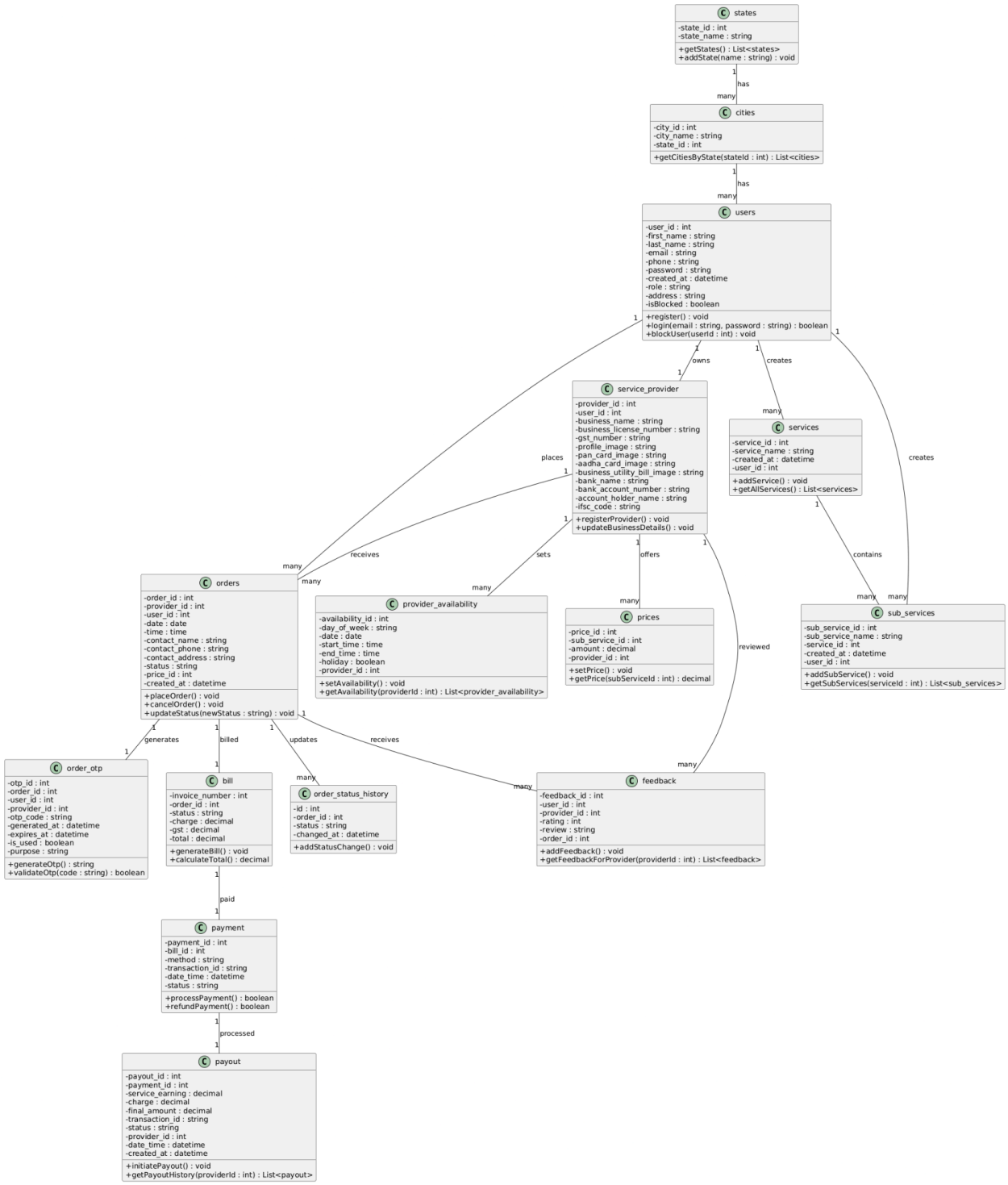
- **Controllers**: Handle incoming HTTP requests, validate input, and delegate to the service layer.
- **Services**: Contain the core business logic of the application.
- **Repositories**: Abstract the data access layer, interacting with the database via JPA.
- **Models**: Represent the data entities (e.g., `Users`, `Orders`).
- **DTOs**: Data Transfer Objects used to shape the data for API responses.
- **Config/Security**: Manages application configuration, security rules, and JWT handling.

[Placeholder for Component Diagram]



4.2 Class Diagram

The class diagram below illustrates the key domain models and their relationships, focusing on the core booking workflow. It shows how a `User` can have multiple `Orders`, each linked to a specific `ServiceProvider` and `SubService` through the `Prices` table.



5. REST API Contracts (Swagger/OpenAPI)

The REST API is the interface between the frontend and the backend. The API endpoints are documented using the OpenAPI specification in two separate files: `customer.yaml` 🔗and `admin.yaml`🔗.

- **`customer.yaml`**🔗: Defines endpoints for customer-facing actions, such as registration, login, browsing services, creating bookings, making payments, and leaving feedback.
- **`admin.yaml`**🔗: Defines endpoints for the admin dashboard, including user management (viewing customers and providers), managing the service catalog, viewing financial reports, and handling service provider requests.

These specification files provide detailed information about each endpoint, including request/response formats, required parameters, and security schemes.

6. Wireframes or UI/UX Mockups

The user interface design and wireframes for all major screens and user flows are finalized. This includes mockups for customer, service provider, and admin dashboards. The complete set of designs serves as the visual blueprint for the frontend development team.

The final mockups can be found in the project's Git repository at:
Group_10_UiUx_Designs.pdf